Teacher Student Curriculum Learning applied to Optical Character Recognition

An Analysis based on a Case Study

Rodrigo Jorgeluis Laguna Queirolo

Postgraduate program in Computer Science
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
April of 2025

Teacher Student Curriculum Learning applied to Optical Character Recognition

An Analysis based on a Case Study

Rodrigo Jorgeluis Laguna Queirolo

Master's Thesis submitted to the Postgraduate Program in Computer Science, Facultad de Ingeniería of the Universidad de la República, as part of the necessary requirements for obtaining the title of Master in Computer Science.

Director:
D.Sc. Prof. Guillermo Moncecchi

Academic director:
D.Sc. Prof. Guillermo Moncecchi

Montevideo – Uruguay
April of 2025

Laguna Queirolo, Rodrigo Jorgeluis

Teacher Student Curriculum Learning applied to Optical Character Recognition / Rodrigo Jorgeluis Laguna Queirolo. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2025.

IX, 104 p. 29, 7cm.

Director:

Guillermo Moncecchi

Academic director:

Guillermo Moncecchi

Tesis de Maestría – Universidad de la República, Program in Computer Science, 2025.

Bibliography: p. 93 – 99.

1. Curriculum Learning, 2. Teacher Student Curriculum Learning, 3. Optical Character Recognition, 4. Comparative Analysis. I. Moncecchi, Guillermo, . II. Universidad de la República, Postgraduate Program in Computer Science. III. Title

MEMBERS OF THE THESIS DEFENSE COURT

Ph.D. Prof. Lorena Etcheverry

Ph.D. Prof. Martín LLofriú

Ph.D. Prof. Federico Lecumberry

Montevideo – Uruguay
April of 2025

To all those who continue
searching for their loved ones,
seeking truth and justice.

¿Dónde están?

Acknowledgements

To my family, for their continuous support and encouragement to persevere.

To my advisor, for the always enriching discussions, and above all, for his patience and knowledge.

To Universidad de la República, for my professional formation, including this work, which would have been impossible without it.

To Khipu[1], and especially to its organizers, for the inspiration, motivation, and suggestions.

To ClusterUY: all the experiments presented in this project were carried out using their platform [2].

To my colleagues for their invaluable feedback.

Most especially to my wife, for sustaining and encouraging me in this project I embarked on a few years ago.

[1]site: https://khipu.ai
[2]site: https://cluster.uy

ABSTRACT

This thesis explores the application of Teacher Student Curriculum Learning (TSCL), a Reinforcement Learning (RL) based Curriculum Learning (CL) method, to the task of Optical Character Recognition (OCR) in a dataset from the LUISA project. The aim of the LUISA project is to develop tools for extracting information from digital images of historical documents stored in the *Archivo Berruti*, a collection of documents generated by the Uruguayan Armed Forces during the last dictatorship in the period 1968-1985.

The proposed approach uses a seq2seq model as the Student in the TSCL framework, which was trained on the same data as a previously developed model (Chavat Pérez, 2022), but with modifications to the training method. This allows for a fair evaluation on the benefits of TSCL. This work contributes to a better understanding of TSCL and its potential application to OCR. Moreover, it presents a thorough theoretical review on CL, with a special focus on RL-based methods, including TSCL, and compares its results with traditional methods.

While TSCL results show only minimal improvements in OCR performance, this work contributes to the understanding of TSCL's functioning and provides a stepping stone for future implementations in supervised tasks beyond OCR. In an effort to compare TSCL against strong benchmarks, the study also enhances Chavat's work by proposing improvements in model training with image augmentation techniques and beam search, surpassing previous metrics reported by over 16% for Character Error Rate (CER).

The code developed for this work is publicly available. Based on available information, this appears to be the first attempt to apply CL techniques, specifically TSCL, to an OCR task.

Keywords:
Curriculum Learning, Teacher Student Curriculum Learning, Optical Character Recognition, Comparative Analysis.

Contents

Chapter 1

Introduction

1.1. Motivation

Leyendo Unidos para Interpretar loS Archivos (LUISA) project aims to transcribe digitalized documents from the last military civic dictatorship in Uruguay (1973-1985) of the so called *Archivo Berruti*. LUISA is in fact part of a major project called CRUZAR which consists on "the systematization of information from archives of the recent past linked to state terrorism and serious violations of Human Rights". CRUZAR's objective is "the ordering and classification of the material, and the elaboration of a computer program that allows the crossing of the information contained in those files" ("Cruzar, sistematización de archivos militares", 2019) [1]. In this context, one of the early stages in processing this information is Optical Character Recognition (OCR): the task of extracting text from digital images. Given the huge size of the archive, this task can not be manually performed, and given its poor conservation and digitization conditions, in addition to the heterogeneous nature of the documents, available multi-purpose OCR tools such as Tesseract (Kay, 2007) do not have a satisfactory performance on it. LUISA started as a platform[2] to manually label instances by anonymous volunteers to later train specific models on this data.

Recent efforts to improve OCR performance included. a degree thesis (Chavat Pérez, 2022) exploring seq2seq models to perform this task. Despite improving the results and giving valuable insights on this specific task and

[1]Since their launch in 2019, LUISA and CRUZAR have remained active up to today, to the best of my knowledge

[2]Available at mh.udelar.edu.uy/luisa/.

dataset, there is yet room for improvement in terms of results within the task.

Our work is an attempt to improve the results shown in Chavat's work by modifying the training process, using Curriculum Learning (CL) instead of the traditional training loop.

CL methods try to improve the results by reformulating the way that Machine Learning (ML) models are trained: instead of training with data sampled uniformly at random from the training set, CL methods try to first train in the easy examples, and then, as the model masters them, move forward to harder examples. This fundamental idea is based on the observation that both, humans and animals, do learn better when knowledge is acquired following certain order (Bengio et al., 2009; Elman, 1993). These ideas were first proposed in late 1990s, in The Importance of Starting Small paper (Elman, 1993), but it was only in 2009 that Bengio coined the term *Curriculum* (Bengio et al., 2009). As humans, we have all experienced the benefits of following educational curriculums, which makes the potential of CL methods all the more enticing.

However, despite having been around for the last 30 years, and its successfully applications over several problems, from Computer Vision (CV) (El-Bouri et al., 2020; Guo et al., 2018; Hacohen and Weinshall, 2019; Karras et al., 2017; Tan and Le, 2021) to Natural Language Processing (NLP) (Platanios et al., 2019; Zhang et al., 2018; Zhou et al., 2020) among many others, (Jiang, Meng, Mitamura, et al., 2014; Soviany et al., 2022; Zhao et al., 2015), CL is not widely adopted (X. Wang et al., 2021). Therefore, another motivation in this work was to gain a comprehensive understanding of CL methods, particularly focusing on those based on Reinforcement Learning (RL).

1.2. Curriculum Learning

A *Curriculum* consists on a sequence of *concepts* following certain *order* to be learned. This *order* represents a subtle balance between easiness and challenges, as concepts that are too easy add little new knowledge while those that are too complex might make it impossible for the learner to learn (Elman, 1993). For the purposes of this work, *concepts* are defined as the sub tasks that the learner needs to learn, to actually solve a task. A task can be divided into smaller sub tasks. For instance, the task of classifying figures based on its shape can be divided into two sub tasks: classifying regular shapes and classifying irregular ones (Bengio et al., 2009). How sub tasks are defined depends on various factors, including the task being learned. Some tasks are naturally splittable into sub tasks, like RL problems. However, this work focuses only on Supervised Learning problems, where defining sub tasks is generally more challenging.

Accordingly, crafting a Curriculum involves determining the sequence of sub tasks to be learned and allocating appropriate time to each sub task. Roughly speaking, there are three alternatives for the curriculum. The first is a curriculum designed by an expert who is supposed to know how to teach the model (Bengio et al., 2009; Khan et al., 2011; Spitkovsky et al., 2010). Then, there are other approaches where the curriculum is auto-generated from the samples or its labels (Hacohen and Weinshall, 2019; Jiang, Meng, Mitamura, et al., 2014; Jiang et al., 2017; Kumar et al., 2010; Toneva et al., 2019; Tsvetkov et al., 2016; Zhao et al., 2015). Last but not least, there are some novel approaches where curriculum is learnt concurrently with the task, by using RL (El-Bouri et al., 2020; Graves et al., 2017; Matiisen et al., 2019).

RL based approaches are particularly relevant, because those methods are going to be tested in this work to validate its contribution on solving the OCR task. In particular, we are going to use Teacher Student Curriculum Learning (TSCL), a framework purposed in (Matiisen et al., 2019).

1.3. Teacher Student Curriculum Learning

In TSCL, there are two models: a Teacher, who chooses which sub task is appropriate on each step of training, and a Student, who trains on the sub tasks that the Teacher choose to actually solve the task. For the purpose of this work, the task to solve is OCR. Both, Teacher and Student, start the process without any previous knowledge, and learn on the process. The Teacher uses **RL** to learn how to choose appropriate sub tasks for the Student on each step.

Matiisen et al., 2019 propose two formulations of TSCL: one designed for RL problems and another adaptation formulated for Supervised problems. Since the OCR task we are solving is Supervised, we focus on this adaptation. The main difference among them is that, instead of letting the Teacher choose a single sub task to train at the next step, the Teacher samples instances from all the sub tasks according to a sampling function. This sampling function defines how many instances are considered from each sub task on the next step, ensuring that the Student can train on each step over examples from all sub tasks, for stability reasons. The Teacher learns the sampling function during the training time, using the feedback that the Student provides on each step. On each step, the Student receives data from the Teacher, containing instances from different sub-tasks, and uses it to perform some Stochastic Gradient Descent (SGD) steps.

In this work we use Chavat's model, with some improvements that we propose, as the Student, which is trained from scratch using this TSCL framework. The Teacher is a simple tabular representation of the sampling function, as in (Matiisen et al., 2019). At the end of the training, the Student is trained following several SGD steps, over the same data and same parameters than our base model. The main difference between the TSCL trained and the traditionally trained model, is in the training process: in TSCL, the Teacher changes the distribution of the training dataset that the Student actually uses for training. Since the model and dataset are the same, differences in performance could be attributed to the usage of TSCL.

1.4. Objectives

The main objective of this work is to validate the effectiveness of CL strategies when applied to the challenging problem of OCR, using a specific dataset introduced in Chavat Pérez, 2022. Our study will compare traditional training methods with CL, and in particular TSCL: a RL based CL method proposed in Matiisen et al., 2019.

Before actually starting to experiment with TSCL, some improvements are proposed to Chavat's work, in order to include simple techniques that are widely used: Image Augmentation and Beam Search, in addition to fixing some minor implementation details, given that we forked his implementation[1]. These improvements aim to raise the level of the starting point before implementing TSCL, so we can ensure that the comparison is against a strong base. This is especially important for us, because we are evaluating a method, TSCL, which is not widely known and has never before been adapted to seq2seq problems, using a dataset that has not been released. As a result, the only available performance reference is Chavat's work.

[1]Chavat's implementation available at https://gitlab.fing.edu.uy/felipe.chavat/prograd-luisa

The research questions for this study are the following:

- Does CL improves the results with respect to using only a traditional training strategy, in the selected OCR task?
- How can CL be used to train an OCR model?
- How and why does CL work?

1.5. Document Organization

The rest of this document is organized as follows: In Chapter 2 CL is presented. The chapter defines it formally, and presents RL based implementations, with special emphasis on the methods discussed and implemented in this work: TSCL.

Then, in Chapter 3 the OCR task chosen for this study is presented along with its proposed implementation. The chapter includes an overview of the dataset used, its construction, and the primary challenges it presents. Additionally, it introduces the proposed framing for this task in the context of TSCL.

The Chapter 4 shows the experiments and results obtained, including some error analysis and qualitative comparison among them.

Finally, some conclusions are shared on Chapter 5, and following research direction are proposed.

Chapter 2

Curriculum Learning

We want to understand if using CL while training an OCR model does improve its performance. This chapter provides an overview of such methods, starting with its origins and basic principles as proposed by Elman, 1993 and formalized by Bengio et al., 2009. Then, it covers the main alternatives in terms of CL framing, following a simplified version of the taxonomy proposed by X. Wang et al., 2021. Note that, since our OCR task is a supervised problem, we will restrict our coverage to CL instantiations designed to solve this kind of tasks: semi-supervised, self-supervised, unsupervised, or RL problems are out of scope for this work and hence will not be covered. Finally, it introduces the technique that we choose to implement in this work: TSCL.

CL methods are part of a wider family of techniques called continuation methods, which have been shown to be effective in improving generalization performance and reducing the risk of overfitting (Bengio et al., 2009).

Once this more general case is introduced, we will dive into the specifics of CL, exploring various generalizations and extensions of the basic framework, to finally discuss the specific methods and techniques that will be used in this work.

2.1. Continual Learning

Continual Learning is usually defined as the ability to learn consecutive tasks without forgetting how to perform previously learn ones, for a potentially infinite stream of data (Delange et al., 2021; Kirkpatrick et al., 2016). This definition encompasses a wide range of problems, including RL and CL (Bengio et al., 2009), among many others. In its more general sense, even a single neural network trained with SGD can be seen as a Continual Learning system, where each mini batch is a task (Toneva et al., 2019). Some other specific forms of Continual Learning includes lifelong learning, sequential learning and incremental learning (Delange et al., 2021; Mai et al., 2021).

The main goal of continual learning is twofold: to preserve acquired knowledge while also acquiring new knowledge. This gives rise to the stability-plasticity dilemma, where stability refers to the ability to preserve current knowledge, and plasticity denotes its adaptation to new one (Mai et al., 2021).

In CL, the objective is to train our model initially with easy examples. Once the model has mastered them, it proceeds to train on progressively harder examples, aiming to learn the new ones while retaining the ability to solve the easier ones encountered at the beginning. This situation presents as a specific instantiation of the stability-plasticity dilemma.

The lack of stability is close related to Catastrophic forgetting: it is a phenomenon that affects Continual Learning systems, preventing them from performing earlier learned tasks after training in different ones. It is important to notice that not all Continual Learning tasks are prone to catastrophic forgetting (Thai et al., 2021) and catastrophic forgetting is not restricted to neural networks algorithms (Goodfellow et al., 2014). In fact, even biological learning systems suffer from it at different levels (Goodfellow et al., 2014; Kirkpatrick et al., 2016). A well-supported model of biological learning in humans suggest that neocortical neurons learn using an algorithm that is prone to catastrophic forgetting. To avoid this, there is a complementary mechanism consisting on a virtual experience system, that replays memories in order to continually reinforce tasks that were not performed in a while (McClelland et al., 1995). There is also evidence that the mammalian brain may avoid catastrophic forgetting by protecting previously acquired knowledge in neocortical circuits. When a new skill is acquired, a portion of the involved synapses are strengthened to prevent forgetting. Once those neurons are selectively erased, the corresponding skill is forgotten (Cichon and Gan, 2015).

Overcoming catastrophic forgetting is a key point in the development of artificial general intelligence (Kirkpatrick et al., 2016), and therefore it is a wide field of research (Arpit et al., 2017; Cichon and Gan, 2015; Goodfellow et al., 2014; Toneva et al., 2019).

To avoid this problem, several approaches were proposed. The most straightforward is to ensure that data from all tasks are available during the whole training process, which is often called multitask learning. If data from multiple tasks is always available, forgetting does no occur because the weights can be jointly optimized for all tasks (Kirkpatrick et al., 2016).

Another approach to avoid catastrophic forgetting, especially used in RL tasks, is the usage of replay memory (Mnih et al., 2013), which allows the algorithm to retain old data to be re-used while training.

There is also some research on the role of activation functions in catastrophic forgetting (Goodfellow et al., 2014; Srivastava et al., 2013).

Toneva et al., 2019 examine the dynamics of catastrophic forgetting at a single example level in the context of a classification task. They define unforgettable and forgettable examples based on whether an example has been correctly classified after the first time it is learned. They find that forgettable examples are usually composed of strange or mislabelled cases, and that they are located near the frontier. Additionally, they observe that unforgettable examples are not that useful for the task, and up to 30% of the dataset can be removed without affecting the performance if those examples are chosen from the more unforgettable ones. Following those observations, a CL strategy could be imposed based on the training dynamics. Up to my knowledge, such strategy has never been explored before.

2.2. What is Curriculum Learning?

The core idea of CL is to start training on the easy examples first, and then as training progress, also train with harder examples. Despite of the simplicity of this idea, there are two major points that need to be addressed before actually implementing it. First, there is a notion of easiness over the examples, that is not obvious to define. The second point is about adding harder examples to the training process: how and when should them be added is a major concern. In a recent survey by X. Wang et al., 2021, those two points are identified as the **Difficulty Measurer** and **Training Scheduler** respectively. Properly defining those two components is enough to frame most of the CL implementations, including the one we will use in this work. It should be noted that while there may be methods that fall outside the presented framework, the techniques and procedures utilized in this research align perfectly with the framework as presented here. Any additional methods that did not directly

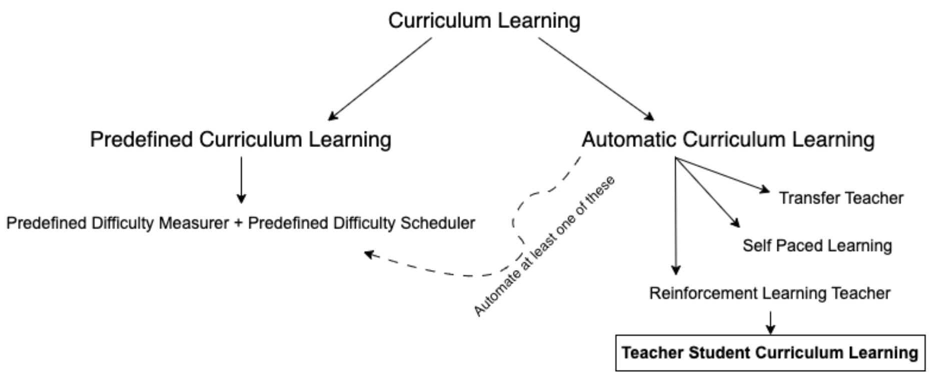

Curriculum Learning
Predefined Curriculum Learning
Automatic Curriculum Learning
Predefined Difficulty Measurer + Predefined Difficulty Scheduler
Automate at least one of these
Transfer Teacher
Self Paced Learning
Reinforcement Learning Teacher
Teacher Student Curriculum Learning

Figure 2.1: This figure shows the key concepts in CL, as suggested by X. Wang et al., 2021. To define a CL framework, we need to define a Difficulty Measurer and a Difficulty Scheduler. The easier way is to pre-define both of them following expert knowledge. In such case, we talk about **Predefined Curriculum Learning**. An improvement of such methods consists on automating one (or both) of those components with a data-driven approach. In this case, we talk about **Automatic Curriculum Learning**. In this section we cover three of them: Transfer Teacher, Self Paced Learning and Reinforcement Learning Teacher. TSCL is a particular algorithm of this family.

contribute to understanding the implementation and experimental procedures in this study were not included. The reference work by Soviany et al., 2022 provides a comprehensive taxonomy of CL.

As pointed out by X. Wang et al., 2021, CL methods can be classified into two broad categories: Predefined CL and Automatic CL, depending on how the Difficulty Measurer and the Training Scheduler are defined. However, this classification is not always clear-cut, as some methods may belong to both categories simultaneously.

In the following section, we will focus on Predefined CL methods, which were among the first to be proposed and are often regarded as a useful starting point for understanding CL frameworks. We do so for two main reasons: first, due to their historical significance, and second, because of their relative simplicity, which allows for a clear illustration of the practical implementation of CL. Later we will cover Automatic CL, presenting some alternatives and digging into the one we will later use: TSCL. A semantic map with an overview of those contents is shared in Figure 2.1.

2.2.1. Predefined Curriculum Learning

Predefined CL includes cases where the Difficulty Measurer and the Training Scheduler are designed by an expert based on prior knowledge, without data-driven algorithms.

2.2.1.1. Predefined Difficulty Measurer

In those cases, an expert defines how difficult each example is. This can be done as a discrete or continuous label, depending on how difficulty is expressed. In some works, examples are just divided into two cases: easy and hard. Some others include more cases but still as a discrete domain of cases, and there are some others where difficulty is mapped to a continuous score. The criteria here is based on the expert prior knowledge on the task.

The difficulty of an example can be explicitly labelled by humans, telling if examples' difficulty is low, medium or hard, according to their knowledge (Khan et al., 2011). There are two major constraints for this approach: first, an example that is easy for a human may not be so for an ML algorithm. Second, there is an extra cost associated with labelling each example according to this criteria.

There are other cases where examples' difficulty is defined based on some of its characteristics, yet based on human knowledge. For instance, in NLP tasks, a popular approach is to assign the difficulty of an example with its sequence length (Platanios et al., 2019; Spitkovsky et al., 2010): larger sequences are usually considered harder since to solve them, sub sequences of it must also be solved.

Similar to this, there are other cases where the difficulty is assigned depending on the dimension of examples (Karras et al., 2017; Tan and Le, 2021). This is typically done for computer vision algorithms, since small images are easier to generalize and faster to train on, while larger ones become easier to overfit, hence requiring more regularization, and making the algorithms unstable, particularly for generative models.

Sometimes the complexity of the examples is used as a measure of difficulty. For instance, recognizing regular shapes as an easier task than irregular ones, for image classification (Bengio et al., 2009), or considering the vocabulary complexity in NLP: the wider the vocabulary, the harder it becomes (Elman, 1993).

Finally, in classification tasks, certain authors classify examples not based on their inherent difficulty, but rather on the challenge of distinguishing between certain classes. This means that while an example may not be inherently difficult, it can pose difficulties in accurately classifying it within specific categories. In Mallol-Ragolta et al., 2020, they have to score each example from 0 to 10 to classify the pain expressed in faces. They start training only with extreme cases (0 and 10) and then increase the training set from extremes to the middle range examples. Something similar happens with Kakerbeck, 2018 where she addresses a classification problem. At the beginning of the training, a few classes are used, and, as training progress, new classes are included in the dataset. In this case, each class is not hard itself but it is hard to distinguish one class from the other.

Before moving forward to the next section, it worth mentioning that Predefined Difficulty Measurer is the name given by X. Wang et al., 2021; however I think that instead, it should be named as Expert Based Difficulty Measurer. This alternative proposal puts the focus in the fact that expert knowledge is used to measure the difficulty of each instance. Its current name highlights the fact that difficulty is pre-computed and do not get changed or updated while training, which is true, but, there are some measurers named Transfer Teacher, which we will cover later, in which difficulty is also assigned before training begins and it also do not change during it. However, they do not lie in current category because the difficulty is not based on expert knowledge but in a surrogate model, which allows to automate its computation.

2.2.1.2. Predefined Training Scheduler

While measuring the difficulty of an example strongly depends on the task and the prior knowledge available, defining a Trainer Scheduler is often model agnostic, and most of the literature proposes similar strategies (X. Wang et al., 2021). Schedulers can be divided into two groups: discrete and continual schedulers. Discrete schedulers adjust the training set after some epochs of training, while continual schedulers propose a function to dynamically adjust the portion of examples for each difficulty during the whole training. Note that, this function is known before hand, chosen and implemented by a human, based on its knowledge. This function is often called *pacing function* and is formulated as a function $\lambda(t)$ that maps the number of epochs t to a scalar

$\lambda \in (0, 1]$ defined as the portion of easiest training examples to be included at epoch t. An straightforward definition of such function consists on growing the portion of hard examples linearly with time but there are many other alternatives.

In addition to this, it is worth mentioning that while most of the schedulers lead to a final training set containing all of the available examples (Elman, 1993; Kakerbeck, 2018; Platanios et al., 2019; Spitkovsky et al., 2010), there are other cases where easier examples are only used to train in early stages, and then discarded to focus in the harder examples only (Bengio et al., 2009; Karras et al., 2017). This observation is pointed out by Graves et al., 2017: the first scenario is called *multi task target* while the second is named as *task target*. Note that those different scenarios apply to both, predefined and automatic schedulers. As we will see later, TSCL corresponds to a multi task target formulation.

2.2.2. Automatic Curriculum Learning

Previous approaches to instantiate a task as a CL problem are all based on human prior knowledge of the task or how the training is expected to progress in order to properly define them. This may be hard to achieve or lead to non-optimal solutions given that it could be difficult to map the knowledge properly to the implementation or there may be mismatches between what is expected to happen and what actually happens. On the other hand, Automatic CL tries to overcome those problems by using data driven approaches to define both, the difficulty measurer and the trainer scheduler, aiming to automate those decisions.

The upcoming sections intend to provide a general overview of Automatic CL methods. Specifically, we will focus on three categories: Transfer Teacher, Self Paced Learning (SPL) and RL Teacher, as classified in the recent survey by X. Wang et al., 2021. While other Automatic CL methods exist, we limit our discussion to these categories to keep the scope of this work manageable. In addition, we will dedicate a separate section to an in-depth analysis of the TSCL method, which falls under the RL Teacher category.

2.2.2.1. Transfer Teacher

The key point here is to use a surrogate model to act as a Teacher who measures the difficulty of the examples based on its performance on them. The Teacher can be trained in the same training data that will be used for the Student, or with an external source, and can be exactly the same model that will be used for the Student (Hacohen and Weinshall, 2019; B. Xu et al., 2020) or it can be a different one (Weinshall et al., 2018). The idea is that a mature model that can solve the task good enough, can be used to tell which examples are harder than others. The Teacher transfers its knowledge about the difficulty of each example to allow the Student to learn from easier to harder (Hacohen and Weinshall, 2019; Weinshall et al., 2018).

Usually the loss of each example in the Teacher model is used straightforward as the difficulty measure to be transferred. This approach was applied in image classification (Hacohen and Weinshall, 2019; Weinshall et al., 2018) and in Neural Machine Translation (NMT) (B. Xu et al., 2020) for instance.

In other cases, it is not the loss what the Teacher uses to measure the difficulty of examples, but some metrics related to uncertainty over examples, for a Teacher model: the lower the uncertainty, the easier the sentence (Zhang et al., 2018; Zhou et al., 2020). In W. Wang et al., 2019 something similar is done, but Teacher model is used to consider the level of noise involved in the labels.

Note that hard examples and noisy labels are closely related: considering an error metric to measure the difficulty of an instance, such as its loss, makes it hard to differentiate a mislabelled example from a genuine difficult one.

Those methods measure the difficult of an example with respect to a different model instance than the one being trained: the Teacher, which implies that complexity of examples is measured against a goal hypothesis (Hacohen and Weinshall, 2019). This contrasts with other techniques that we will later explore, like SPL or Prioritized Experience Replay (PER), where the complexity is measured against the model that is being trained: current hypothesis. Another difference is that once the curriculum is defined, there is no feedback from the learner and it remains static throughout the entire training process, in contrast to SPL (Kumar et al., 2010), PER (Schaul et al., 2015) or non-uniform Batch Selection (Loshchilov and Hutter, 2015) among others, where there is feedback from the model to adjust the examples being chosen according to the model's performance.

In addition to this, those methods define the difficulty for each example, and then difficulty remain fixed while training, so the difficulty is actually predefined with respect to the training process. Predefined Difficulty Measurer is the name proposed in X. Wang et al., 2021, however, Expert Based Difficulty Measurer looks like a more appropiated name for it, since it puts the focus in the fact that there is expert knowledge involved to define it, and hence, Transfer Teacher methods, as presented in this section, are indisputably out of the Expert Based Difficulty Measurer category.

In the following section, we will dive into SPL which is a popular alternative to automate the CL implementation. PER and non-uniform Batch Selection are covered later in a specific section discussing the Harder First Methods (HFM).

2.2.2.2. Self Paced Learning

The idea behind SPL is to automatically define the curriculum while training, based on the model's own knowledge (Kumar et al., 2010). This is analogous to a student who decides by itself what to study, the order and time spent on each topic. SPL is one of the most spread CL framing (Avramova, 2015; Gong et al., 2018; Jiang, Meng, Mitamura, et al., 2014; Jiang, Meng, Yu, et al., 2014; Kumar et al., 2010; Li and Gong, 2017; Sangineto et al., 2018; C. Xu et al., 2015; Zhao et al., 2015).

The difficulty of an instance is defined as its current loss value, which avoids the usage of subjective difficulty measures. The greater the loss, the more difficult the example. In order to define which examples are easy enough to be included on training, a threshold is defined: those samples whose loss is below the threshold, are considered, otherwise ignored. The threshold is linearly increased with training, starting with very few samples considered for training, and adding new ones as the training proceed until all samples are considered. This function defining which examples are or are not considered for training is called the pace function.

Notice that not only the threshold grows as training proceeds, but also the loss function is expected to decrease, which also increases the number of new samples included on each stage. However, if for any reason a single example increases its loss, it may be removed from the curriculum, since it is updated iteratively before each training epoch. This is something expected to happen and is specially useful for mislabeled instances.

There are many other alternatives to define the pace functions. Linearly increasing it is the simpler way (Kumar et al., 2010), but there are others that mainly try to get rid of the hard weights that this basic formulation imposes: at each step, instead of a binary weight for each instance meaning considered or not, those alternatives propose continuous or soft weights, where harder examples are weighted lower than easier, and weighting functions are continuous. Some of those are Linear Soft Weighting (Jiang, Meng, Mitamura, et al., 2014), Logarithmic Soft Weighting (Jiang, Meng, Mitamura, et al., 2014) or Mixture weighting (Zhao et al., 2015) among others (Gong et al., 2018; C. Xu et al., 2015). There is also an alternative implementation of SPL which is Self Paced Learning with Diversity (SPLD) (Jiang, Meng, Yu, et al., 2014; Zhao et al., 2015) that aims to not only consider the difficulty of an example to train or not with it, but also cares about the diversity of the examples picked to train. It is also possible to formulate SPL so that it accounts for prior knowledge when weighting the instance (Jiang et al., 2015). This weight-based interpretation can be implemented straightforward as part of the error function being optimized (X. Wang et al., 2021).

One of the most outstanding properties of SPL and its variations, is the capacity to learn from noisy data, even with heavily noisy data (Jiang et al., 2017; Meng et al., 2015). Highly noisy samples, with larger loss values, will be automatically screened out from the training set since its loss value will be large. This way, SPL prevents noisy samples from negatively influencing the learning quality (Meng et al., 2015).

As a disadvantage, since SPL methods rely in its own knowledge, it may not be optimal at the beginning of training when the model doesn't know anything about the problem. Following the analogy of a person who studies by itself, it is hard to know where to start without any external advice (X. Wang et al., 2021).

In the next section, RL Teacher methods are presented. These methods tries to overcome this problem by using RL to learn which order should be followed by the Student using its performance as feedback, trying to get the best of the two alternatives presented up to the moment.

2.2.2.3. Reinforcement Learning Teacher

In several formulations of CL, the curriculum is automatically designed using RL techniques (El-Bouri et al., 2020; Graves et al., 2017; Matiisen et al., 2019). These methods allow the curriculum to be defined based on the training dynamics, taking into account the difficulty of an example with respect to the current state of the model being trained, in contrast to predefined scheduler or pace functions.

One notable difference among the RL-based methods discussed in this section lies in how the problem is formulated as an RL problem. Most of them adopt a multi-armed bandit approach, while others frame it as a traditional RL problem. The key distinction lies in the presence of a state: in the former, there is no explicit state and actions are evaluated on their own. The goal is to learn a function that recommends the best action to maximize the expected total reward over some time period. Actions are independent among them and throughout the time, and do not change anything else. In the later case, there is a state that changes every time an action is taken. The objective is to learn a function that identifies the action that maximizes cumulative reward over time at each state. This function is commonly referred to as the value function, which takes a single argument in the former case ($V(a)$), but must consider the state in addition to the action in the later ($V(s, a)$). In summary, multi-armed bandits are memoryless and treat each decision in isolation, while full RL takes into account how actions influence future states and rewards.

As a consequence, when RL is addressed as a multi-armed bandit problem, a single training session is performed until the model reaches the desired performance in the main task. On the other hand, when pure RL is used, a single training session is considered as an episode, and multiple episodes are needed to train the RL model before actually training the final version of the model used for the main task.

With respect to the actions, each action corresponds to choose on which sub task to train. The sub task definition for supervised methods consists in just splitting the dataset into subsets that follow certain pattern so that can be considered as a sub task. It may be just a cluster strategy based in the inputs (El-Bouri et al., 2020) or for a particular feature of the dataset related to the complexity of the problem, such as the sequence length (Matiisen et al., 2019).

The reward definition must be related to the learning progress, and is inspired in the self-motivation paradigm (Graves et al., 2017). There are two main strategies for its definition: to measure the progress in terms of loss changes or in terms of the complexity of the model, that is, changes in the learner's weights. For both scenarios, changes are measured before and after training the model for some time called step.

In the following section we will dive into a particular case, that is the one we will later implement: TSCL.

2.3. Teacher Student Curriculum Learning

This method is proposed by Matiisen, Oliver, Cohen, and Schulman in Matiisen et al., 2019. The idea is that the main task is solved by a Student model, while there is also a Teacher model that learns how to deploy the best curriculum to the Student, and both are trained simultaneously in a single round of training. The Teacher is trained following two premises: the learner should train more often on those tasks which are improving the most, but also on tasks on which the performance is getting worse, to prevent catastrophic forgetting.

The formalization of TSCL as a Partially Observable Markov Decision Process (POMDP) is as follows:

- state: the complete internal state of the Student, basically the neural network, which is not observable to the Teacher.
- action: which task chooses the Teacher to be learned by the Student.
- observation: the score obtained by the learner on the task.
- reward: the change in the score for the task with respect to the last time the task was chosen.

TSCL solves the curriculum design problem, since the Teacher learns how to do it in the interaction with the Student. However, the POMDP formulation of TSCL is given in terms of tasks, but tasks are manually designed (Matiisen et al., 2019).

Despite of being formalized as a RL problem, the authors say that, to address the only-train-once constraint, they implemented it as an heuristic, based on non-stationary multi-armed bandit problem.

This implies two major things. First, there is no state. At each action, the multi-armed bandit framing assumes that actions' consequences does not depend on what happened before, and actions are valuable for them selves. However, this is a simplification. Second, the non-stationary reflects the fact that reward function changes over time. It happens because, on early training steps, the score usually changes a lot, and as the model converges, scores has small changes until the end.

They propose four alternatives for the implementation of those heuristics, which are the following:

- **Online algorithm**: is inspired by the basic non-stationary algorithm (Sutton and Barto, 2018). The expected return Q is exponentially weighted by a moving average:

$$Q_0(a) = 0 \quad \forall a \in \mathit{Actions},$$
$$Q_{t+1}(a) = \alpha r_t + (1-\alpha) Q_t(a),$$

  where α is learning rate. This weighting scheme is appropriated for the non-stationary problem being learned. The next task to train is randomly chosen using a softmax distribution. This solves the exploration problem. Alternatively, they also try epsilon-greedy strategy.
- **Naive algorithm**: The estimation of learning progress is more reliably estimated after several practices. This algorithm trains each sub task K times in sequence, observes the resulting scores and estimates the slope of the learning curve using linear regression. This regression coefficient is used as the reward in the above **Online algorithm**.
- **Window algorithm**: The goal is to avoid repeating the same task for a fixed number of times, since it is expensive and may have consequences on the learning process. Here, each task is executed only once, and the score is kept with a timestamp. To compute the reward, linear regression is performed, considering both, the scores and timestamps. Then, the coefficient is used as a reward in the online algorithm above.
- **Sampling algorithm**: The previous approaches require tuning of hyper parameters to balance exploration, and also need to compute a linear regression for the reward estimation. This algorithm get rid of those steps, taking inspiration from Thompson sampling. The Sampling algorithm

keeps a buffer of last K rewards for each task. Then, to choose a new sub task, a recent score is sampled from each sub task's K last rewards buffer. Then, the highest sampled score determines the task to be chosen. This makes exploration a natural part of the algorithm.

The framework is tested for a supervised problem and a RL problem: addition of decimal numbers in Long short-term Memory (LSTM) and navigation in Minecraft respectively. For both tasks, TSCL improved considerably the results with respect to not using CL, and showed faster convergence than hand made curriculum strategies. However, no single strategy is universally superior among the four proposed, and even the usage of epsilon-greedy or Softmax distribution for exploration is problem dependant. They also observe that the usage of absolute value of Q boosts the performance, which shows that effectively, this is efficient against catastrophic forgetting: training more often also in those sub tasks being forgotten to prevent it.

When this work started, exploring CL techniques with a special emphasis on RL-based methods was important in its own right. Multi-armed bandit methods showed an important advantage among such methods: in a single training session, both Teacher and Student are trained. Despite pure RL methods reporting better results (El-Bouri et al., 2020), the differences are not as large as the differences in resources involved in training, especially for a heavy task like the addressed in this work. In those other methods, each Student's training session is used as a single episode for training the Teacher, and several episodes are needed to correctly train the Teacher and finally, train the Student that is actually used for the task.

TSCL has two additional features that made it an interesting framework to try: first, it exposes an explicit mechanism to deal with catastrophic forgetting, and second, TSCL provides an open source implementation with the code used to run the experiments from the paper, which was actually useful for fully understanding the framework while re-implementing it. Based on these reasons, we chose to implement and try the TSCL framework out of the available options in the literature. The implementation details are covered later in Chapter 3.

To finish our overview in CL methods, the next section presents some other ideas that may or may not be considered as CL, but will make us revisit some concepts exposed up to the moment to question them.

2.4. Harder First Methods

HFM are also sometimes called anti-curriculum strategies, and they consist on doing exactly the opposite to what CL proposes: training first on harder examples. There are at least two ways in which researchers arrive to such methods: the first is straightforward as an unexpected result. When comparing results, authors often propose to compare the CL method against the one that consisted on doing exactly the opposite of what the curriculum proposed, and it turns out to be the best scores they achieve in some scenarios (Avramova, 2015; Zhang et al., 2018). In some other cases, the methods are proposed following the reasoning that, in order to decrease the error among several examples, it may worth to start by those examples where the error is the larger. It makes sense since those are the one that contribute the most to the average error, so it is sound to start with optimizing them. Those methods include PER (Schaul et al., 2015), non-uniform Batch Selection (Loshchilov and Hutter, 2015) or Hard Example Mining (Shrivastava et al., 2016).

This gap could be explained by how noisy examples are managed in each of them. For instance, consider a mislabelled example. With a loss-based difficulty measurer at instance level CL strategy, like SPL, it will be selected as a difficult example, hence affecting the frontier only at the end of the training, when most of the regions are already defined or could even be excluded from the training set. However, using this same instance with a HFM, it will weight more in the training, and will be used at the very beginning when the regions are being defined. Hence the method may not work. Following this observation, some authors suggest that CL strategies are more suitable for noisy datasets when there is some uncertainty on the labels, while HFM are more suitable for clean dataset when labels are trustworthy (Chang et al., 2017). However, this explanation may not cover all cases, and more research is needed to understand this difference (X. Wang et al., 2021).

There are some strategies that explicitly try to balance those two conditions while training: sample too easy cases as well as too hard in order to balance those criteria, trying to cover both cases, noisy and clean datasets, since in the practice the labelling quality may be unknown beforehand (Chang et al., 2017), and in fact, RL methods as previously discussed could be a particular case of such methods, since examples are not chosen for training explicitly based on how easy or hard they are, but according to what are the best examples to feed the model in the next step (X. Wang et al., 2021). However, in TSCL this is done in terms of sub tasks only because the Teacher usually chooses which sub task to sample, in opposition to automated methods like SPL which are able to pick single examples.

Now that we have a wider understanding on what is CL and how those ideas were put into practice, the following section delves into a critical inquiry: what is the underlying reason for its efficacy?

2.5. Why does Curriculum Learning work?

Now that we presented several approaches to CL implementation, with its pros and cons, the reader may be wondering why does CL works. This is one interesting question, and its answer depends a lot on which framing of CL does it refer to, and in which context: noise in the dataset, for instance, may affect the behavior of the CL strategy adopted as earlier discussed. In this section we will first provide a couple of broad thoughts trying to share some general insights related to it, but without a complete answer. This section does not include the specific case of TSCL, that is the technique we will later implement. Instead, this will be addressed in section 5.1 as part of this work's conclusions, once the method is implemented and results are shared. To the best of my knowledge, there is no previous literature addressing this question for TSCL.

First of all, it is important to notice that some problems need the complete dataset at once to be solved. For instance, the XOR function can be learned with a three layer neural network with sigmoid activation function, but if one of the fourth pattern is withheld until late in the training, the network typically fails to learn XOR. Instead, it will learn a function compatible with the parameters shown on training, for example the OR function (Elman, 1993).

That said, a new question raises: what kind of problems can or cannot be addressed with CL? Of course XOR is a toy problem, but it is still a valid one. In fact, as previously discussed, there are tasks where HFM performs better, while doing exactly the opposite to what CL proposes.

Going back to the original question in this section, Elman, 1993 proposes some points:

First, since the activation function he uses is sigmoid function, and is usually initialized near zero, those neurons are very sensitive at the beginning of the training, but as training advances, weights move to less active zones, making neurons less sensitive. In fact, there are entire research studies which propose different algorithms to initialize the weights to ensure that, as the training begins, neurons are as sensitive as possible, for example, the widely used Xavier initializer (Glorot and Bengio, 2010). This makes clear that, as humans, on the early live of a neuron it learns faster (Elman, 1993).

Second, and related with the previous point, is the gradient descent algorithm. It changes the hypothesis on each step, but modifies it only a little bit. It is also known that converging to local minima prevents the network from arriving to global minima, and in this early convergence, the initialization and first steps plays a crucial role. Once a network is committed to an erroneous generalization, it may be unable to escape this local minimum. Considering the first point, this get worst as training proceeds.

It is interesting to notice that this analysis is based on the usage of the sigmoid activation function, but it is not necessarily true for all activation functions: it is only true for bounded activation functions, such as sigmoid or tanh, partially true for relu, which is only bounded below, and not true for functions like leaky relu or maxout. The point described is related to the vanish gradients or dead neurons and has been a field of research for years. As a result, functions such as sigmoid and tanh are not that popular nowadays, with relu becoming a sort of *de facto* standard (Hendrycks and Gimpel, 2016). However, the gradient descent algorithm also plays a role here: even when non-saturable activation functions are used, it is common to use dynamic strategies with respect to learning rate, decreasing its size as the minimum is expected to be nearer, which also ends up in the same phenomena: learning starts faster initially and slows down as it converges.

For a linear regression model, the convergence is monotonically with the difficulty of the examples, measured with respect to a target hypothesis (Weinshall et al., 2018). Among examples with similar difficulty, convergence is faster when training in points with higher loss with respect to the current hypothesis. This is also verified in practice for deep learning models (Hacohen and Weinshall, 2019; Weinshall et al., 2018).

Those explanations are closely related to the fact that some CL methods can be interpreted as a regularization method (Bengio et al., 2009; Kumar et al., 2010) that helps to heavily smooth the error function, specially in the early steps, allowing the training to converge easier since this regularization guides it towards better regions in parameter space (X. Wang et al., 2021). Another alternative is to view the CL framework from a data distribution perspective: CL changes the frequency that each example is used for training, which for instance allows under-sample noisy examples while giving more importance to the correctly labeled ones (X. Wang et al., 2021).

With respect to TSCL, up to the moment, there is not a theoretical analysis and characterization of the method in the literature. We will share our insights once we conclude this work.

2.6. Summary

In this chapter, we first introduced the concepts of Continual Learning and Catastrophic forgetting. We then explored various implementations of CL, with a focus on RL-based methods. We provided a detailed overview of TSCL and explained why we selected it as the algorithm to implement and try. We also discussed methods that may seem counterintuitive in the context of CL: HFM, and provided insights into why these ideas work.

In the next chapter, we will present a detailed description of the dataset and the problem, and moreover, describe the TSCL implementation proposed in this work.

Chapter 3

TSCL applied to OCR

The main focus of this work is to understand the effects of CL, implemented as TSCL, on the training process and resulting model. To establish the impact of CL, we use a strong base: an OCR model trained under traditional conditions. We then use the same model, hyperparameters, and dataset to train from scratch using TSCL. By using the same model, hyperparameters, and dataset for both traditional training and TSCL, we can isolate the effects of CL and establish its impact on the training process and resulting model.

In this Chapter, we first present the particular OCR problem and the dataset we will use, then describe the base model proposed by Chavat Pérez, 2022, which serves as the basis for our implementation. We propose some improvements to his work and also define our problem as a TSCL-based one. Finally, we run several experiments to validate our approach and compare it to traditional training methods.

3.1. Problem description

The selected problem for this work is an OCR task that is part of the LUISA project ("Leyendo Unidos para Interpretar loS Archivos", 2019), which is part of a larger project, called CRUZAR ("Cruzar, sistematización de archivos militares", 2019). Both of them were launched in 2019 and are still active.

CRUZAR project aims to recover and analyze documents produced during the last civic-military dictatorship in Uruguay. LUISA is a side project that intends to transcribe those documents after they were digitized. In LUISA, volunteers are given certain small images containing text, and they must type what they see there, using a context image to help themselves. Chavat Pérez, 2022 took thousands of those examples and processed them to create some datasets, in particular the one that we will use in this work.

An OCR task involves converting an image containing multiple lines (and columns) of text into its plain text version, making it suitable for machine processes. This process typically comprises several steps, which may vary across different implementations, but commonly include the following:

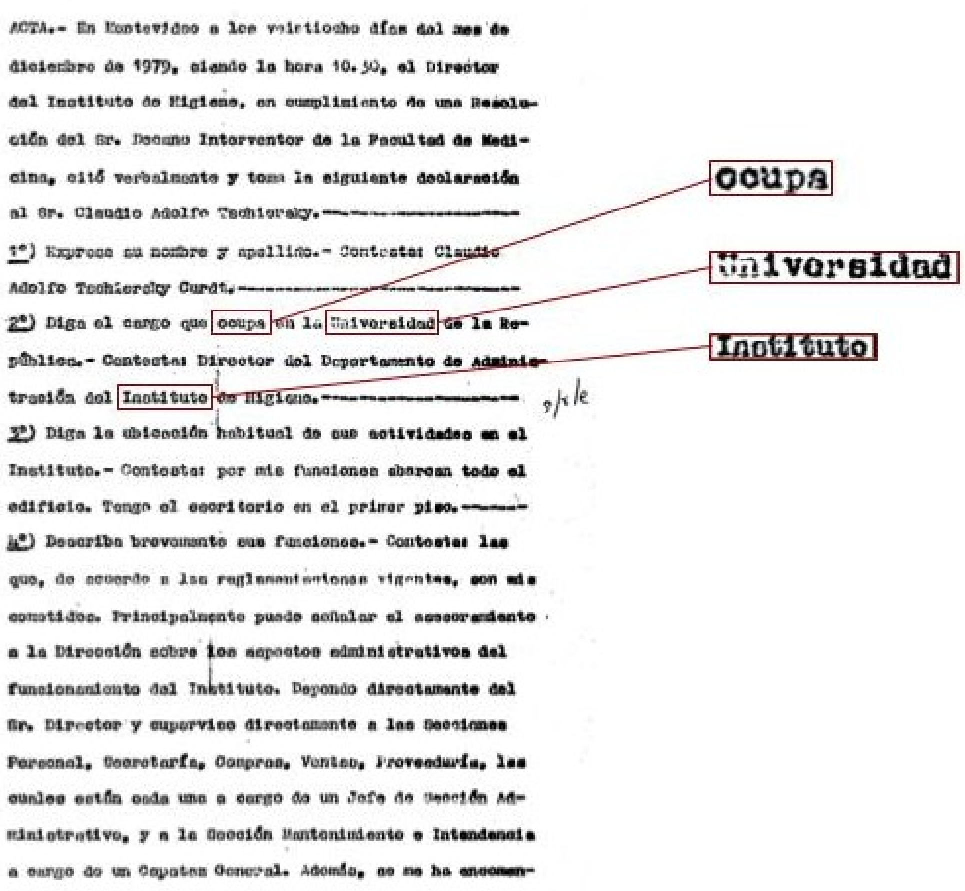
1. Preprocess the image by converting it to a binary format (purely black and white) while enhancing its quality to facilitate subsequent steps.
2. Correct the orientation of the text, as images may be rotated concerning the lines of text.
3. Identify sections of the image containing lines of text and extract them.
4. Convert the characters found on each line of text into machine-encoded text; this process is referred to as the **transcription** of a single line of text in the image to plain text.
5. Transcribed lines must be re-assembled to a single string containing the text from all the document.

In this work, we focus on the step of the **transcription** from a line of text that is already identified in the image into a sequence of characters present on it. This is framed as a sequence to sequence task, where input is interpreted as a sequence of pixels while output is a sequence of characters transcribing the content of the given line in the image. The following section describes the dataset used for this transcription task in depth.

3.2. The Dataset

The full dataset consists on documents produced between 1958-1985 during last civic-military dictatorship ruled in Uruguay, that were found in the Ministry of Defence in 2006-2007. Documents were digitalized into binary images and transcribed with Tesseract [1] (Kay, 2007), an open source OCR tool. Those documents that, according to some heuristics, could not be correctly transcribed with this tool, were separated to be labelled by volunteers (Etcheverry et al., 2021). This crowd-sourcing labeling process carried on by

[1]https://github.com/tesseract-ocr/tesseract


ocupa
Universidad
Instituto

ACTA.- En Montevideo a los veintiocho días del mes de
diciembre de 1979, siendo la hora 10.30, el Director
del Instituto de Higiene, en cumplimiento de una Resolu-
ción del Sr. Decano Interventor de la Facultad de Medi-
cina, citó verbalmente y tomo la siguiente declaración
al Sr. Claudio Adolfo Tschiersky.--------------------
1°) Exprese su nombre y apellido.- Contesta: Claudio
Adolfo Tschiersky Curdt.------------------------------
2°) Diga el cargo que ocupa en la Universidad de la Re-
pública.- Contesta: Director del Departamento de Adminis-
tración del Instituto de Higiene.--------------------- 9/1k
3°) Diga la ubicación habitual de sus actividades en el
Instituto.- Contesta: por mis funciones abarcan todo el
edificio. Tengo el escritorio en el primer piso.-------
4°) Describa brevemente sus funciones.- Contesta: las
que, de acuerdo a las reglamentaciones vigentes, son mis
cometidos. Principalmente puedo señalar el asesoramiento
a la Dirección sobre los aspectos administrativos del
funcionamiento del Instituto. Dependo directamente del
Sr. Director y superviso directamente a las Secciones
Personal, Secretaría, Compras, Ventas, Proveeduría, las
cuales están cada una a cargo de un Jefe de Sección Ad-
ministrativo, y a la Sección Mantenimiento e Intendencia
a cargo de un Capataz General. Además, se me ha encomen-

Figure 3.1: This image shows a digitalized document and some blocks extracted from it. Each block is designed to contain a single word on it. Those blocks that, following some heuristics, can not be properly transcribed with open source OCR tools, are given to humans annotators. Once labelled by at least three human annotators, some heuristics are applied to properly combine them into lines of text with a single transcription for each. This allows to take advantage of relations between the words in a line. The image belongs to Chavat Pérez, 2022.

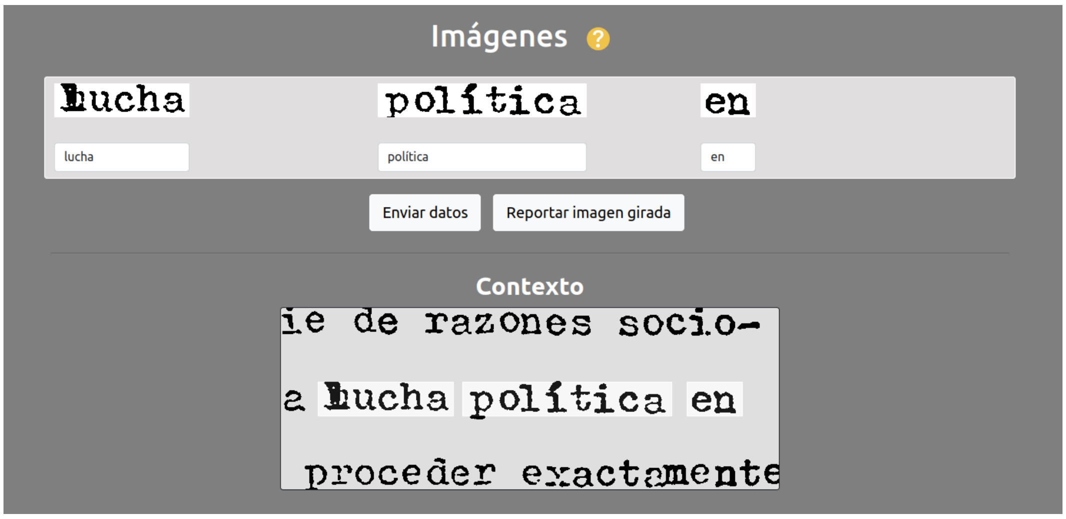


Figure 3.2: This is the interface used during the crowdsourcing labelling tool LUISA. The user receives the blocks with the words to be transcribed, and also a small context image which corresponds to a region larger than just the blocks. Using the blocks and context image, the user is expected to label each block with the text shown in the blocks. This image is taken from Chavat Pérez, 2022.

thousands of volunteers is the LUISA project ("Leyendo Unidos para Interpretar loS Archivos", 2019).

Before giving images to annotators, there was a pre-processing stage to improve image quality, identify the regions containing text, and split them in small boxes with a single word on it, as shown in Figure 3.1.

Once pre-processed, the examples are given to the volunteers, with a few blocks at a time to be transcribed, with some context surrounding the blocks that the annotator can use to provide the text associated with each block. This is shown in Figure 3.2. More details on the labeling process can be consulted in Etcheverry et al., 2021

Crowd sourced labelled images constitute a unique dataset with its examples share some properties: had several damage caused by the time and bad conservation conditions, and some distortion due to the digitalization process, which explains why available OCR tools do not perform well on them.

Each block example was labelled at least three times by three different annotators, then some heuristics were followed by Chavat Pérez, 2022 to compute a single most-common tag out of them. In addition to this, they also applied some rules to convert blocks of text into lines of text, which is non trivial since some documents are written in columns, so, when all blocks from the same line are concatenated, they could cross different columns, and hence they must be

Split	Num. examples	Num. of tokens	Image width	Image height	Image width / Num. of tokens	New symbols
Train	66900	23.0 ± 24.1	675.7 ± 710.6	49.5 ± 18.7	29.0 ± 22.5	∅
Validation	9600	23.3 ± 24.2	683.9 ± 716.6	49.3 ± 18.4	28.7 ± 15.2	▪️ ⬛ ♦️ ⚪
Test	19114	23.0 ± 24.4	675.7 ± 714.8	49.1 ± 18.1	28.7 ± 16.9	← ŕ ↓ ª Ç ⚫

Table 3.1: This table summarizes each split from the *dataset líneas* defined by Chavat Pérez, 2022. Each rows reports the average and standard deviation for each feature in the dataset. Number of tokens refers to the number of tokens that need to be predicted, which includes the <EOS>. Height and width are in pixels, so the column containing image width over number of tokens can be interpreted as the average number of pixels that a character consumes. The column New symbols refers to symbols that are in the split but are not in the training set. For the training set it is empty. There are four new symbols in the validation set and six in the test set. On average, all features are similar within the dataset, but the standard deviation is high, which shows that there are lots of variability in the dataset.

splitted to keep lines at column level. The present work uses the same dataset built by them, named as *dataset líneas*, which is summarized in Table 3.1.

Now that the problem and dataset have been described, it is time to discuss the model used. Firstly, we will review the approach outlined by Chavat Pérez, 2022, and subsequently suggest some enhancements to establish a foundation for our TSCL implementation.

3.3. Base Model

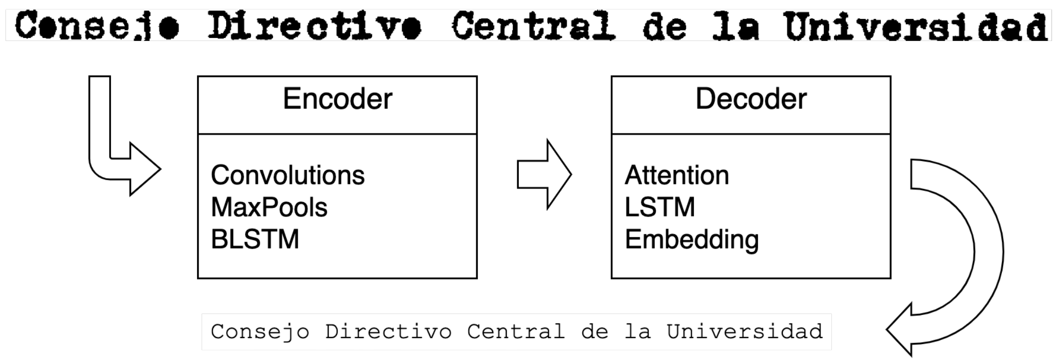
The base model comes as a result of a Final Project for the Bachelor's Degree in Computer Engineering, in Sequence to Sequence models to perform OCR tasks by Chavat Pérez, 2022. This work experimented with several configurations and hyperparameters, trying with different approaches to address the same problem, and concluded in a model that has a good performance on the task. That model serves as a reliable starting point for our work.

In addition to it, this model will play the role of Student in our TSCL implementation. We will delve into its implementation in the following subsections.

3.3.1. Architecture

The model proposed by Chavat Pérez, 2022 follows a sequence to sequence approach where input is an image, interpreted as a sequence of pixels, while output is text which is a sequence of characters.

The input are binary single-channel images, which are resized to a fixed



Consejo Directivo Central de la Universidad
Encoder
Convolutions
MaxPools
BLSTM
Decoder
Attention
LSTM
Embedding
Consejo Directivo Central de la Universidad

Figure 3.3: This is a high level view of the base model used to solve the OCR task. The model receives a binary image interpreted as a sequence of pixels, and outputs plain text, as a sequence of characters. Hence, a seq2seq method. More details and low level description are given by Chavat Pérez, 2022.

height of 64 pixels while maintaining the aspect ratio, and a minimal width of 24 pixels. Images that, once resized, are still shorter than this length, are padded with ones, that is the white background value. Since the model needs to stack several images to conform a batch of examples, all images within a batch must have exactly the same dimensions, but, all images are in different sizes, so to fill this gap, all the images within a batch are padded on its length to let them as large as the largest in the batch.

Once batched, the images are fed to an Encoder model. It has two layers of convolutions, followed by a max pooling, then one more convolution and finally another max pooling. As a result, the image is encoded into a feature vector that is fed to a bidirectional LSTM.

Once the image has been encoded, the model moves on to decoding the text output: an LSTM with an attention mechanism is used. The attention mechanism allows the model to weigh all the information obtained from the input, given the output that was produced up to the moment. On each step, a single character is predicted.

Figure 3.3 shows an schematic simplification of the OCR model.

To improve the learning process, there is a mechanism called **Teaching Forcing**, which is used while training the decoder: at each time step, with certain probability, the predicted char is replaced with the correct one to predict the next character. Finally, the model is fit using SGD algorithm with ADAM variation. For both, Teacher Forcing and ADAM, parameters are used exactly the same than Chavat.

Finally, Chavat, trained each model for a fixed number of epochs, storing the weights at each. Then, he picked up the best model in terms of validation set error, and defined this as the trained model. This is a valid comparison as far as all runs are compared with the same approach, but may lead to overfitting on the validation set. Only once all validations over different configurations are done, the final model is chosen and measured against the test set. This exact methodology is followed in the current work.

3.3.2. Metrics

The metrics employed in this study are the same as those used in Chavat Pérez, 2022, and were implemented using the exact code from that study.

- **Negative Log Likelihood (Loss)**: This is the widely known loss function for classification problems. This function penalizes the model inversely proportional to the probability assigned to the correct class. This error is summed over all the predicted sequence. As a result, larger sequences lead to larger losses. This is the metric being optimized by the SGD algorithm.
- **Character Error Rate (CER)**: This metric measures the portion of chars that are wrong in the predicted sequence. It is implemented as the Levenshtein distance, where each operation (insert, delete, replace) has a value of one between predicted and ground truth sequences, divided by the ground truth length. It is formally defined as follows:

  $$CER = \frac{I + D + R}{N}$$

  where I, D and R corresponds to the number of inserts, deletes and replaces operations needed to convert the predicted sequence into the ground truth sequence, and N is the number of characters in the largest alignment between the sequences.
- **Longest Common Subsequence ratio (LCS)**: The LCS is a metric that measures the ratio of the largest sequence of characters in common between two strings, regardless of their exact content. Formally it is defined as follows:

  $$LCS(P, T) = \frac{2 * len(C)}{len(P) + len(T)}$$

split	Epochs	CER	SoftCER	LCS
Validation	11	28.24	-	74.0
Test	-	28.22	27.74	74.00

Table 3.2: This is table shows the best results reported by Chavat Pérez, 2022 for validation and test splits for the same dataset used during this work. This table is composed of two different tables in his work, and non reported values are not shared here.

where P and T are the predicted and target strings, and C is the largest common sequence to strings P and T.

- **Soft CER** and **Soft LCS**: Those metrics come from an observation during Chavat's work: several examples were annotated as case insensitive. Some volunteers just used lower case or upper case for some labels, no matter how is the text actually written. This metric addresses this issue by computing the CER or LCS after both, predicted and ground truth strings, are converted to upper case. Hence, they are case insensitive metrics.

The way that metrics are computed during training and evaluation differs slightly. During training, the model makes predictions for each example until an End of Sequence (<EOS>) character is predicted, or until the same number of predictions as in the ground truth sequence are made. During evaluation, predictions are also made until an <EOS> character is predicted, but up to 105 predictions are allowed. This stop criteria is based on the average plus two times the standard deviation of sequence length in the dataset. The reason for allowing up to 105 predictions during evaluation is to avoid leakage of information from ground truth to validation. This stop criteria is also used during our evaluation to ensure comparability. However, it is worth noting that the larger predictions are allowed, the larger sequences are generated and as a result, the loss value is also larger, because there are more characters which can accumulate errors.

Table 3.2 presents a summary of the best results reported by Chavat Pérez, 2022 for the validation and test splits of the same dataset used in this work. The table consolidates two different tables from Chavat's work and reports the results for the metrics of CER, Soft CER, and LCS. The reported number of epochs corresponds to the epoch at which the best validation result was found, out of 20 epochs. Non-reported values in Chavat's work are excluded from this table.

3.3.3. Strengthening the Base Model

The implementation made for our work is based on the same code used by Chavat. During our implementation, some simple changes were made to the base model, in order to make it faster, improve the results and fix some bugs. This section summarizes those details. Evidence of the actual benefits for those changes is presented and discussed in Chapter 4.

The model is implemented in PyTorch (Paszke et al., 2019) and run in a NVIDIA Tesla P100 12 GB RAM in ClusterUY (Nesmachnow and Iturriaga, 2019).

3.3.3.1. Mixed Precision

By default, PyTorch performs its operations in tensors of float32 datatype: the use of float32 tensors allows fast computing while preserving stability for the operations. However, all operations are known to run faster with lower precision representation, like float16 (NVIDIA, 2023). In addition to the speed up, float16 takes half of the memory to represent a float, compared with float32. So, if instead of using float32, the implementation switches to float 16, it will be faster and lighter. Nevertheless, some specific operations such as accumulators, become unstable when performed under low definition representations. To get the best of those two worlds, stability from float32 and speed from float16, comes up the mixed precision. The idea is simple: those operations that are safe to be run in float16, are downcast to float16, and the others are kept as float32. In PyTorch this is called autocast, and the decision of down-casting or not a tensor is automatically taken by the library. That said, as some operations are downcast, the gradients, that are usually small, may underflow when represented in float16, moving their representation to zero. To prevent this, PyTorch has another tool called GradScaler. It scales up the gradients in float16 to make them representable with this precision, by multiplying the gradients with a scalar while doing the forward, and then, it scales down the values to the proper values before updating the weights. This scalar is actually dynamically computed to ensure that overflow nor underflow of the gradients occur during the training.

As a result, by implementing those changes, the code should run faster with more available memory while keep same performance on the task being solved.

3.3.3.2. Improvements over vocab chars

There are some modifications with respect to the vocabulary used by the model. Since predictions are made at character level, the vocabulary is made of characters. In this sense, previous work used a fixed vocabulary with 215 chars on it, including the Start of Sequence (<SOS>) and <EOS> tokens. This was modified to keep only those characters that do appear on the training set, doing it on the fly. This cuts the size of the vocabulary to 150, which is an important change, since representing outputs that are not actually present in the output makes the problem unnecessary more complex and adds nothing to the model. In addition to this, a new character was added to represent out of vocabulary tokens: that is, tokens that may appear in the validation or testing set but are not present in the training set are now represented with this character: ❌. This is specially important when encoding validation and testing examples, and even more in productive environments in uncontrolled, real-world conditions. Previous version just skipped out of vocabulary characters, which means that a single out of vocabulary character will shift all the labels one place. However, the model probably will output something for this unknown character, and this will induce a larger error. This scenario wasn't actually found in Chavat's implementation since the larger vocabulary covered the characters from all the dataset. However, since the vocabulary now has changed, the bug was fixed to prevent it in the future.

Related to this, out of vocabulary token can be learned only if there are some out of vocabulary characters in the training set. To this end, those characters that appeared only once in the training set were removed. This also helped cutting the number of possible outputs, since characters that only appeared once in the training set will probably not even be learned. Choosing single occurred characters to be removed is a discretionary decision, and this could be cross validated: maybe the optimal minimal number of occurrences could be greater. This change removed the following 16 chars from the vocabulary, that only appeared once in the training set:

\t « ¶ × æ è ë ù Í œ º ☑️ ♠ ✅ ➡️ U+2591[1]

[1]Can not display this emoji. It is called "Light shade" and U+2591 is the Unicode value

Upon examining the single-occurrence characters, it appears evident that a majority of them are likely due to errors in the labeling or encoding process. Additionally, a pattern emerges regarding the accent marks. Several of the infrequent characters were found to be incorrect accent marks, such as in the following case where *ì* was used instead of *í* and *ù* was used instead of *ú*:

*honestos de todas las condiciones sociales, de las convicciones pol**ì**ticas, sin ning**ù**n tipo de*

The examples using those wrong accent marks were manually checked, and all of them, according to the context of the label, looked like errors from the annotation process. So, they were fixed just by replacing wrong accent marks with the correct ones, for the five vowels, lower and upper case. This has two implications: reduce the vocabulary size, and increase the occurrence of accent marked vowels.

As a result of described changes, a vocabulary of 132 tokens is generated, including <SOS>, <EOS>, and out of vocabulary token, which implies 38.6% decrease relative to the original fixed vocabulary, and hence a simpler output.

3.3.3.3. Image Augmentation

Data augmentation is a well-known technique which consists of artificially increasing the training set size. When the inputs are images, this is usually referred to as Image Augmentation. The goal is to do this automatically, by performing small modifications to the image such that the labels are still valid. A small change in some pixels value could not affect the label at all, may end up with an image that looks the same, but, at the pixel level, it is a different image. The images are modified on the fly, so that it does not increase the storage needs and potentially allows to provide a different image on each step. Those modifications are also called augmentations. When a batch is sampled, before feeding the model for training, a random choice is made regarding whether to apply those augmentations. This choice is controlled by a hyperparameter `augmentation_portion`. Through several experiments, this parameter was validated and consistently produced improved results with larger values. Consequently, in practice, this parameter was set to 1, leading to augmentation being applied to all batches.

associated with it.

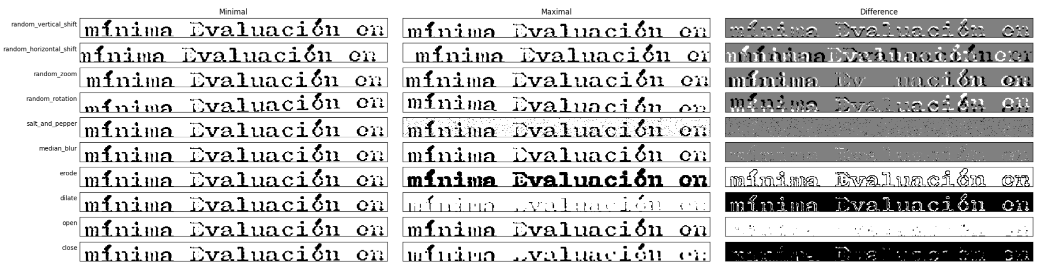


Figure 3.4: This image shows each transformation applied to the same image. From top to bottom those are: `random_vertical_shift`, `random_horizontal_-shift`, `random_zoom`, `random_rotation`, `random_salt_and_pepper`, `median_blur`, `erode`, `dilate`, `open` and `closed`. First column shows the results for minimum values for each transformation. Note that for the last six rows, the minimal change is not defined, so, the original image is showed for those cases. In the middle, images are transformed according to the maximum values for each transformation. The third column shows the difference in pixels between image in maximal column and minimal. It shows how the pixels are modified in different ways according to different transformations. The ground truth for this example is "*mínima Evaluación en*".

This technique is so standardized that most of the packages to work with images already provide some functions to perform it. However, for this case, since the images were so particular and the noise included in them was not that common, instead of using standard packages, the augmentation was implemented from scratch with OpenCV and Numpy.

The following functions were implemented to augment the training examples. Each of them have some hyper parameters that limits its behaviour. Manual inspection of some extreme cases, such as those shown in Figure 3.4, were used to determinate the maximum and minimum parameters used in the transformations, with conservative values.

- **`random_horizontal_shift`**: This function shifts the image horizontally, padding with ones if needed to fill it. To control the maximum shift values, there are two parameters telling the maximum and minimal portion of the image width to shift. Then a random value is sorted following a uniform distribution in this range, and the image is horizontally shift this is portion of pixels. Those levels were set to -0.025, that means shifting the image to the left by 2.5% of the width pixels, and 0.025, for 2.5% to the right.
- **`random_vertical_shift`**: This is the same than before, but for vertical shifts. In this case, the portion is with respect to image height and is set

to -0.05 and 0.05, which corresponds to up and down shifts of 5% of the height.

- **`random_zoom`**: With this function, images are zoomed in or zoomed out. There are also some limits to the zoom, which are set to -0.1 and 0.1. In this case, the limits are the portion of pixels for the largest dimension (height or width). Negative values are for zoom out, that is, the image is reduced and new pixels appear in the border to paddle it, letting the same size. Positive values are for zoom in: in those cases, the image is enlarged and external borders are lost. The definition of *zoom* here is not formal, just to illustrate the concept. The values are sorted in range $[-0.1, 0.1]$.
- **`random_rotation`**: Since examples are not all perfectly aligned, the goal here is to add small rotations to the examples. Since they are randomly applied in both directions, they may end up compensating the wrong initial alignment, they can modify a perfectly aligned example a little bit, to emulate a rotated one, or in the worst case, it could increase the miss alignment. The rotation is chosen uniformly random -1.5 to 1.5 degrees. The rotation is performed with the center of the image as the center of the rotation. Note that this operation has little effect for short sequences but becomes more dangerous for the longer ones, since the rotation may imply that some chars end up out of the image.
- **`salt_and_pepper`**: This is a common noise in digital images, and was intended to be reduced while preprocesing the dataset. However, it still does appear. It is expressed as some pixels that randomly change the value to white or black. In this case, there is a single parameter that controls the portion of pixels to be randomly set as 1 or 0. This parameter is a value between 0 and 0.05, which means that, in the worst case, 5% of the pixels are overwritten: half of them with 0 and the other half with 1. However, the value randomly chosen to overwrite a pixel may be exactly the same that the pixel already has, so, in the practice, it is lower than 5%.
- **`median_blur`**: The goal is to apply a median blur filter to the image. This median blur can sometimes improve the image quality if there was some noise on it, but it can also damage it. Here the parameter that controls how this augmentation is applied, is the kernel size, which is 3 with a probability of 2/3 and 5 with a probability of 1/3.

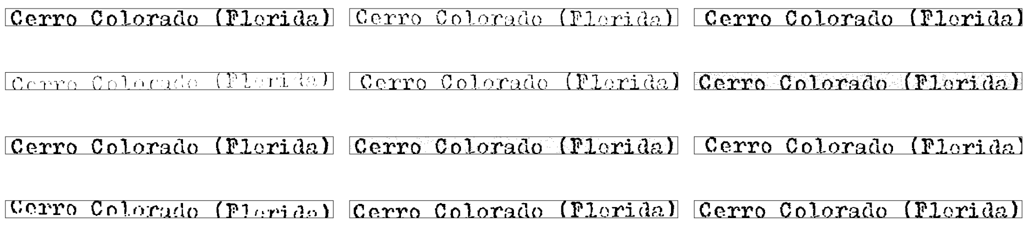


Figure 3.5: The top left image is the original one, while all other variants are the result of randomly modifying that image. Each image was created independently from each other. The ground truth for this example is "*Cerro Colorado (Florida)*".

- **`erode`**, **`dilate`**, **`open`** and **`close`**: Those are morphological transformations. In this case, any of those operations are applied to the image, with kernel size of 3 or 5 that is picked at uniform random. Those transformations are really similar to the noise observed in the dataset, so, for images with good quality they may simulate noised images, and for bad quality images, they may help to improve it. However, and once again, some times with bad quality images and certain transformations may lead to useless examples.
- **`no_op`**: this is just no operation, and is a special case to let the chance that images are not modified at all.

Those transformations were divided into two groups, according to the effect they have, and this distinction also defines the order in which operations are applied. The first group is integrated by the vertical and horizontal shifts, the rotation and the zoom. Those are transformations that apply globally to the image. The structure of the image, its chars, are barely affected. Which of the four operations are applied and in which order, are two decisions taken at random for each image, giving even the chance that no operation is performed.

Then we have the `median_blur`, erode, dilate, open and close operations. Those do affect the structure of the image since are applied locally to each region of the image. Applying more than one of them may cause troubles, since one can remove the effect that the previous added, or both can potentiate issues, like when eroding and opening are combined. So, for this second group, only one operation is chosen at random, to be applied at the image, or no operation if `no_op` is chosen. Since both, the first and second group of augmentations are independently applied, the probability that no operation is applied in any of the groups is 1/25.

On the top of this augmentation pipeline, there is a global parameter called `augmentation_portion` that defines the portion of training examples that are randomly chosen to be run through the augmentation pipeline already explained. So, if augmentation_portion is set to 0.5, then half of the examples are going to be augmented as described above, while the other half are not. As was shown, 1/25 of the examples that go to the augmentation pipeline are not augmented at all, so the global portion of non augmented examples is defined as $\min((1 - augmentation\_rate) + 1/25, 1)$. In Figure 3.5 there is a single example and multiple variations of it that were randomly generated following the process here described.

3.3.3.4. Beam Search

During the sequence generation process, the model picks the most probable token for each step based on its probability. The output sequence is build by taking the best prediction on each step. Instead, what beam search proposes is at each decoding step to consider not only one but b candidates which are build and updated while decoding. For each candidate, it first considers the best b alternatives, compute the probability of the sequence as the product of the probabilities of the candidates on it. So, in a single step, b new candidates are computed for each of the best b candidates that it had up to the moment. Those $b \times b$ candidates are then reduced to the best b, and the process starts again for the next token. This process ends when all candidates reach the <EOS> token or when the sequence reaches the maximum enabled length. This process is illustrated in Figure 3.6.

The b parameter is often called beam size. When set to one it corresponds to exactly the initial greedy approach. As it grows, more sequences are considered during the decoding process, before returning the best candidate. However, the computational cost associated with it also grows. To choose an appropriate value, a validation set is typically used, and it is increased until no significant improvement is observed or the decoding process becomes too expensive.

Note that the probability of the sequence is calculated as the product of the probabilities of each of the tokens that compose it. Since probabilities are values between 0 and 1, larger sequences naturally leads to smaller probabilities. To alleviate this problem, those probabilities can be normalized by multiplying them by $\frac{1}{len(sequence)^{\alpha}}$, where $\alpha \in [0, 1]$. This parameter allows to

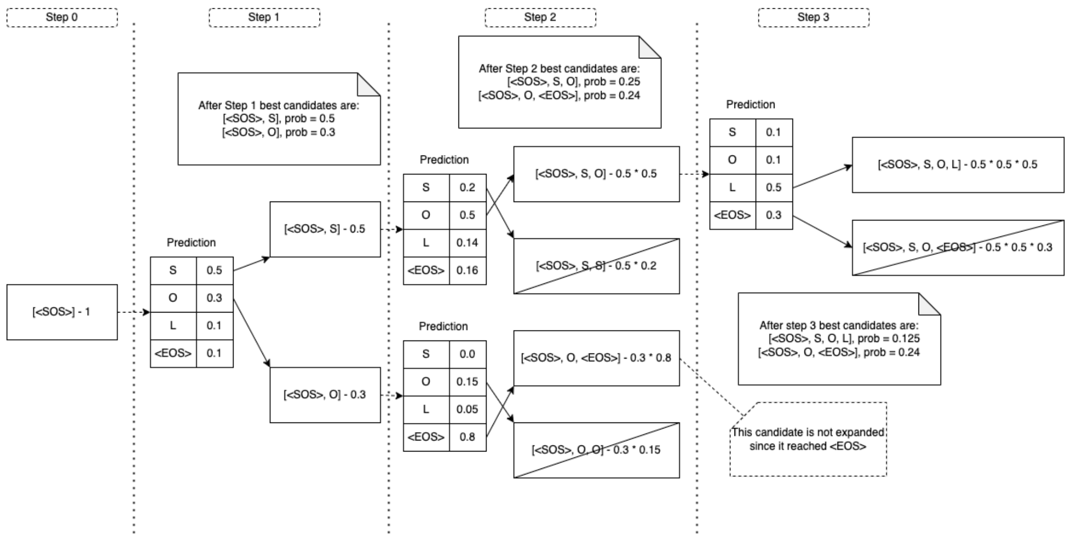

Step 0
Step 1
Step 2
Step 3
[<SOS>] - 1
Prediction
S 0.5
O 0.3
L 0.1
<EOS> 0.1
After Step 1 best candidates are:
[<SOS>, S], prob = 0.5
[<SOS>, O], prob = 0.3
[<SOS>, S] - 0.5
[<SOS>, O] - 0.3
Prediction
S 0.2
O 0.5
L 0.14
<EOS> 0.16
Prediction
S 0.0
O 0.15
L 0.05
<EOS> 0.8
After Step 2 best candidates are:
[<SOS>, S, O], prob = 0.25
[<SOS>, O, <EOS>], prob = 0.24
[<SOS>, S, O] - 0.5 * 0.5
[<SOS>, S, S] - 0.5 * 0.2
[<SOS>, O, <EOS>] - 0.3 * 0.8
[<SOS>, O, O] - 0.3 * 0.15
Prediction
S 0.1
O 0.1
L 0.5
<EOS> 0.3
[<SOS>, S, O, L] - 0.5 * 0.5 * 0.5
[<SOS>, S, O, <EOS>] - 0.5 * 0.5 * 0.3
After step 3 best candidates are:
[<SOS>, S, O, L], prob = 0.125
[<SOS>, O, <EOS>], prob = 0.24
This candidate is not expanded since it reached <EOS>

Figure 3.6: This figure illustrates the beam search process with a toy example where the vocabulary contains only 5 tokens: "<SOS>", "<EOS>", "S", "O", and "L", and a beam size of 2 is used. It starts by feeding the "<SOS>" token to the model, which generates the most probable next tokens conditioned on this initial input, resulting in two candidate sequences: "<SOS>, S" and "<SOS>, O". Then, each candidate sequence is fed back into the model to generate the next token, i.e., predictions are conditioned on the full sequence generated so far. For each of them, the two best continuations are considered. At this point, there are 4 candidates, which must be pruned to keep only the two most likely sequences. Candidates "<SOS>, S, S" and "<SOS>, O, O" are discarded since they have the lowest probabilities among the four. In the next step, the two remaining candidates are expanded: one of them already ends with the "<EOS>" token and is therefore not extended, while the other is used to generate two further continuations. After that, we have three candidates: "<SOS>, O, <EOS>", "<SOS>, S, O, L", and "<SOS>, S, O, <EOS>". Again, only the two most likely alternatives are retained based on their cumulative probabilities. If the maximum sequence length is reached at this point, the best of those two is returned. Otherwise, the generation process continues until all sequences reach the "<EOS>" token or the length limit is met. This example does not apply sequence length normalization, which should be done at each step.

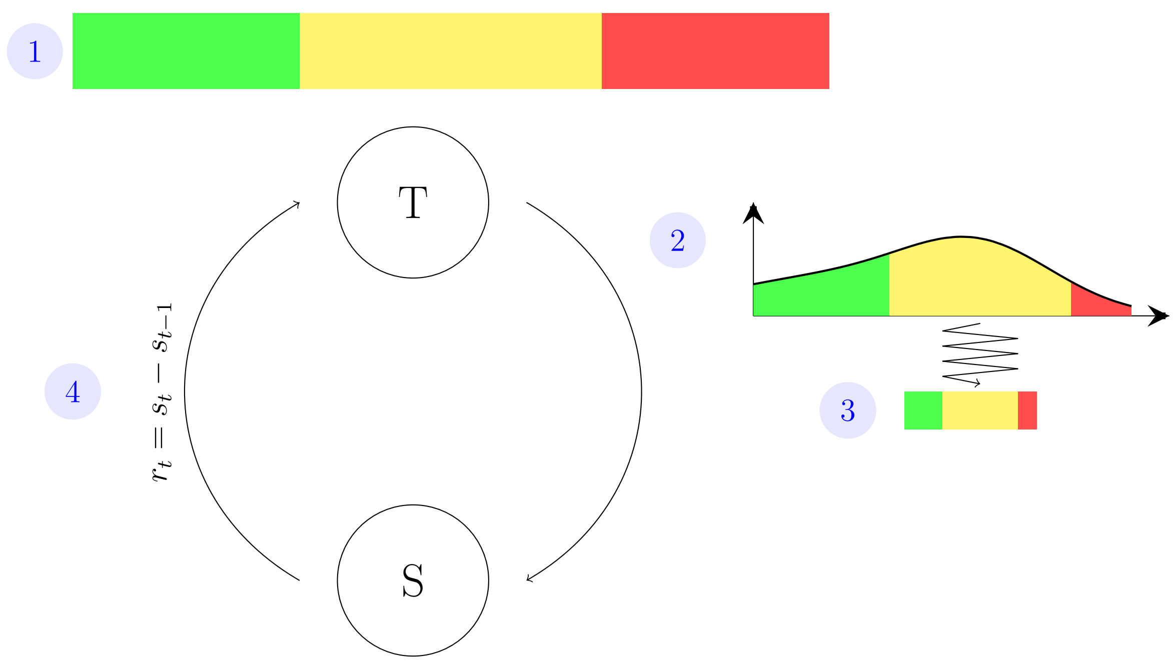


Figure 3.7: This schema summarizes the training loop in TSCL for supervised scenarios. 1: The instances are splitted into sub tasks, in this example there are three: green, yellow and red. This is done only once at the beginning. 2: The Teacher gives a distribution function over the sub-tasks, which is learned in the same process. 3: A batch of examples is sampled, according to the distribution over sub tasks. This is given to the Student to update its model via standard gradient descent algorithm. 4: Finally, a reward is computed as the difference in some performance score with respect to previous iteration, over a held out dataset. This reward is used to update the Teacher's sampling function. Now the loop continues at point 2.

balance between different sequence lengths. When set to zero, it removes this term and just the original beam search strategy is applied, and as it grows, larger sequences are more likely (Y. Wu et al., 2016).

This strategy is used only for the decoding step at prediction time, no matter how was the model trained.

In the next chapter we will evaluate the improvements proposed up to this point with respect to the base model, which turned out to be of 16% in terms of CER. But, before moving on to the experimental Chapter 4, is time to introduce the TSCL formulation of the problem in the next section.

3.4. TSCL Formulation

This section discusses how this OCR task can be formulated as a TSCL problem. Let's briefly recap how TSCL roughly works in supervised scenarios before diving into the framing of OCR as a TSCL formulation.

At the beginning, examples are grouped into sub tasks. Subsequently, the Teacher selects which sub tasks to be sampled, creating a distribution over the selected sub tasks based on its Q-function. A training batch is then sampled following this distribution and provided to the Student. The Student utilizes this batch to update its model's weights using the standard gradient descent algorithm for several epochs. Then, the Student computes metrics over a validation sample of those sub tasks and returns them to the Teacher. These metrics serve as the basis for computing rewards, which the Teacher employs to update its Q-function. This entire process is illustrated in Figure 3.7.

One of the key points when framing a problem in TSCL is to define which are the sub tasks. The task addressed here is to convert a sequence of pixels into a sequence of characters. This task does not provide an intrinsic distinction between sub tasks. We must define a function that assigns each instance in the dataset to a sub task, not only in the training set, but also in the validation set, and for both sets all sub tasks must have some examples from each on it. Otherwise, if a sub task does not contain instances from the validation set, it is impossible to give feedback to the teacher on how good is the Student doing for this sub task.

Ideally, all the examples in a sub task should share a common difficulty to make it possible for the Teacher to choose from easy or hard examples, depending on the sub task it chooses to teach the model.

3.4.1. Sub task definition

This section describes and analyzes some alternatives explored to define the sub tasks in this work, including the one that was finally used.

3.4.1.1. Clusterize raw pixels

Clusterizing the dataset to define the sub tasks is already mentioned in the bibliography (Guo et al., 2018). This is an approach used to apply TSCL on CIFAR dataset by El-Bouri et al., 2020, which is also a supervised task. To build the sub tasks, authors apply Principal Component Analyzis (PCA) to the raw pixels, just to reduce the dimension, and clusterize the examples into k sub tasks with k-means.

The advantage of this method is that there are few parameters to tune:

since raw pixels are being clusterized with k-means, similar images are expected to be together. However, the dataset used in our work is made of larger images than CIFAR and irregular in terms of shape, so, it turned out to be too expensive to compute this PCA transformation.

3.4.1.2. Clusterize features

In this case, the key idea was to compute inexpensive features for each example based on both the text and the image: the number of characters, the number of words, the proportion of black pixels in the image, the ratio between image length and the number of characters, the count of different letters in the sequence, the number of black regions in the image, the image's area, among many others. These features were then correlated with the loss for each example, obtained from a surrogate model: an instance of the base model trained over 10 epochs. As explored in Chapter 2, similar to many of the alternatives, an example's difficulty can be related to its loss value. Instead of explicitly computing this loss, we aimed to cluster based on features highly correlated with loss values. This approach satisfies the constraint that examples with similar difficulty belong to the same task, while avoiding the explicit computation of the loss once the features are selected.

Despite of using different features, changing the number of tasks and some additional parameters, this approach still had two main issues that make it hard to use: first, since training examples were used to fit the k-means, every training example has a sub task assigned, but it didn't happen for validation examples. Some training examples seem to be so particular, in terms of the features built, that exclusive clusters were created for them. This was the worst part, since it does not meet our requirement that all sub tasks must have some representatives from both, training and validation set. The second point was that this approach is not that efficient: there are several features which could be built to fit the cluster, evaluate each of them and its combination is hard in practice. Finally, once built it is not easy to understand what does each cluster represents, which later will make it harder to interpret what is going on. So, this was also discarded, but, the insights taken while analyzing the features were valuable for the project.

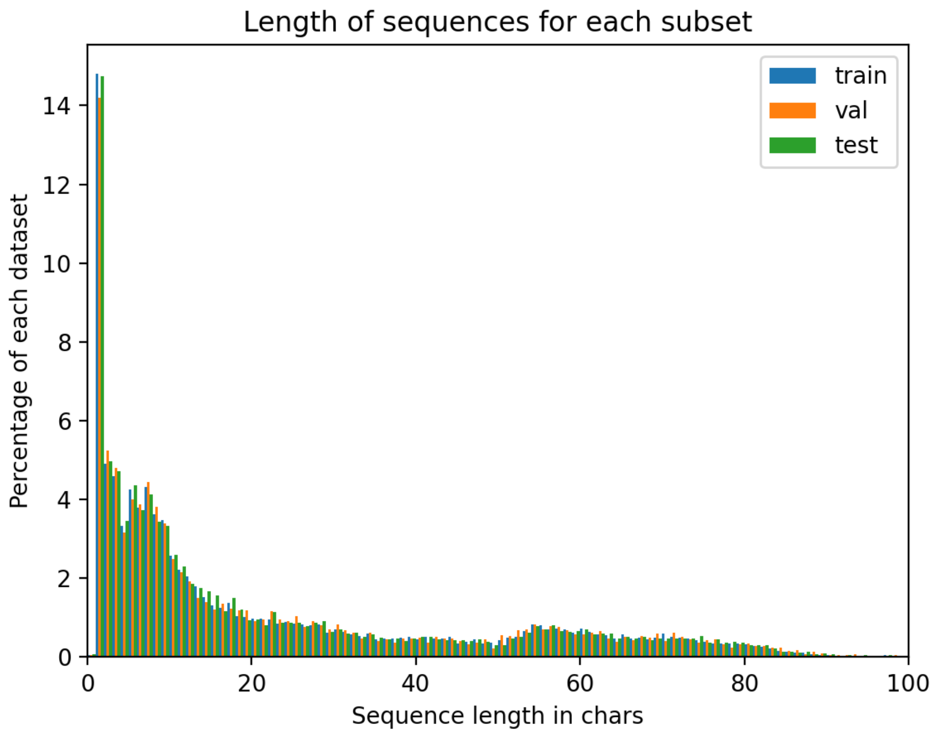


Figure 3.8: This plot show how are the text length distributed for training, validation and test dataset. There are some instances with 100 or more chars on it, they are not represented in the plot but only represent 0.14% of the examples in training and validation sets, and 0.13% for the test set. There are also some examples without characters that also represent a small portion in all sets.

3.4.1.3. Split according to sequence length

From the previous analysis over different features, one conclusion was that, the lower the dimensional space where to clusterize is, the less unpopulated clusters in the validation set, so reducing the number of features to consider when splitting into sub tasks helps to decrease the number of empty clusters in validation set. Therefore, trying to use as few features as possible, considering just the sequence length is a solution.

The number of chars in the sequence was one of the highest correlated features with the loss function among the analyzed features, which is in part due to the loss definition: sum of errors for predicting each character. The more characters in the sequence, the more error will accumulate, leading to a larger loss. In addition to this, all three dataset show similar distributions in terms of number of characters, as can be seen in figure 3.8: if sub tasks are splitted according to this criteria, then both training and validation examples will cover all the sub tasks. Using sequence length to sort the training examples in the context of sequence to sequence models is something widely used in the literature (Matiisen et al., 2019).

The distribution of the length is a highly skewed distribution, and the same happens for other features, so, when they are are combined, k-means results in some huge clusters that contains lots of examples, and then some other with just a few of them. That is, the instances are not uniformly distributed among this feature space. To mitigate this effect, we simplify the feature space to just the sequence length and instead of using k-means, the sub tasks were assigned relative to the percentiles in the distribution of sequence length: by definition, each percentile contains the same number of examples, so, this approach should balance the number of examples at each sub task.

The algorithm to assign examples to sub tasks was the following: first, compute the percentile that represents each sub task: sub task 0 represents percentile 0, sub task 1 represents percentile x, where x is the result of $100/n\_subtasks$. Then, with the training distribution of sequence lengths, each reference percentile is computed. Finally, for each example, no matter if it belongs to training or validation, the number of characters in it is counted, and it is assigned to the sub task whose percentile is nearest. In case of ties, that is, an example that is at the same distance from two sub tasks, then the sub task is randomly assigned to one those two. This looks like a corner case, but it turns out that, given the skewed distribution of sequence length, and the fact that lengths are integers, it happens quite often. In fact, if ties were broken with the lowest sub task only, some sub tasks would keep empty in both sets when using 20 sub tasks, that is the proposed default manually chosen.

After following this algorithm, the distribution is near uniform, but has some details to keep in mind:

- **sub task 0 is small**, since it is only made of empty examples, which are really few, this sub task contains really few examples.
- **sub task 1 and 2 both contain single-character sequences**, as a result of the highly skewed distribution, plus the discrete values for the lengths, it turns out that most of the examples at sub task 2 come from the ties randomly broken against sub task 1.
- **The number of chars grows with the sub task id**, as a result of

the sub task assignment process, the shorter examples are at sub task 0, and the larger ones are at sub task 19. This is a great point that will be used to interpret and analyze the results later.

3.4.2. The reward

For the reward definition, the difference in loss for a validation held out, on each sub task is used. Despite of TSCL framework admits other error metrics to define the reward (Matiisen et al., 2019), the loss function is a good option since it is the error being minimized by the model. Other metrics, such as CER were also validated, but it turned out to be far more expensive in terms of compute and did not show significant improvements in results, as we will later show. The only benefit that such metrics provide was that they are upper bounded since they are in range [0, 1]: the loss function do not has this desirable property, since it does not have an upper bound, and hence, the difference in loss between two consecutive steps is unbounded too. Those changes in the reward during the training may lead to instabilities while learning the Q function (van Hasselt et al., 2016). This is usually mitigated with clipping the reward (van Hasselt et al., 2016; J. Wu et al., 2018). In this case, the reward is clipped in range [-max_reward, max_reward] and linearly converted from this range to [-1, 1], with max_reward defined as 3 after some preliminary experiments. The main advantage for this is that it is cheap, straightforward and easy to understand and interpret. However, since the loss function is expected to be minimized at each step, the large differences in loss observed at the beginning of the training, are reduced as the training progresses. This is an area with room for improvement: to dynamically adapt the reward as training proceeds (J. Wu et al., 2018).

One more thing: when sampling the sub tasks to validate in the held-out and compute the rewards, all the sub tasks are uniformly sampled with the same number of items on each. Since each sub task is built based on percentiles, they should be also balanced and so, if we took the same number of examples from each, the resulting distribution theoretically is the same as in the original distribution.

3.4.3. The Teacher

The Teacher learns the Q function at the same time that the Student is being trained to perform OCR. This function defines which examples should be used to train in the next step.

In the RL formulation, choosing which sub tasks to sample for training are the actions. Since this is a supervised problem, the batched approach of TSCL (Matiisen et al., 2019) must be taken: the Teacher's action is not to choose a sub task but a probability distribution over sub tasks. $a_t = (p_t^1, \ldots, p_t^n)$, where p_t^i corresponds to the probability of task i at time t. As a result, instead of choosing a single sub task to train, the Teacher predicts a distribution over sub tasks, and samples them using this distribution, plus forcing that at least one example of each sub task is chosen for each step. Predicting a distribution rather than single action is crucial: otherwise, the training would be done only over instances of a single sub task at each step, which leads to instability in the Student's training. Despite of being already reported (Matiisen et al., 2019), this phenomena was also empirically verified in early experiments of our work.

Previous to each training step, in addition to sampling the examples that are going to be used for training, validation samples are also build. In this case, all sub tasks are uniformly sampled from validation dataset, ensuring that at least one example is chosen for each sub task. This validation dataset was sampled from the training dataset and is used as a held-out to evaluate the Student's performance after the training step, and compare it with the obtained in the previous step to compute the reward.

For the Q function, a tabular representation is used, since there are no states, and the number of actions is small. The update rule used in this job was the following:

$$Q_0(a) = 0 \quad \forall a \in Actions,$$

$$Q_{t+1}(a) = \alpha r_t + (1 - \alpha) Q_t(a)$$

where a is the action and $\alpha \in (0, 1]$ is the learning rate: higher values of α means that recent rewards are more important than the initial rewards. In the limit, if $\alpha = 1$, then only the very last reward is used to overwrite the Q function. On the other hand, small values of α puts more weight to the previous values of Q and gives a small weight to the current reward. At this point, it is important to note that the reward function is a non-stationary function, so initial rewards should not care too much to approximate the Q function after several steps. Therefore, this parameter α is expected to be close to one.

Matiisen et al., 2019 report that, choosing actions according to the absolute value of Q improves the results, since it helps to prevent catastrophic forgetting. Sub tasks with large but negative Q values are those in which the Teacher is predicting a decrease on the students performance over them. Using the absolute value allows the Teacher to choose not only those sub tasks that are being learned the most, but also those that are being forgot the most.

3.4.4. Exploration vs Exploitation

In each step, the Teacher must balance between exploiting what it knows up to the moment, and pick the best known action or to try something different, and explore alternative actions hoping to find a new and better alternative action. Such balance is important in any RL task (Sutton and Barto, 2018). Typically the agent, in this case the Teacher, should start exploring a lot, since it knows nothing, and then starts to exploit what it knows. Once it converges, it could just pick up the best action up to its knowledge. However, this particular framing where rewards distribution changes over time, is said to be **non stationary**, since the reward is related to what the Student learned or forgot with respect to the previous step, and since this reward is based on a an error metric that is expected to decrease fast at the beginning and then slow down, the rewards distribution will be changing. Even when the model already tried all the actions several times, and learn that some action maximizes the expected value, it still needs to keep exploring because such optimal action may change over time, as the reward distribution changes.

In this work, two alternatives were considered to balance those alternatives: epsilon greedy and softmax.

3.4.4.1. Epsilon-greedy

This policy balances exploration and exploitation, by picking up the best known action with a probability of epsilon or otherwise, taking a random action. Since this must be implemented as a distribution over all the states, this distribution is computed as follows:

$$p(a) = \frac{1-\epsilon}{n_{actions}} \quad \forall a \in actions, a \neq a^*$$

$$p(a^*) = \frac{1-\epsilon}{n_{actions}} + \epsilon$$

where a^* is the action that maximizes current Q function, hence, the best known action up to the moment.

3.4.4.2. Softmax

This policy consists on applying a softmax function to the Q function, to convert it into a probability distribution. It is defined as:

$$p(a) = \frac{\exp^{Q(a)/\tau}}{\sum_{i=1}^{n_{actions}} \exp^{Q(i)/\tau}}$$

where τ is a parameter often called temperature, which regulates the smoothness of the distribution, usually it is set to 1. For smaller values of τ, the resulting distribution is sharper and looks more like the epsilon greedy, with higher probabilities for maximizing actions and lower for the rest. On the other hand, when this parameter is greater, the distribution tend to be more soft and the probabilities between actions are more similar.

3.4.5. Steps and epochs

TSCL is made of two main components: the Teacher, that is fitted following a multi armed bandit algorithm, and whose progress in time is measured in terms of steps. And the Student, that is an OCR model that, on each step, is trained following standard SGD. This Student model is the one that matters at the end of the day, and is also the one whose performance is important to compare against the base: a simple OCR trained with standard SGD. To make sure this comparison is fair, the number of training epochs are still computed and reported for TSCL, even when they make no sense for the framework itself: just to allow fair comparisons with the base.

In traditional training, an epoch is concluded when all training examples where used to fit the model one time. It is usually implemented as a list with all the dataset that is shuffled before starting the epoch, and then it is iterated extracting batches of data from this shuffled list. Once the iteration over the training set is done, an epoch is completed. This ensures that each example was exactly seen once in the epoch.

As reported in the base, the OCR model needs a few epochs to learn the task, however, the number of steps to actually learn the Q function is usually higher than that, in the order of, at least, hundreds, depending on the problem complexity. So, following those observations, one can conclude that steps should be smaller than epochs. But how long? For this work, it was arbitrary defined to 2.5% of the training set by default: so, each step samples 2.5% of the training set, and use these to feed the model. Once 40 steps are taken, it means that the model was trained with as many examples as it would be fed with the traditional approach: the full dataset. This is how an epoch is defined in our work.

Despite of both epochs imply that the OCR model was fed with n_training examples, this comparison is still not perfect: in traditional training, each example was given only and exactly once, while in TSCL, since the data for each epoch is sampled 40 times, once for each step, some examples may appear more than one time, which also implies that some other are not seen at all. What is more, since larger sequences contains more characters, then, in terms of characters, during an epoch the model could be trained more often with larger sequences, and therefore, more characters are seen by it while training a single epoch. In addition to this, due to rounding errors when computing the 2.5% of the training set, after 40 steps the model is actually fed with few more examples than the examples fed with traditional SGD.

3.4.6. The training loop

Finally, the training loop shows how both, Teacher and Student interact with each other. Despite being defined by Matiisen et al., 2019, this loop is not clear in their work. However, they shared their code in github[1], which allowed to dig into it and better understand this process. This TSCL loop in Figure 3.9 may look simple, but there are some practical implications on its implementation: all the frameworks used to implement deep learning, are designed and optimized to run like in the traditional case. A full dataset that is usually shuffled and given in random batches to the model. There is out-of-the box parallelization used to fetch and pre-process the data on CPU, before

[1]Their code is available here https://github.com/tambetm/TSCL.

Listing 3.1: Training loop for traditional training.

```
base_model = create_model()

for e in range(TOTAL_EPOCHS):

    model.fit(train_full)

    # epoch ended;
    # check the performance over the full data set
    base_model.eval(val_full, full_metrics=True)
```

Listing 3.2: Training loop for TSCL.

```
teacher = get_teacher(train_full, val_full, n_sub_tasks=20)
student = get_student()

for e in range(TOTAL_EPOCHS):
    while remaining_steps_on_epoch > 0:
        # sample instances to train
        train_i, val_i = teacher.choose_task()

        # fit breaks if  remaining_steps_on_epoch <= 0
        # This is done with SGD + ADAM
        student.fit(train_i)

        loss = student.eval(val_i)

        # Compute reward and update Q function
        teacher.learn(loss)

    # epoch ended;
    # check the performace over the full validation set
    student.eval(val_full, full_metrics=True)
```

Figure 3.9: This figure shows the pseudocode used for the training process in both: base (top) and TSCL (bottom) implementations.

feeding the model that runs in GPU. As a result, the GPU is expected to be at full capacity training all time, while CPU is used to let everything ready to fed the model in GPU, until the epoch ends. Then the model is evaluated over the validation set, with a small penalty for switching contexts, and then it starts a new epoch again. However, with TSCL, the interruptions and context switching occur more often because it must be done after each step, and each step represents 2.5% of an epoch, then there are 40x extra context switches and validations. To reduce this cost, two simplifications were done to the inter step validation rounds:

- Instead of evaluating the full validation dataset, a portion of it is sampled: exactly the same portion that is being used to build the sample, 2.5% of the dataset. So, at the end of an epoch, the model would be fitted with as much examples as in the training set, and also validated over the same number of examples that there are in the validation set.
- When validating, instead of letting the model predict characters until a maximum of 105, it is dynamically modified for each example to let the model predict up to 20% more of characters than the number of characters in the sequence. This is an important change: as shown in Figure 3.8, there are lots of instances with fewer characters, like 5 characters for instance, that instead of allowing the model to make up to 105 predictions in the worst case, now it makes up to 6 character predictions. This is particularly important on early stages, when the model is barely trained. This change in validation was only done for the validation step at TSCL when computing the reward: the reported metrics after each epoch and as final results are all computed using the original hard limit of 105 characters.

This is an overhead from an implementation perspective, and is the main reason why so much care and effort was put to improve and optimize the base model in terms of speed.

Now that the TSCL framing of the problem is presented and discussed with its initial values, the next step is to see how those affects in the practice. In the next sections we will first provide some insights on how the proposed improvements work to later start with the experiments on the TSCL implementation.

Chapter 4

Experiments

This chapter documents the experiments performed in this work. It is divided into two sections. In the first, we carry out some experiments to validate and understand the improved model introduced in Section 3.3.3. These changes aimed to establish a stronger base before actually running experiments on TSCL. Then, the second section documents the main results from experiments over TSCL implementation.

By the end of this chapter, we aim to have a better understanding of the initial question of whether TSCL can improve results compared to traditional training methods in this particualr setting.

4.1. Experimental design

As a general rule, all experiments are run for 10 epochs, even when more than 10 epochs are needed to obtain the best results. However, this small number allowed for cheaper experiments while the conclusions for different configurations do not change after this number of epochs.

After each epoch, the model is evaluated over a validation set. The best CER (Character Error Rate) obtained from these evaluations is reported, along with the corresponding epoch, consistent with Chavat's methodology (Chavat Pérez, 2022). The reported results for each experiment are based on a single execution due to the high computational cost and time involved in running multiple iterations. This approach, while practical, means the results may not fully capture the variability in the model's performance. The experimental parameters are consistent with those used in (Chavat Pérez, 2022), unless stated otherwise: learning rate = 0.0001, teacher forcing = 0.85, and batch size = 8.

For both experimental sections, the base ones and the TSCL follow the same idea: first, try modifying a single parameter to understand just its effect, and then use the lessons learned in those steps to change all parameters at the same time to search for the better configurations.

4.2. Base: traditional training

We conduct these experiments for two purposes: first, to validate in the practice the changes discussed in Chapter 3 with regards to the implementation described by Chavat. Second, to establish a reference for this work.

4.2.1. Batch size

In Chavat's work, the batch size was established as the larger possible, as long as the model fits into the GPU's RAM. However, as part of current work, some changes in the implementation were made. In particular, the usage of autocast, which should reduce considerably the memory usage and hence allow for greater batch sizes which could reduce the training times while also modifying the performance. This experiment aims to validate the effects of using larger batch sizes on both training times and performance, given that both, Chavat and us, use the same hardware.

To this end, the base model was executed with batch size 8, 12, 16 and 20, to check whether it was possible and see the effects. However, batch sizes greater than 12 failed, since the available memory is not enough. Since the base model supported a batch size of 8, setting the batch size to 12 represents a 50% increase with respect to Chavat's implementation. This validates the correctness of the implementation.

For this experiment, the base model was trained using Image Augmentation for 10 epochs, both with the same parameters, except for batch size. The results indicate that while a batch size of 12 is feasible, increasing it leads to inferior performance. As can be seen in Table 4.1, all metrics considered are better when using batch size of 8, as defined by Chavat, and also the training process looks smoother in this case, as can be seen in Figure 4.1. This may be caused because lots of his experiments carried on to set the proper

epoch	**batch size**	**LCS**	**CER**	**train loss**	**val loss**
9	8	**0.723**	**0.295**	**0.723**	**1.966**
10	12	0.708	0.313	0.724	2.120

Table 4.1: This table compares the results after training the same model just with different values for batch size. The best results were obtained with a batch size of 8, which was used to determine the optimal values for other parameters: all decisions regarding model development were made by Chavat based on a batch size of 8. Larger values for batch size could not be tested since they did not fit into GPU.

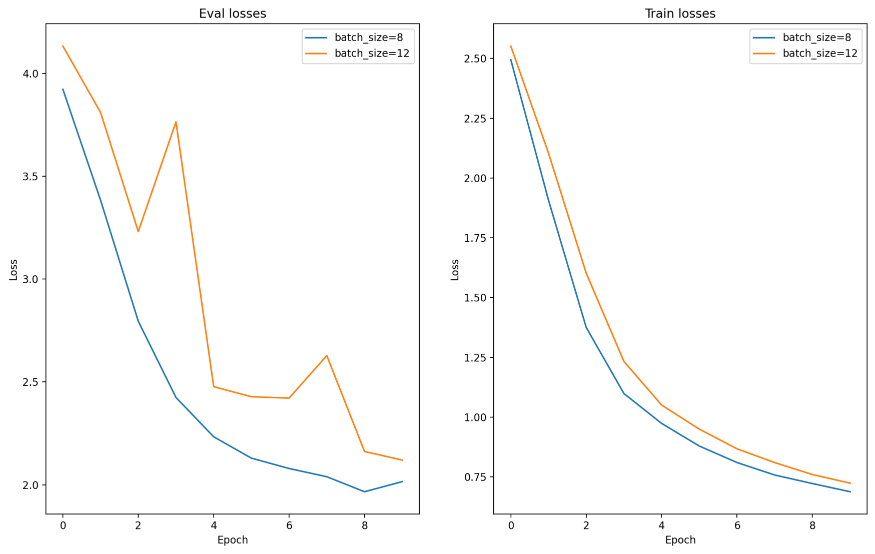


Figure 4.1: This image shows the evolution of the loss function with respect to the training process. Using a batch size of 8 leads to better results for every epoch, and more over, the validation loss is smoother when training with it.

Vocabulary	Epoch	LCS	CER	Train loss	Eval loss
Fixed charset	8	0.713	0.305	0.647	2.082
Computed charset + accents	10	0.716	0.304	**0.542**	2.115
Computed charset + accents + min_count=2	9	**0.721**	**0.298**	0.587	**2.061**

Table 4.2: This table shows the results of running the base model for 10 epochs under different settings with respect to the vocabulary being used, as the incremental improvements proposed over base.

learning rate, architecture and teaching forcing parameters, among others, were validated with a fixed batch size of 8. Now, if batch size is increased, may be the optimal value for those parameters should be re-validated, which is out of scope.

Furthermore, in terms of training times, these were found to be similar in both scenarios. This is a significant issue with the model: lots of the training time is used decoding the sequence, which is performed in CPU, and hence, all GPU-related changes, including a larger batch size, have a poor impact on training times. Therefore batch size is fixed at 8 for the rest of this work.

4.2.2. Vocabulary

This experiment tests the hypothesis that modifying the charset used in the base model will improve performance on the OCR task. The key idea here is that the charset used in the base model was fixed, and contained nearly 30% of characters that are never seen while training, 16 characters that appear only once in the dataset, and several occurrences of incorrect accents marks. Three trials were conducted, one for each of these hypothesis.

The results for vocabulary experiments are shown in Table 4.2. The metrics confirm that, as the vocabulary is reduced, and fixed the accent marks, it works better. As a result, the computed charset + accents + min_count=2 is kept for future experiments.

In Section 1.1 of the Appendix 1 the reader can find an interesting discussion about how the changes in the vocabulary interacts with the Image Augmentation strategy that we will discuss in the next section.

4.2.3. Image Augmentation

To validate the usage of Image Augmentation, the base model was trained with exactly the same version but varying the augmentation rate. In this case,

augmentation rate	epoch	LCS	CER	train_loss	val_loss
1.0	29	0.7643	**0.2534**	**0.311**	**2.151**
0.8	31	**0.7644**	0.2536	**0.277**	2.262
0.5	24	0.757	0.261	0.306	2.176
0.0	15	0.746	0.272	0.34	2.209

Table 4.3: This table shows how the performance changes with different values of augmentation rate. It can be seen that, as this increases, the metrics improve, and additionally to this, the best epoch is also larger. This is explained by the fact that, when using augmentation, is harder for the model to overfit since training set is all time being slightly modified. These executions were run for 32 epochs. Augmentation parameters are not exactly the same as discussed in Section 3.3.3.3 but they are really close to those. It turns out that those parameters were slightly modified after this experiment, to the values presented there which were also used in further experiments.

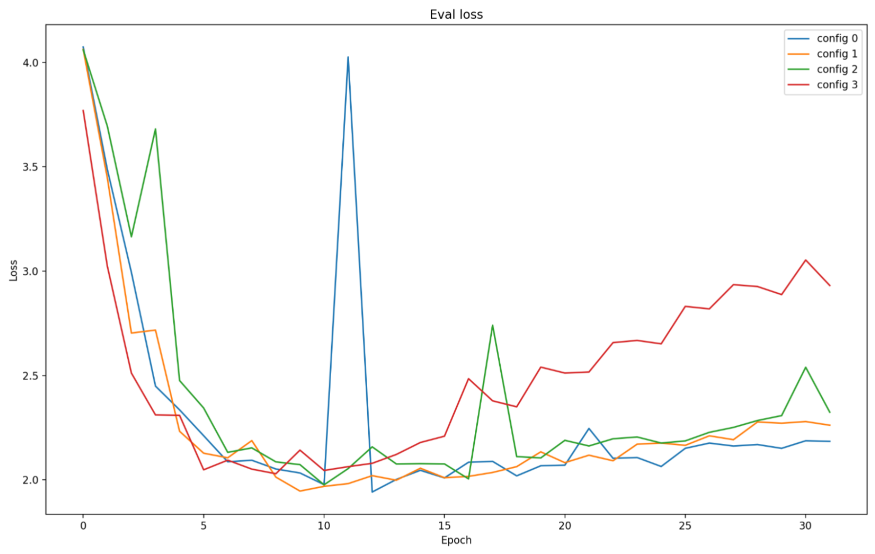


Figure 4.2: This image shows the evolution of the loss function with respect to the training process, for the evaluation loss only. Config 0, 1, 2 and 3 in the image, corresponds to using augmentation_rate 1.0, 0.8, 0.5 and 0.0 respectively. It can be seen that, after some epochs, all configurations tend to overfit the model, that is reflected by an increase in the validation loss. However, the greater the augmentation rate used, the later this overfitting starts.

the model was run for 32 epochs, without the improvements proposed for the vocabulary, discussed in previous section.

Table 4.3 shows the final results for each augmentation rate configuration, and Figure 4.2 illustrates the evolution of validation loss while training. As

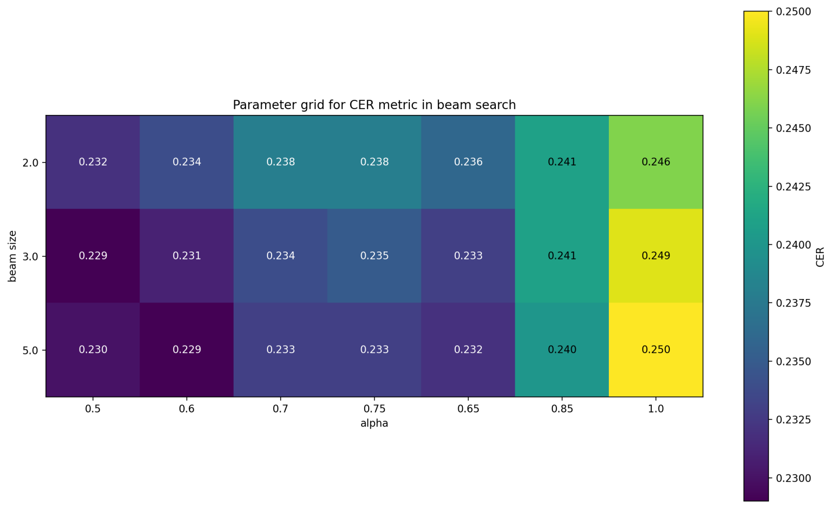


Figure 4.3: This image shows the CER for each combination of alpha and beam size, over 10 % of the validation set, for the base model trained over 20 epochs. As alpha reduces, the performance improves, this is close related with the fact that the dataset contain short sequences on average.

can be seen, as the augmentation rate increases, the model can be trained for more epochs before starting to increase its loss. This can also be seen in the table, as larger values for the augmentation rate corresponds to larger epoch values for the best CER found. This implies that, larger training sessions are required for larger augmentation rates.

This experiment also confirms that Image Augmentation improves results as expected, and a larger augmentation rate leads to better performance. Therefore, we proceed with an augmentation rate of 1.0 for the following experiments.

4.2.4. Beam Search

In this section we analyze the impact of beam search over predictions. The first point is to find a suitable configuration, that is: choose an appropriate beam size and its alpha. Since this heuristic is applied on prediction time once the model is fit, those hyperparameters should not be validated over the training set, since those examples will produce better results for probabilities just because the model was already trained on them. To this end, we take the

beam size	2		3		5	
alpha	**LCS**	**CER**	**LCS**	**CER**	**LCS**	**CER**
0.5	0.785	0.232	**0.787**	**0.229**	0.786	0.230
0.6	0.783	0.234	0.786	0.231	0.786	0.229
0.65	0.781	0.236	0.783	0.233	0.784	0.232
0.7	0.779	0.238	0.782	0.234	0.783	0.233
0.75	0.779	0.238	0.781	0.235	0.783	0.233
0.85	0.776	0.241	0.775	0.241	0.777	0.240
1.0	0.771	0.246	0.768	0.249	0.766	0.250

Table 4.4: This table shows the results for different configuration of beam-search over 10% of the validation dataset, randomly sampled with replacement. This allows for a quick and cost-effective overview of how those parameters behave and interact. CER metric reported here is graphically shown in Figure 4.3. When no beam-search is used, over this set, the model achieves 0.777 in LCS and 0.241 in CER, which implies near 5% improvement in terms of CER for the best configuration.

beam size	**alpha**	**LCS**	**CER**
1	-	0.756	0.260
3	0.5	0.772	**0.244**
3	0.6	0.770	0.247
5	0.5	**0.773**	**0.244**
5	0.6	0.770	0.247

Table 4.5: This table shows the results for different configurations over 30% of the validation set randomly sampled without replacement. The first line, with beam size 1 corresponds to not using beam search, and is reported for comparison. As can be seen, all of them improves the results with respect to not using beam search. However, there is no major difference between beam size of 3 and 5, and same thing is for alpha. Any of those configurations look like good candidates.

validations set. However, to avoid overfitting our hyper parameter selection to this dataset we will sample a fragment of this. Initially, we sampled 10%. Such a small portion allows us to proceed quickly and cover several configurations fast. This is shown in Table 4.4, which shows that small values of alpha appear to be more promising, since they lead to better results. We can also see that, some unappropriated values for alpha, such as 1, end up in worst results than when not using beam search at all. Best results also tend to occur with beam sizes of 5. So, the next thing we did was to double click into this promising region, trying few configurations but each with a larger dataset, to allow for stronger conclusions, so sampling 30% of the validation dataset.

The results for this larger run are shown in Table 4.5. It shows that any of

Max epochs	Beam size	Augmentation rate	epoch	LCS	CER	train loss
50	3	1	35	**0.781**	**0.236**	**0.349**
20	3	1	20	0.769	0.247	0.474
20	1	0 (Chavat)	11	0.74	0.282	-

Table 4.6: Metrics over validation dataset. This table shows that just running for 20 epochs already improves the results compared to what is reported by Chavat, shown in the last row. The row with a beam size of 1 correspond to not using beam search, as discussed earlier. The proposed strengthen base also improves when more epochs are allowed: an improvement of near 16.3% (from 0.282 to 0.236) in CER, and almost 6% for the LCS (from 0.74 to 0.781).

those configurations look like good candidates, so we choose to use a beam size of 3, since using 5 takes larger inference time and differences in performance are not significant. For alpha we use 0.5, since it reports the best results for any beam size in this table, and from the previous table there is also a tendency that lower alphas yields better results. This may be related with the fact that, as alpha grows, shorter sequences are more penalized, since our dataset consists of short sentences on average, a small value of alpha is more appropriate.

This beam search configuration will be used in our best scenario: beam size of 3 and 0.5 for alpha. Beam search will be applied only when evaluating the model, and not on intermediate experiments.

4.2.5. Strongest base

This section showcases the most robust base attained by Chavat's OCR model, which includes the enhancements proposed in Chapter 3 and verified through the experiments in previous sections of current chapter. This robust model was also used as the student in our TSCL implementation proposal.

As previously discussed in Section 4.2.1, the batch size used is 8, since it showed better results while keeping the times unchanged. The vocabulary is smaller since it is now computed on the fly for this specific dataset, accent marks have been fixed and characters that appear only once are replaced with out of vocabulary token, as discussed in Section 4.2.2. The augmentation ratio was fixed at 1.0 based on our observation in Section 4.2.3 that larger values of augmentation ratio tended to yield better results. Finally, beam search is used with a beam size of 3 and an alpha value of 0.5 as defined in Section 4.2.4.

The results in Table 4.6 were computed over two scenarios: training for 20 epochs or for 50. The first is performed to make sure these results are comparable with the ones reported by Chavat: he trained the models for 20 epochs, so, the goal is to understand if the difference in performance can be attributed to the improvements in the model or just because more training epochs are used. The second shows that, for larger runs, the model still improves. This makes sense, as discussed in previous sections, some changes may only have an effect in the long run, like the charset or the Image Augmentation.

If the reader is wondering about the contribution that each of those changes means for the strengthen base, in Section 1.2 of Appendix 1 there is a discussion about it, with further experiments concluding that Beam Search is the most important, followed by the increased number of epochs and Image Augmentation, letting the changes in vocabulary in the last place.

Now that all improvements to Chavat's proposal have been validated, the strongest base is established, and with it, the Student for TSCL is also defined. The next section will focus on TSCL-related experiments to determine the optimal configuration and understand the behavior of different parameters. Consider that, beam search, will only be applied when inferring over the test set for the final evaluation. All intermediate experiments are validated without its usage.

4.3. TSCL

In TSCL there are lots of decisions to validate, and hence, several configurations and hyperparameter combinations to try. This grows exponentially and is unfeasible to explore it all. To make it possible to validate some of the parameters, the following strategy was followed. First, there was a main configuration proposed whose parameters were defined based on some intuitions and validated in preliminary experiments, which is discussed in 4.3.1. In 4.3.2 we validate the effect of Q function comparing the purposed in our initial configuration against a random one, and validate the absolute value over Q function. Then, a single parameter was changed at a time, to understand its behavior and performance, for a few epochs: the selection of step size is covered in 4.3.3, number of tasks 4.3.4, and reward function in 4.3.5.

Furthermore, building upon the knowledge gained from those single-parameter experiments, we conducted a randomized search of parameters in an attempt to find an improved configuration, which is done in 4.3.6.

policy	exp. param.	alpha	epoch	LCS	CER	train loss	val loss
epsilon greedy	0.9	0.99	10	**0.717**	**0.302**	**0.466**	**2.170**
softmax	1.0	0.99	10	0.686	0.336	0.510	2.299

Table 4.7: Basic configuration performance. This was purposed based on our intuition and validated through early experiments to find a suitable combination of alpha and the exploration parameter (epsilon or τ, depending on the policy). The goal was to have an initial configuration to begin with our experiments.

At this point we had three main configurations: first, the basic we found in early experiments which was based in intuitions and the validation of the main hyperparams. Second, we have the configuration that arises from taking this basic configuration and applying all changes that showed some improvement with the alternative parameters. Note that it is not obvious that this second configuration performs better than the basic: when all changes are made together, they may interact leading to worst results since the effect of different changes could overlap or have contradictory effects. As a result, when multiple changes are applied at the same time, the performance could even drop. Finally, the third candidate is the one that was found with random search.

Last, those three candidate configurations are executed for a larger session of 20 epochs.

All experiments described up to the moment do not consider the effect of the Beam Search mechanism, since it was only executed for the final execution over the validation and test datasets to compare the best TSCL trained model with the traditionally trained one, which is done in 4.3.7, to explore the implications of TSCL on the results. Beam Search was not used until this point based on the observation that it is an heuristic which plays a role in prediction time only. Therefore, this process is not directly modified by TSCL, and should improve any model. On the top of this, beam search is more expensive to be run.

Finally some quantitative and error analysis is performed for the best model in 4.3.8 and 4.3.9, to share some insight on the effect of TSCL from a qualitative perspective.

4.3.1. Initial configuration

The initial configuration was defined based on intuitions which were also validated in short runs. The main goal is to find a good enough configuration on hyperparams that work as initial version to later improve them.

We set up the exploration parameter, epsilon for epsilon greedy policy, as 0.9 and temperature for softmax in 1.0. We choose 10 epochs as a base number of epochs because the training had not yet converged but its performance could serve as a good proxy for the model's future behavior.

The reward is based on the loss metric. We let the value of α defined as 0.99. This makes sense since the rewards change a lot from one step to the other, specially when the step size is large. Greater values of α imply that recent rewards are more important than older ones when updating Q function. 20 sub tasks are defined to be used with each step size defined as 2.5% of the training dataset. The Q values for each action are initialized at zero. For this basic configuration, without considering data augmentation usage, the metrics obtained are summarized in Table 4.7. It shows just that, a priori, epsilon greedy looks a little better than softmax, however this first configuration is not enough to prefer one over the other, and both policies will be considered in the next experiments until until we have further evidence to support the decision.

4.3.2. Adjusting the Q function

In this section the main goal is to understand the effects of the Q function and validate the usage of absolute value over it.

In the first experiment, the Teacher is replaced with a random policy that chooses actions uniformly random, and compared against a Teacher that follows an epsilon-greedy policy and another with softmax policy.

To follow a random policy looks similar to the base approach with traditional learning where all examples are sampled uniformly at random. However, there is a difference: in the base, the examples are sampled exactly once per epoch. In this random Teacher setting, an example can be sampled many times in a single epoch, or can not be sampled at all. This is because each step is randomly sampled with replacement, and independent from the previous step. Another point is that, since sub tasks are not splitted to exactly represent the percentiles of the sequence length, some of them, e.g. sub task zero, are sampled more often than with the traditional training set. This comparison is empirically shown in Figure 4.4.

Table 4.8 shows the exact map that exists between sub tasks and the se-

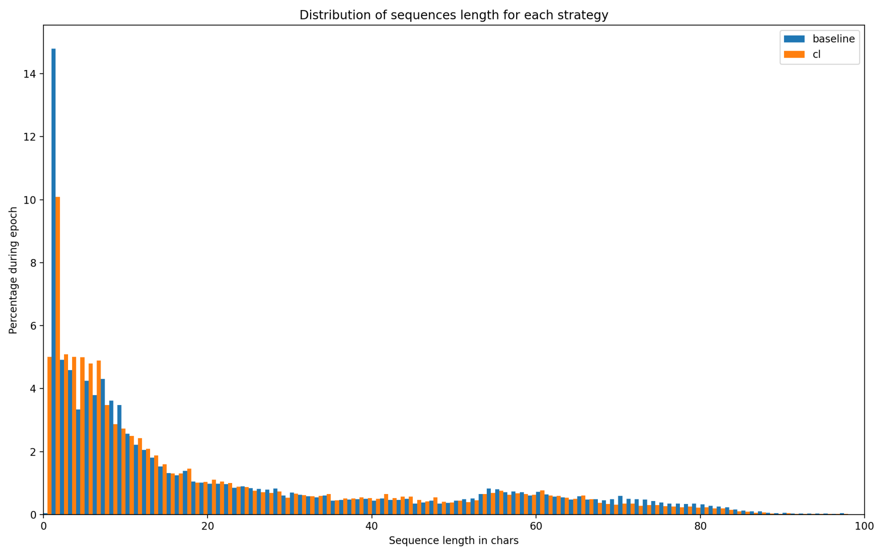


Figure 4.4: This image shows the distribution of sequence length during an epoch for two scenarios: the baseline, which is essentially the distribution over the full train set, and for CL using random Q. That is, at each step, examples from every sub task are sampled uniformly random. Main differences are at shorter sequences. For instance, sub task 0, which corresponds to examples without chars on it, represents a tiny percentage of the dataset, since there are just 30 of those examples in it, but when sampled according to the random agent, it grows several times: this is because sub task 0 is only composed by those 30 examples, which are over sampled to represent 1/20 of the full dataset. Other significant changes can be seen for sequences of length 1, but for the opposite reason: they are too frequent in the base, representing more than 14% of the cases, but constitute a single sub task, so they are undersampled to just 5%. This effect is better observed in shorter sequences because each sequence length constitutes a single sub tasks. As the sub tasks grows, there is more diversity with respect to sequence lengths among them and differences decreases.

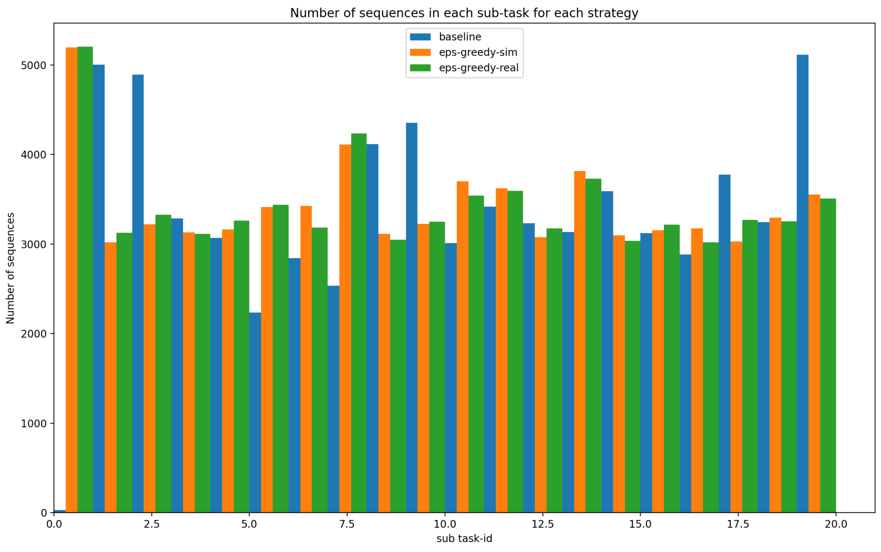


Figure 4.5: This is the sub task histogram for the sample built during the first epoch on different scenarios: baseline, which corresponds to the full training set, eps-greedy-sim that is a distribution obtained by simulating an epsilon greedy policy trained over actual rewards, and eps-greedy-real which corresponds to sampling on each step, according to epsilon greedy policy over the actual Q values from an execution.

sub task	length range	sub task	length range
0	[0 , 0]	10	[10 , 12]
1	[1 , 1]	11	[12 , 15]
2	[1 , 1]	12	[15 , 19]
3	[2 , 2]	13	[19 , 24]
4	[3 , 3]	14	[24 , 31]
5	[4 , 4]	15	[31 , 40]
6	[5 , 5]	16	[40 , 50]
7	[6 , 6]	17	[50 , 58]
8	[7 , 8]	18	[59 , 67]
9	[8 , 10]	19	[67 , 259]

Table 4.8: Table showing mapping between sub task and example length. Note that some sub tasks contains all examples with the same length, and there are also some lengths may be assigned to more than one sub task. This is done randomly: if there are two sub tasks that may be suitable for an example, then the assignment is performed randomly. This phenomena is explained by two facts: first, the length of a sequence is a natural value, second, the length distribution over the training dataset is highly skewed towards zero, so, when computing the percentiles, length 1 to 3 cover more than one case.

Policy	Epochs	CER	LCS	Loss
Eps-Greedy	32	**27.9**	**74.0**	**2.946**
Softmax	26	28.6	73.3	2.864
Random	24	29.8	72.1	2.703

Table 4.9: This table shows the results of running different policies for the Teacher at TSCL. The epoch is expressed as the epoch that provided the better CER, after 32 epochs. Random is for the Teacher agent choosing sub tasks uniformly at random.

Policy	**abs(Q)**	**epoch**	**LCS**	**CER**	**train loss**	**val loss**
Epsilon greedy	no	9	0.694	**0.323**	0.678	**2.104**
	yes	9	**0.697**	**0.323**	0.675	2.177
Softmax	no	10	**0.697**	0.324	**0.650**	2.278
	yes	10	0.672	0.347	0.664	2.251

Table 4.10: This table shows the difference between using or not absolute value for Q while picking the sub task to learn. It can be seen that for both policies, epsilon greedy and softmax, using the absolute value improves the results for all the metrics, but the differences are not that big.

quence length. Something similar happens when compared at sub task level, as can be seen in Figure 4.5.

The experiments shown in Table 4.9 shows that, the best of the Teacher for this setting, was to follow an epsilon-greedy policy, and, as expected, the Random policy is the worst idea. With respect to the epsilon-greedy and softmax policies, it must be said that, due to a bug in the code, softmax was not considering the absolute value of the function Q to choose the sub tasks, while epsilon-greedy did considered them. For epsilon greedy, epsilon was set to 0.9 and for softmax, the temperature is 1. Also the vocabulary used for this execution was the full vocabulary computed on the fly, without any other change on it. Image augmentation was not used for this experiment, and all the parameters except from the above mentioned, were kept fixed in both cases.

From this experiment we can conclude that the Teacher is actually learning something useful in this process.

Absolute value over Q function

This was a single experiment, carried on for both exploration policies, with the best parameters found for them, epsilon=0.9 and temperature = 1 respectively and using image augmentation for 10 epochs. The goal is to validate the

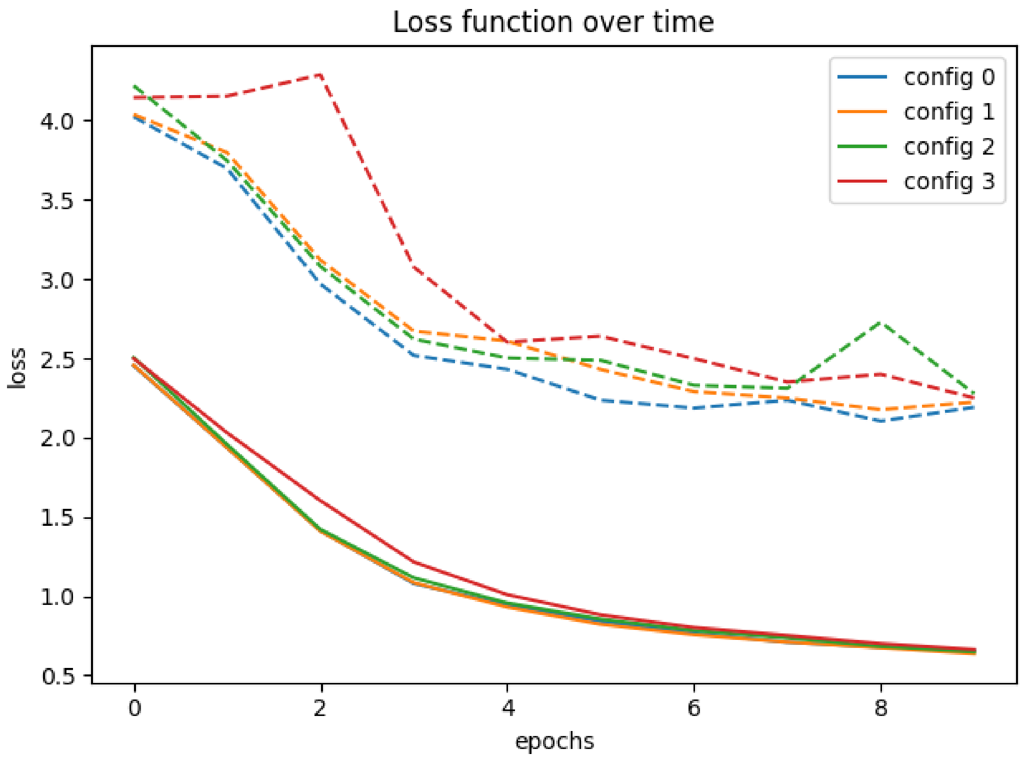


Figure 4.6: Evolution of the loss over training for the different configurations executed. config i legend corresponds to row i described in Table 4.10, solid lines are for trainig error while dashed ones are for validation.

observation made by Matiisen et al., 2019 with respect to the usage of absolute value over Q function to prevent catastrophic forgetting.

Large negative Q values implies that the model is decreasing the performance in the task, so, to revert this, the Teacher should also consider them to be included in the training data for the next step.

The main result is summarized in Table 4.10: for both policies, using Q's absolute value does improve the results significantly, for all metrics, but those differences are not as large as expected. What is more, the evolution of both scenarios shows similar behaviour over time, as can be seen in Figure 4.6.

This experiment concludes that using absolute value when selecting the action shows little improvement over training, but, as it is a cheap operation and there are theoretical reasons to use it, this will be used in the future, and final version of the TSCL setting.

policy	step size	epoch	LCS	CER	train loss	val loss
Epsilon greedy	2.5%	10	0.674	0.344	0.644	2.244
	5%	8	0.687	0.335	0.733	2.146
	10%	10	**0.706**	**0.314**	**0.639**	**2.126**
Softmax	2.5%	9	0.700	0.318	0.674	2.088
	5%	10	**0.727**	**0.291**	**0.635**	**2.018**
	10%	9	0.684	0.336	0.727	2.175

Table 4.11: This table shows the results for epsilon-greedy and softmax policies with different values of step size. It is interesting to note that the best value for this problem is different for each of the policies: for ep

4.3.3. Step size

This experiment aims to change the step size to see the effects that it has in the practice. The step size is defined as a portion of the training set which is chosen by the Teacher and given to the Student to do a gradient descent iteration on this data, obtain feedback and thus complete a training step in the Teacher's RL scheme. Most of the experiments were run with a step size of 2.5%, which implies that a step contains 2.5% of the training set to train, and 2.5% of the validation set to validate. This number was not cross validated, so may be modifying it, better results could be obtained.

The goal is to try larger values: 5% and 10 %. Larger values for step size means that, on each step, more updates are made to the model by the Student before checking again its performance, and giving it to the Teacher as feedback to let it pick up the next sub tasks, which may augment the variance between different steps. It also means that the validation carried on each step is with respect to a larger dataset, which implies to reduce the variance in this metric, that is used as a reward for the Teacher. A good balance between them should be found, and finding the optimal value is not obvious. Note also that, after an epoch of training, no matter how large the step size is, the Student updates its weights as many times as instances in the dataset, because this is the definition of an epoch, but not necesarily over the same instances, as discussed in 4.3.2. As a result, these experiments which were run for 10 epochs each, but different step size among them, produce a different number of steps each time, as can be seen in Figure 4.7. This figure shows the changes on Q function over time, illustrating the distinction between step size and epoch as units of training duration and their relationship: the former measures how often the Teacher is updated, while the latter corresponds to the extent of training for the Student.

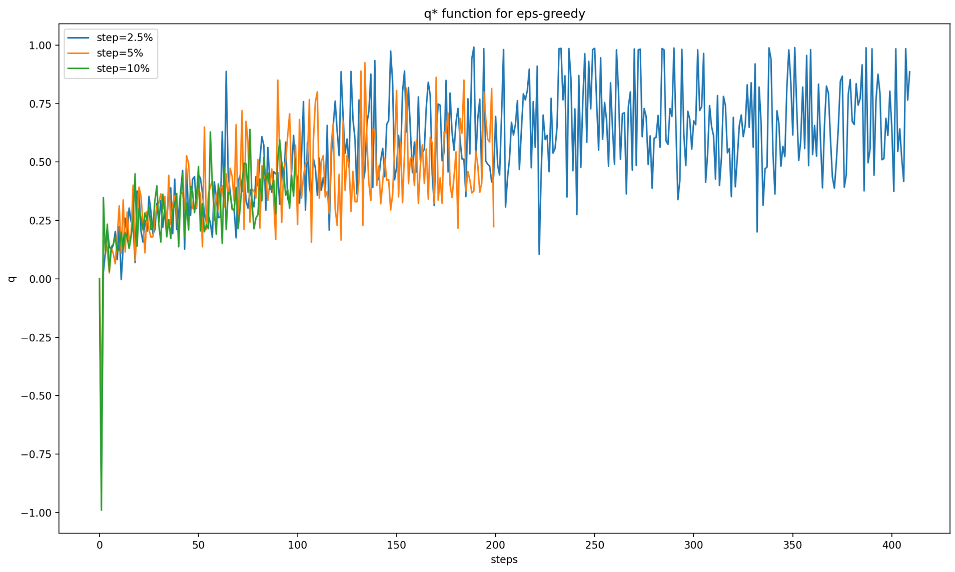


Figure 4.7: This plot shows the evolution of the Q function for the epsilon-greedy policy for different step sizes. Since the number of epochs is fixed for all of them, increasing the step size implies to reduce the number of steps per epoch. As can be seen, they behave similar to each other, but the function Q seems to stop improving only after 200 steps. This plot is only interesting to understand what is happening under the woods, but for the main task, the OCR, it is irrelevant.

Policy	N sub tasks	epoch	LCS	CER	train loss	val loss
Epsilon-greedy	20	10	0.697	0.320	0.630	2.074
	10	10	0.687	0.330	**0.607**	2.189
	5	8	**0.715**	**0.304**	0.780	**2.057**
Softmax	20	9	0.691	0.327	0.677	2.174
	10	10	0.703	0.316	**0.625**	2.065
	5	10	**0.729**	**0.288**	0.705	**2.039**

Table 4.12: This table shows the results for different number of sub tasks in both, epsilon greedy and softmax policies. For both of them the best CER is achieved with 5 sub tasks, and softmax policy is the best among them. It is interesting that, for the softmax policy, the performance seems to improve as the number of sub tasks decreases, but this is not true for epsilon greedy.

Table 4.11 presents the results. Surprisingly, the best step size depends on the policy being run, and in both cases the best is not the one being used: for epsilon greedy policy, an step size of 10% shows the best results, which is also globally the best among those experiments, and for softmax is 5%. Running for less steps may lead to a Teacher that do not converges. To check this, in Figure 4.7, the Q function is showed for each scenario, for the epsilon greedy policy. It is interesting to note that they behave similar for all step sizes, and they seem to converge only after 150 steps. However, talking about convergence is difficult because the optimal value is unknown and the reward function is completely non-stationary, so, nothing guarantees that, if the model is run for more steps, the function keeps its behavior.

4.3.4. Number of sub tasks

This section is similar to the previous, but instead of moving the step size, here the number of sub tasks is being changed.

In TSCL, according to the batch-wise formulation early described, all instances in the dataset must be assigned to a sub task, splitting the dataset (both, validation and training) into n-sub tasks. The number of sub tasks to be used is not obvious, and this section aims to find an appropriated value for it.

As the number of sub tasks decreases, each sub task contains more examples, and hence more variance. On the other hand, if the number of sub tasks grows, the number of possible actions for the Teacher also increases and makes it harder for it to learn good Q values for all of the actions.

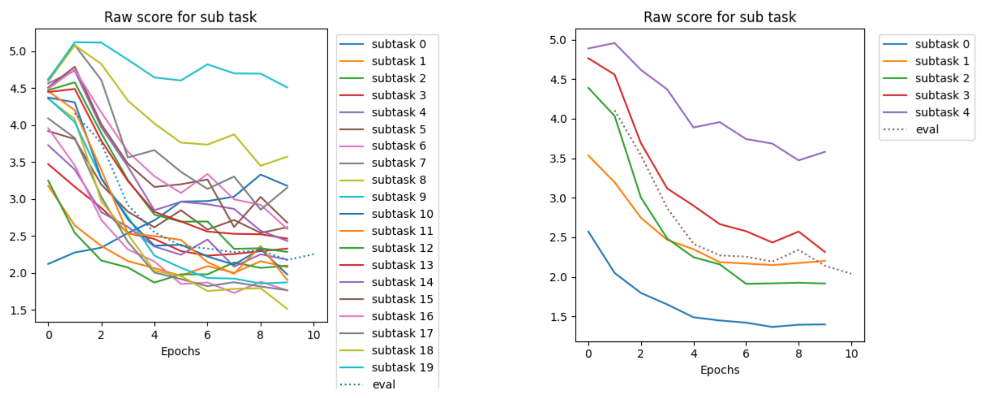


Figure 4.8: This image displays the temporal evolution of the loss for each sub task in different scenarios: 20 sub tasks on the left and 5 sub tasks on the right. Notably, when employing 20 sub tasks, sub task 0 exhibits a performance decrease over time, which is an unexpected behavior. This anomaly is unique to this scenario; all other sub tasks show performance improvement over time. This phenomenon is likely attributed to mislabeled examples within sub task 0, where some labelers have left transcriptions empty for unreadable instances. This is demonstrated in Figure 4.9. In contrast, when utilizing 5 sub tasks, the occurrences of instances with a length of 0 in sub task 0 are relatively fewer, mitigating the aforementioned phenomenon.

Results for this experiment are presented in Table 4.12. It can be seen that, for both policies, the best result is achieved with 5 sub tasks, and among them, softmax shows the best results.

Figure 4.8 shows the performance of the Student measured by loss for each subtask. This performance is measured after each step to compute the reward and update the Teacher. This differs from validation, which is measured after each epoch over the full validation set and predicts up to 105 characters. In the 20 subtask scenario, all subtasks show a similar trend of improvement over time, except for subtask 0, which worsens.

As shown in Table 4.8, subtask 0 corresponds to empty sequences, with only 30 in the training set and 4 in the validation set. We can analyze these cases individually, as illustrated in Figure 4.9. Two examples are mislabelled; they contain text that is hard to read. The labeller may have marked them as empty sequences instead of indicating they contain text. These two cases are in the training set.

Other examples are ambiguous; they include parts of seals, signatures, or ink stains, present in both training and validation sets. Such anomalies in labeling likely explain the decline in performance for subtask 0 in Figure 4.8. Validation is conducted with 2.5% of the validation set, using a uniform

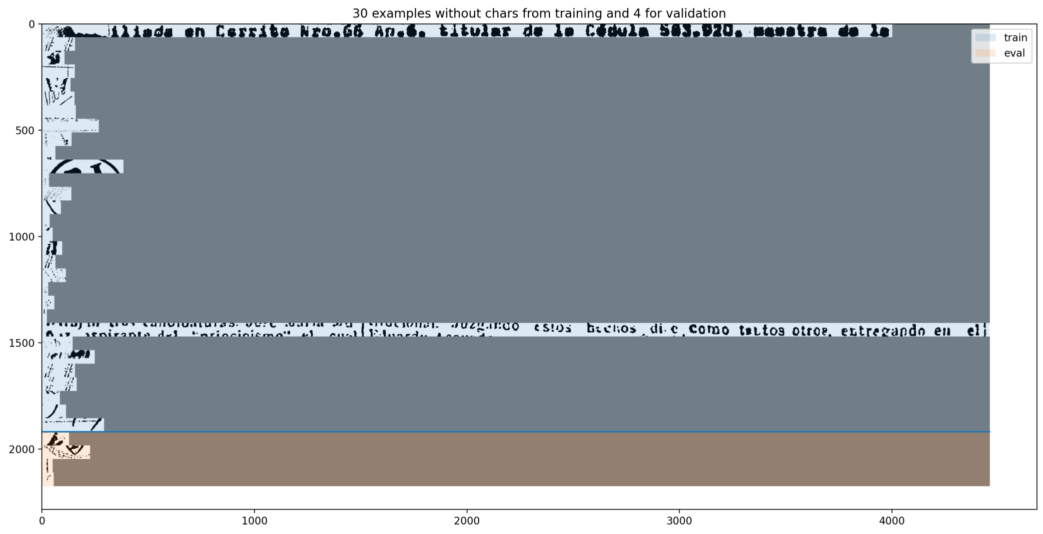


Figure 4.9: This figure shows the images whose sequence length is 0, on the top there are the 30 cases from training set and on the bottom, there are the 4 from the validation set. This shows that there are at least two training examples whose labels are wrong: they do contain some text that is impossible to read, but they are not empty sequences. Then there are other cases which are not that clear, because they look like parts of a seal or a signature.

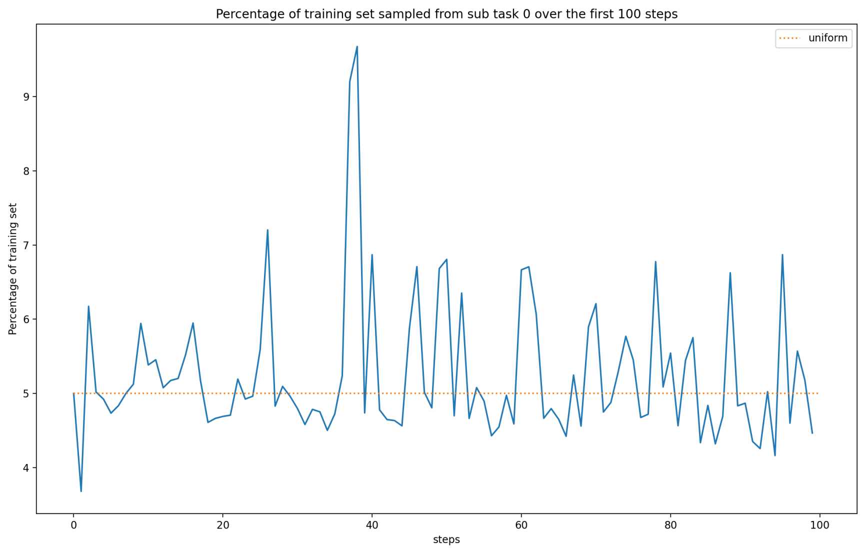


Figure 4.10: This figure shows how much of the training set does the sub task 0 represents, for the first 100 steps of training, a little more than 2 epochs. The dashed line represents a uniform distribution of tasks, which corresponds to 5% and is exactly the case for validation dataset.

Policy	reward	epoch	LCS	CER	train loss	val loss
Epsilon-greedy	CER	9	**0.679**	**0.341**	0.717	2.254
	Loss	10	0.674	0.344	**0.644**	**2.244**
Softmax	CER	9	0.694	0.325	0.688	2.168
	Loss	9	**0.700**	**0.318**	**0.674**	**2.088**

Table 4.13: This table compares different error metrics being used as reward for the Teacher, for both policies. Results are very similar, for the epsilon greedy policy, the CER reward works a little better, and in the softmax strategy, the loss is the better.

sampling method. To validate subtask 0, 12 examples are sampled from the validation set, often with repetition. This means the same four examples are likely being oversampled, contributing to the worsening metrics over time due to noise in the labels.

A similar issue occurs with the training dataset, but the sampling over subtasks is not uniform, as shown in Figure 4.10. This sampling changes over time as the Teacher learns how to impose a Curriculum for the Student. However, the task often represents more than 5% of the training dataset, leading to oversampling for subtask 0, where some labels are incorrect.

Now we understand why the performance in task 0 is poor in the 20 subtask scenario: we are oversampling mislabeled data. This results from a poor definition of the subtasks, leading to subtask 0 being both oversampled and noisy.

But why does this issue diminish when fewer subtasks are considered? As fewer subtasks are defined, each covers a broader range of sequence length cases. Since these are defined in terms of percentiles, they become more balanced than in the initial scenario. This balance helps dilute the anomalies, resulting in smoother training.

Noisy labels still exist, but as the subtasks are redefined, they are less likely to be oversampled. This change contributes to improved training results.

This change over samples may not be the unique explanation to understand why using less sub tasks results in a better performance: there may be another factors playing a role here, such as the action space, since for the Teacher or how confident and representative is the reward computed on each.

step_size	Temp	alpha	max_rew.	n_subtasks	epoch	LCS	CER	train loss	val loss
4.466%	1.12876	0.81104	1.61738	9	10	0.690	0.328	**0.594**	2.234
7.783%	1.06051	0.89982	1.40899	10	10	0.728	0.289	0.601	2.016
4.597%	0.94285	0.9873	1.58411	3	10	0.728	0.289	0.718	2.067
13.191%	1.08282	0.89274	0.54626	5	10	**0.741**	**0.276**	0.664	2.005
5.523%	0.8844	0.94947	2.83346	5	10	0.726	0.294	0.699	2.165
10.352%	0.90932	0.96281	2.50268	10	10	0.681	0.339	0.604	2.255
10.903%	0.94083	0.99616	0.79775	10	8	0.706	0.312	0.668	2.130
4.032%	1.05688	0.92645	2.28852	5	9	0.730	0.288	0.771	2.017
5.194%	1.02273	0.98046	1.98669	9	9	0.683	0.336	0.645	2.188
4.852%	0.90387	0.88053	2.94118	3	9	0.725	0.292	0.732	2.061
10.576%	0.9907	0.88723	1.47132	4	10	0.734	0.282	0.680	**1.994**
8.897%	1.06045	0.83342	1.18969	7	10	0.707	0.312	0.588	2.136

Table 4.14: This table shows the results from 12 random configurations. The best result in terms of CER was achieved at 10 epochs, which means that probably there is room for improvement with more epochs.

4.3.5. Reward: CER vs Loss

The reward in TSCL is defined as the difference in some metric before and after a step of training. Up to the moment, we only considered difference in loss but, CER is another possibility. The first is cheaper and unbounded while the second is in range [0,1] but it is more expensive.

In this section results are compared for rewards function computed based on both: the loss and CER. Table 4.13 shows the result of this experiment. It can be seen that the best case depends for each policy: for epsilon greedy, using the CER to compute the reward looks a little better, however, loss is better when the policy is softmax, and in fact, it is the best of those experiments. However, since differences look small, loss is preferred to compute rewards since all experiments and validations used it, and it is cheaper to compute compared to CER.

4.3.6. Configuring TSCL for Effective OCR Training

The following experiments present the final attempts to improve the TSCL performance, determine the optimal model configuration developed in this research and reports its results, performing a quantitative comparison with the base model, for the test set.

4.3.6.1. Random Search

Up to this point, all the parameters were validated by manually choosing values and training with them. This was done considering values that looks reasonable and that allows to draw some conclusions on how did they behave, since only one parameter at a time was changed. In this section instead, we select random values, considering ranges based on previous experiments. The goal is to perform a small random walk over the hyperparameter space, to potentially identify some configurations that may perform better than what we have up to the moment.

The hyper parameters to be randomly sampled are the following:

- n_subtasks: number of sub tasks to use, an integer in range [3, 10]
- step_size: portion of the training size that defines the size of a single step, a float in range (0.025, 0.15).
- temperature: parameter used to balance the exploration-exploitation compromise for the softmax policy, a float in range (0.85, 1.15).
- alpha: a random number generated from a uniform distribution between 0.8 and 1. This is used to weight previous experience while updating Teacher's Q function.
- max_reward: largest allowed reward. Rewards in range [0, max_reward] are linearly scaled to [-1, 1]. Rewards larger than max_reward are clipped. It is a float sampled from a uniform distribution in range (0.5, 3).

All decimals were rounded to 5 decimal places. 12 combinations of those parameters randomly sampled were chosen and executed for 10 epochs each, all of them using softmax as the policy: based on previous experiments, it looks like softmax policy performs better than epsilon-greedy, which is logical since it allows to express more complex distributions. Results are provided in Table 4.14. The best configuration is the one that minimizes the CER. In the next section, this best combination will be run for larger epochs to compare against other strong candidates and finally define our best model proposal.

4.3.6.2. Best Candidates: longer training

In this section we take some good candidates from previous experiments and let them train for more epochs: 15 instead of 10. There are four candidates: the default model used in all previous sections for TSCL, for both policies, epsilon greedy (A) and softmax (B), the configuration that comes up by applying all the improvements that were reported in previous experiments (C): Note that this combination was never used before, and may not

Policy	Candidate	step_size	n_subtasks	epoch	LCS	CER	train loss	val loss
Epsilon greedy	A	2.5%	20	15	0.729	0.289	**0.523**	2.107
Softmax	B	2.5%	20	14	0.710	0.310	0.537	2.223
	C	5%	5	15	0.744	0.274	0.596	**2.063**
	D	13.191%	5	15	**0.754**	**0.263**	0.565	2.079

Table 4.15: Candidates A and B corresponds to the base TSCL formulation, candidate C contains the combination of best parameters discussed during this section, and last but not least, candidate D is the best configuration found by sampling random configurations, as extracted from Table 4.14. It is interesting to see that, for the base TSCL implementation, epsilon-greedy outperforms Softmax by some points in most of the metrics, despite our previous observation that softmax policy seemed better in short runs. However, the best results appears when two changes are applied together to the base formulation. Finally, the best results where achieved by candidate D which was a random walk over the hyperparameter space. Note that candidate D modifies other parameters such as max_reward, that are not included in this table for clarity. Other three candidates share all parameters that are not shown in the table.

improve the results, since parameters were validated to improve the results when changed alone, but when used together they could not improve them. And finally, the best configuration found through the random walk (D). None of those includes the usage of beam search.

Results are summarized in Table 4.15. The best is achieved with candidate D. This configuration results from a random walk defined by the knowledge gained from previous experiments: for instance, as discussed in Section 4.3.2, using the absolute value of Q was fixed, the reward was computed from the loss instead of CER as proposed in Section 4.3.5, and only the softmax policy was validated. Therefore, this is selected as the best candidate to be evaluated over test set and compared with the strongest base model both, quantitative and qualitative. This will be done in the next section.

4.3.7. Evaluation

This section briefly presents and compares the results for both approaches, the base and TSCL, over the test set. Both use beam search while decoding, with the parameters found for the base model. Results are exposed in Table 4.16. It turns out that TSCL shows best results for all metrics but loss, in both, validation and testing set. In both cases there are improvements when soft metrics is considered, that is, case insensitive versions of the LCS and CER. These findings align with what was previously reported in Chavat's work.

At this point, we can conclude that TSCL reports a small improvement on

Split	Approach	LCS	CER	soft LCS	soft CER
validation	Base (Chavat)	0.74	0.282	-	-
	Strengthen Base	0.781	0.236	**0.790**	0.228
	TSCL	**0.783**	**0.233**	0.792	**0.224**
test	Base (Chavat)	0.74	0.282	-	0.277
	Strengthen Base	0.781	0.236	**0.790**	0.228
	TSCL	**0.782**	**0.234**	**0.790**	**0.226**
	Calamari-OCR (*)	0.75	0.237	-	0.232

Table 4.16: Evaluation over both: validation and test set. Note that the validation set is exactly the same as the one used until now, but this table includes some more metrics: soft CER and soft LCS. These are case-insensitive variations of the CER and LCS. They are reported since Chavat Pérez, 2022 reports that several examples do not respect cases in the dataset. Both, the Strengthen Base and TSCL models do incorporate beam search. This table also includes Chavat's reports for comparison over the same dataset. We can conclude from this table that, TSCL surpasses the traditional approach, although the margin is small. The large differences with respect to previous work on this dataset are due to improvements over the Base model, which are verified in both: validation and test sets. Calamari-OCR, taken from Chavat's work, corresponds to an OCR tool called Calamari (Wick et al., 2020) that was trained on a curated and smaller version of the dataset used in this work. This is not strictly comparable but illustrates the potential of our work.

the target metric with respect to traditional alternative, but only for a small margin. However, our models show a significant improvement in performance compared to the results reported by Chavat. In fact, our Strengthen base results are even better than the ones reported for Calamari Wick et al., 2020. This is an open source OCR model that is currently being used in LUISA. Those metrics are not strictly comparable, since Calamari was trained and evaluated over a smaller and curated version of our dataset, which was not available for us. Nevertheless, these findings highlight the need for additional experiments to evaluate the performance of our Strengthen base model on the same dataset used to fit and evaluate the Calamari instance.

In the reminder of this chapter, we will first dig into our best model trained with TSCL to understand what is under the woods in the next section, and finally, in the last section we will perform an error analysis for both approaches, trying to compare and understand them.

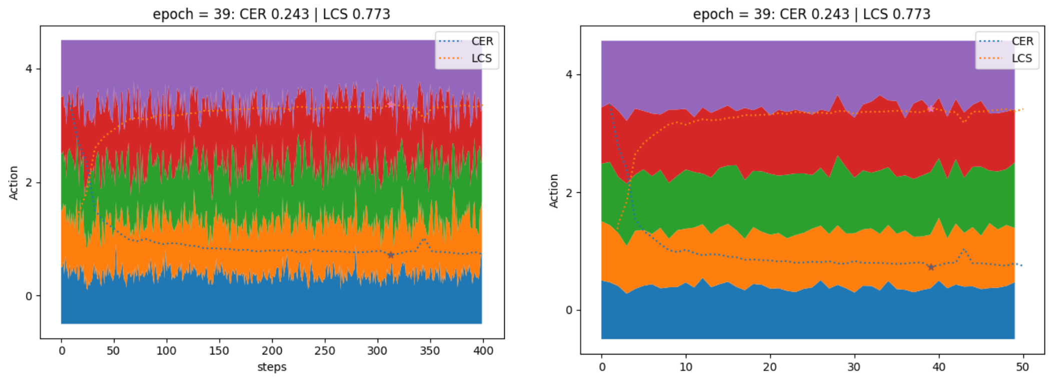


Figure 4.11: This image shows the evolution of the sampling over the time for the TSCL model. On the left, the image shows the sampling distribution for each step. On the right there is exactly the same data, but averaged over each epoch. Naturally, the first is noiser than the second, and both are really similar to a uniform distribution.

4.3.8. An analysis of the TSCL training process

This section aims to explore notable aspects of the TSCL training process, now that we have identified the best configuration and trained it for 50 epochs.

The evolution of sampling for each subtask over time is depicted in Figure 4.11. Unlike the patterns observed in Figure 4.12 from Matiisen et al., 2019, where the Teacher begins by sampling shorter sequences more frequently before progressing to longer ones, our results show no distinct trends. In fact, distributions are close to uniform.

The next point was to understand how different those distributions are from the uniform. To accomplish this, we utilize the Total Variation distance (TVd)[1] which measures the distance between two probability functions p and q. In our case, we compute the distance between uniform distribution and the distribution computed by the Teacher model at each step, which is shown in Figure 4.13. As can be seen, the TVd starts at 0 because the Teacher initialize Q values as zeros, which implies a uniform distribution over examples. Then it grows for some steps and finally it starts oscillating without a clear tendency. When considering all steps after the first two epochs, the correlation between TVd and the step is only 0.02. This explains why it does not exhibit a clear tendency.

[1]TVd is defined as: $TVd(p,q) = \sup_A |p(A) - q(A)|$ where the supreme is computed over all the events A. In particular, when the probability space is finite as in this case, the TVd distance can be computed like this: $TVd(p,q) = \frac{1}{2}\sum_{i=1}^{n} |p(x_i) - q(x_i)|$.

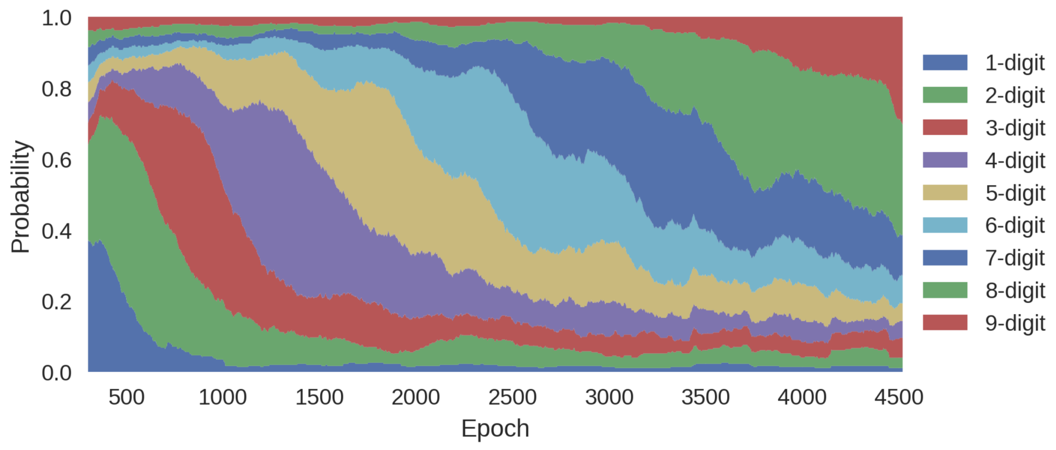


Figure 4.12: This image belongs to Matiisen et al., 2019. It show how the distribution over tasks changes as training proceeds, for 9-digit addition task. The algorithm progresses from simpler tasks with few digits to more complicated. Harder tasks take longer to learn and the algorithm keeps training on easier tasks to counter catastrophic forgetting.

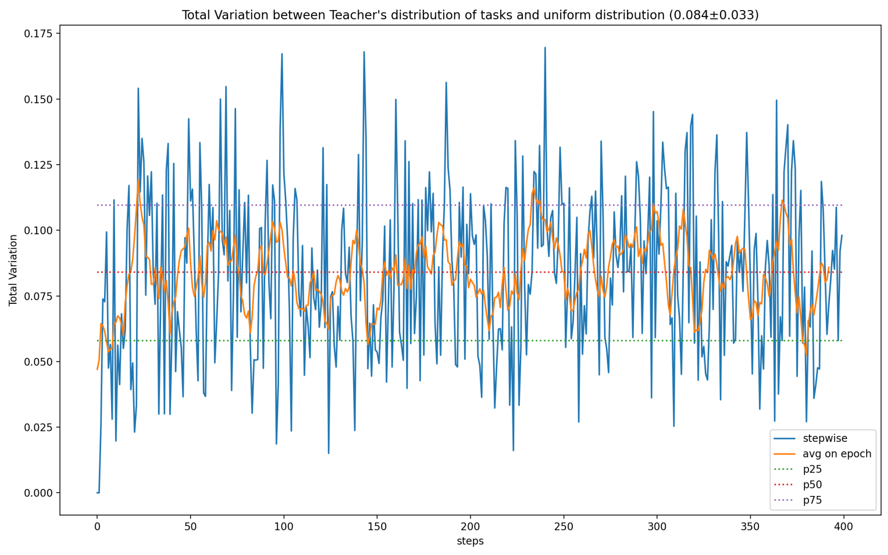


Figure 4.13: Evolution of Total Variation distance between the empirical distributions defined by the Teacher at each step, and the uniform distribution. This distance goes from 0, when both distributions are the same, to 1 one they are completely different

policy	**epoch**	**LCS**	**CER**	**train loss**	**eval loss**
Random	10	0.734	0.283	0.695	2.016
Softmax	10	**0.741**	**0.276**	**0.664**	**2.005**

Table 4.17: This table shows the results of running different policies for the Teacher in TSCL. Random refers to the Teacher agent choosing sub tasks uniformly at random

The TVd is 0.084 on average among all steps. Considering that this distance ranges between 0 and 1, this difference is really small. It means that, on average, the maximum difference in the probability of being sampled for any sub task is 0.084. Therefore, the learned function is similar to the uniform distribution, but not identical. In fact, in the early experiments presented in Section 4.3.2, we showed that replacing the learned Q function with a fixed uniform distribution resulted in worst performance. Therefore, we concluded that the Q function is actually playing a role here. However, this experiment was conducted for a specific case, with the initial configuration that we purposed, which we later confirmed was not the optimal configuration. In this case, we compare the TVd for our best model. There are two major differences among those two configurations: the number of sub tasks changed from 20 to 5 and the step size from 2.5% to 13.2%. Those changes certainly affect the dynamics of the training, hence, we run the same uniform random policy for the Teacher but with the new configuration. As can be seen in Table 4.17, our previous conclusion still holds true: following the learned Q function to sample from the training set is better than sampling sub task uniformly random.

Now that we have confirmed that Q function is actually playing a role, the next section tries to understand the model's behaviour by analyzing its errors and comparing them between the TSCL model and the traditionally trained one, to identify any further differences.

4.3.9. Error analysis

In the previous section, Table 4.16 showed that TSCL performed better in terms of CER than the traditional method. In this section, we will start by looking closer to this difference. We compare our strengthen base model with the TSCL model, considering their errors. Most of the analysis is based in the CER metric since it is our main metric along this work, and in fact this is highly correlated with LCS, so several times the same effect is observed in

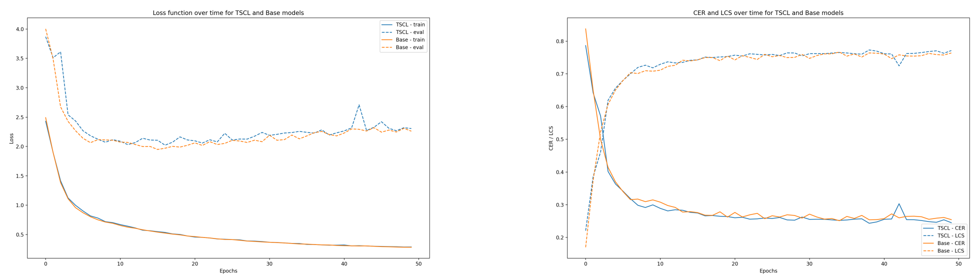


Figure 4.14: This image shows the evolution of loss, CER and LCS metrics over time. The image on the left is for loss. The solid lines correspond to training set while dashed lines are for validation set. The picture on the right is for CER and LCS metrics, both over validation set. As can be seen, both models behaves similar for all metrics, and may be the unique difference is that Base model's metrics tend to be smoother than TSCL. This makes sense since the training set is constant during each epoch, while training set is sampled following a dynamic function defined by the Teacher for TSCL. For those metrics, both models behave similar, which invalidates the hypothesis that TSCL may learn faster or require less epochs to achieve same results than traditional training.

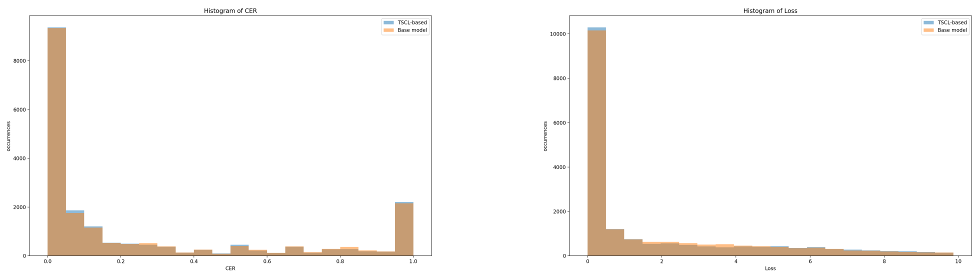


Figure 4.15: This image shows the histogram on CER (left) and loss (right) for each example in the test set, for both models. There are small differences on each, but they are really similar in both metrics for each of the models. For visualization purposes, loss values are kept up to percentile 95 of the base model distribution, which is larger than the percentile 95 of TSCL model. This allows to use the same bins for both histograms.

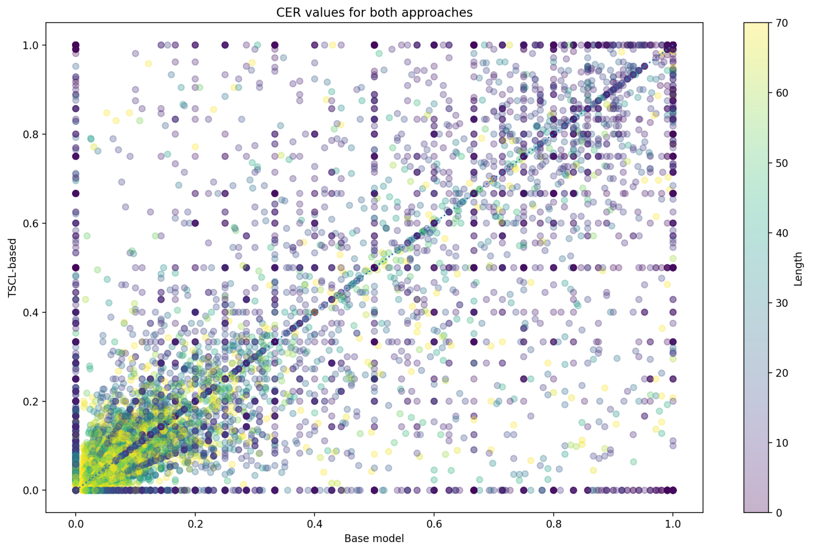


Figure 4.16: This image shows the CER for each example in the test set, for both models. Each dot in the scatter is colored according to the number of characters in the instance, which is clipped to 70 characters to improve the visualization: those instances with more than 70 characters are colored as containing 70. If both models return the same for every single input, this scatter should be just the identity. However, in this case we can see that there are several examples with large CER values in TSCL model, which have a small CER value in the base model.

both metrics. In some specific cases we will include the loss function in our analysis, specially when its behavior differs from the CER. For this analysis, beam search is not considered.

Let's start by comparing the evolution of errors while training. Figure 4.14 shows that during the training session, both models behave similar. The only difference that can be seen is that for TSCL model the metrics are not that smooth, which may be explained by the changes over time in the sampling function that the Teacher is computing. This similarity in terms of the evolution of different metrics over time, implies that the hypothesis of TSCL requiring less time or data to achieve same results than traditional training approach is not supported by the data. In fact, the validation loss for the traditional training approach appears to decrease slightly faster than that of the TSCL model.

Now that we see no big differences in the behavior of the error while training, let's see the difference in the models once they are trained. Consider for instance the histogram of the CER values for the full test set, which is shown in Figure 4.15. Both histograms are really similar, with small differences in particular bins only, and with really tiny differences. It is also interesting to note that the histogram looks skewed to the left, with less examples in each bean as the CER increases, but then, the last bin which corresponds to a CER in $(0.95, 1.0]$, it grows again. A possible explanation for this may be misslabelled examples, but more evidence is needed. However, this kind of bimodal distribution is not observed in the loss.

The next thing to do, in order to compare and analyze those models, is to compare the CER for every single instance in the test set. For instance, Figure 4.16 shows a scatter between CER values for both models. If those models make the same predictions in all the dataset, this scatter should be accumulated over the top-right bottom-left diagonal. Those points which are far from this diagonal, are the ones whose prediction changes the most from one model to the other. In some special cases, like instances with a single character, this is easy to understand: if one model predicts it perfectly and the other predicts a different character, this example would have 1 and 0 for CER in each case, which changes a lot. However, as can be seen in the figure, there are some cases where the sequence length is larger, and therefore those cases are more interesting.

To investigate these cases, we manually examine instances where the CER

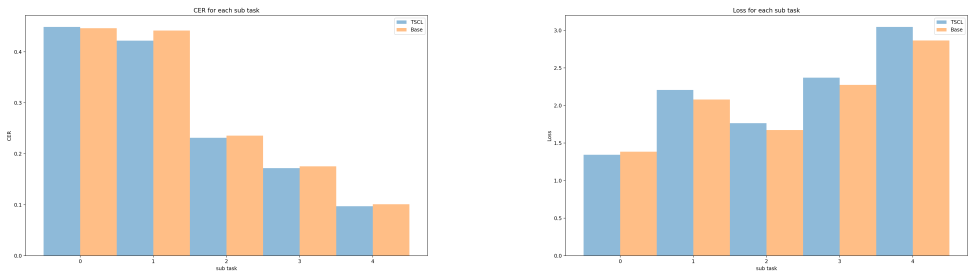


Figure 4.17: This image shows the CER (left) and loss (right) metrics for each sub task in both models. It is interesting to notice that, for loss, as the sub task increases and the sequences are larger, the error is increased, while for CER it is the other way: larger sequences lead to lower CER error. In terms of CER, for most sub tasks the TSCL model is slightly better, and this difference is larger in the sub task 1. In terms of loss is the base model the one that performs better, something that is coherent with the metrics exposed in Table 4.16.

is high in one model but low in the other. We filter cases where the CER falls between 0.80 and 1.0 for the TSCL approach, and between 0 and 0.20 for the base model. This filtering removes examples that are completely wrong or perfectly transcribed. However, there was not a clear pattern that explains those differences. Therefore, before proceeding with further experiments and hypotheses, we wanted to know if this difference in the scatter was actually because of TSCL or if it is something normal when comparing two different model instances. To answer this question, we plot exactly this, but, for the results obtained with the base method, and the epoch inmediate posterior to it. We choose this since an epoch looks like a small change from one instance to the other, and ensures that the training conditions are the same in both cases. We choose the epoch after the best CER was computed and not the epoch before it because, even when both models are just one epoch away, the first was more similar in terms of its CER. The resulting scatter plot showed a similar pattern, with some examples exhibiting significant changes between model instances, despite sharing most of the training. This suggests that the observed differences may not be particularly relevant in the context of understanding TSCL or, at the very least, may not provide significant insights, therefore we abandon it.

Another way to identify differences is by examining the CER for each sub task. This perspective is valuable because sub tasks are correlated with sequence length, and therefore difficulty. By examining the errors for different

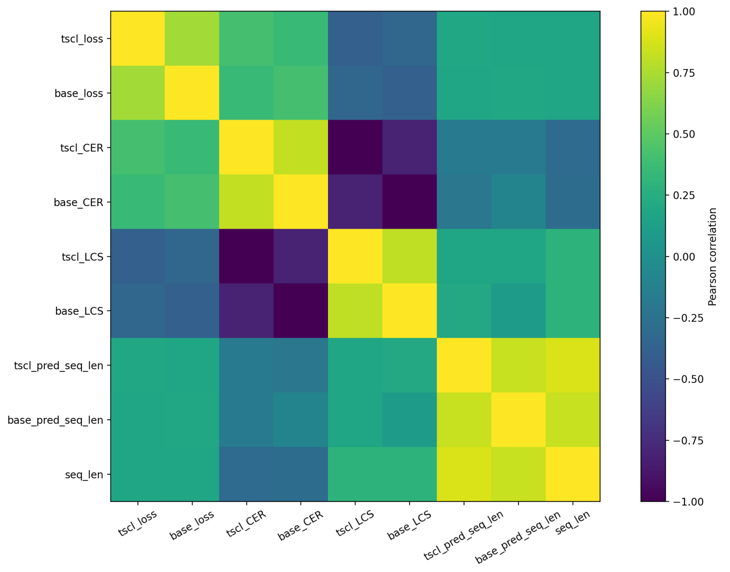


Figure 4.18: This image shows graphically the correlation between pairs of metrics or sequence length. The values can be seen in Table 4.18. Each of the pairs *_CER, *_loss and *_LCS represents those metrics for TSCL and base model, depending on its prefix. Something similar is done for sequence length but in this case it also includes the ground truth sequence length. As can be seen, each of those metrics behave really similar with a high correlation among them and similar correlation between the sub set and any other subset. It worth noting that LCS and CER have a large negative correlation among them which is related to the fact that both metrics are based in the Levenshtein distance.

sequence lengths in each model, we can determine if there are any consistent patterns. This can be seen in Figure 4.17. However, it appears counterintuitive that as the sub task number increases, the sequence length also grows, yet the CER decreases. Larger sequences are supposed to be harder for the model and hence lead to greater error, but according to this plot, the larger the sequence, the smaller the CER error. In terms of loss, the behaviour is as expected. This discrepancy may be attributed to variations in the errors captured by these different metrics and the specific aspects they penalize. This should be considered when designing a CL strategy: not all the error functions behave the same, and in some cases we may be adding biases or features from a particular reward function to our curriculum design.

Finally, we can consider the correlation between the metrics for both models, which is shown graphically in Figure 4.18 while correlation values are in Table 4.18. We chose to do so because this considers the metric obtained at

Pearson correlation	tscl_loss	base_loss	tscl_CER	base_CER	tscl_LCS	base_LCS	tscl_pred_seq_len	base_pred_seq_len	seq_len
tscl_loss	1	0.725	0.411	0.354	-0.388	-0.336	0.191	0.184	0.18
base_loss	0.725	1	0.356	0.411	-0.337	-0.388	0.186	0.188	0.173
tscl_CER	0.411	0.356	1	0.813	-0.994	-0.81	-0.175	-0.176	-0.297
base_CER	0.354	0.411	0.813	1	-0.806	-0.994	-0.205	-0.096	-0.297
tscl_LCS	-0.388	-0.337	-0.994	-0.806	1	0.808	0.177	0.179	0.297
base_LCS	-0.336	-0.388	-0.81	-0.994	0.808	1	0.206	0.098	0.296
tscl_pred_seq_len	0.191	0.186	-0.175	-0.205	0.177	0.206	1	0.831	0.89
base_pred_seq_len	0.184	0.188	-0.176	-0.096	0.179	0.098	0.831	1	0.832
seq_len	0.18	0.173	-0.297	-0.297	0.297	0.296	0.89	0.832	1

Table 4.18: This table shows the Pearson correlation between metrics and the sequence length, for each model. This is also shown graphically in Figure 4.18.

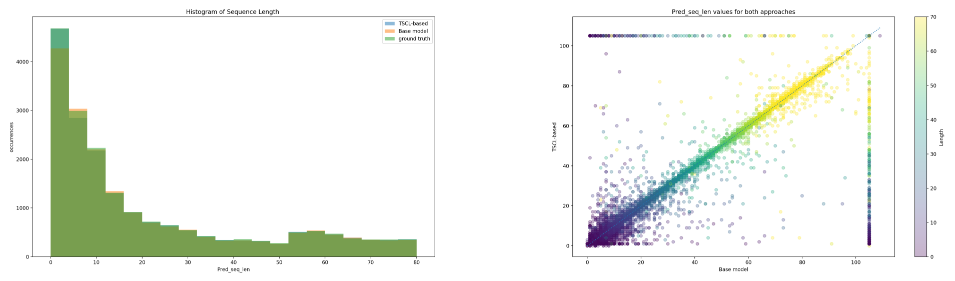


Figure 4.19: On the left, there is a histogram of sequence length for ground truth, TSCL and base models predictions. On the right, there is a scatter which shows how the length of each prediction is related for both models, while the color represents the target sequence length. Points in aligned in the top and right edge are near 105, and is related with the evaluation protocol defined: predict up to 105 tokens top. This happens in both models: examples whose predicted length is far more than the ground truth, and can be seen with the color: dark pints are shorter in ground truth sequence than 70 characters, so those examples are the ones where the model is stocked repeating far more characters than needed. In fact, the test set contains only 18 examples with 105 characters or more, however, the TSCL model generates 350 examples with 105 characters, and the base model 559. Finally, consider that predicting smaller sequences does not imply that they are better.

instance level, instead of comparing averages. It can be seen that, for a given metric, the correlation between two models is really high among them, which seems to validate that both models perform similar also at individual instances.

We can also see that LCS and CER metrics within the same model are really correlated among them: -0.994 for both models. This is related to the way those metrics are defined: both are based in the edit distance between predicted and target sequences. This also supports our previous claim that observations made based on CER are often valid in LCS also.

A small difference can be seen with respect to the sequence length: TSCL sequences' length are better correlated with target sequences length than the base model: 0.89 vs 0.832. This may suggest that TSCL tends to generate sequences of the correct length more accurately than the base model. To validate this hypothesis, Figure 4.19 shows the histogram of sequence length for target sequences and predicted sequences under each model. It can be seen that they are really similar, but, for the first bean, TSCL shows a larger difference with respect to the base model, and in fact it is closer to the ground truth, which goes in the same direction: TSCL model seems to arrive to a model that is better at finding the exact length for the instances, specially for the shorter. However, again, we are only considering a single instance trained with TSCL and a single instance trained with traditional methods, so it may be possible that this difference is just because they are different model instances and not related to the way they were trained.

From this section we can conclude that we found no relevant differences among the models. Also we see that LCS and CER metrics are really close related, so it should be the same to pick up one or another. TSCL model seems to be better at predicting the proper length and shorter sequences. However, this can only be ensured for this particular instance being validated: further systematic work is needed to understand if this difference is due to the changes on how the model was trained (i.e. using TSCL) or if it was just a random phenomena for two models that are almost the same. In terms of CER, the model tends to make less mistakes, but this may also be a consequence or bias of the CER definition itself more than a characteristic of the model.

Chapter 5

Conclusions and Further Work

In this thesis we explored CL, in particular, TSCL and applied it to an OCR problem. In addition to this, we proposed some improvements to Chavat's work than can be applied even without the TSCL framework: the main items are the usage of Image Augmentation and Beam Search. Those changes suggest that seq2seq models, like the one that Chavat proposed, may be able to beat more complex ones like the one purposed by the Calamari OCR tool.

In addition to this, we successfully adapted the TSCL framework for a seq2seq problem, documenting the whole process. This leads to a better understanding of what is and how CL and in particular, TSCL works.

In the reminder of this chapter, we share a summary of this thesis, its main contributions, some possible next steps, based on this work, and our conclusions. But, before closing this work, the next section share some insights on why does TSCL works.

5.1. Why does TSCL work?

In section 2.5 we shared this question with respect to CL. However, in this work we focused on TSCL, framed for supervised tasks. This answer is not addressed in the literature, as far as we know. In this section we share our thoughts about this point.

First of all, this question assumes that the method works. But, what does it mean that the method works? We will not answer this question, but we can tell what we know up to the moment: as shown throughout this work, we were able to run TSCL in one particular setting, for one particular problem, obtaining results comparable to those obtained with traditional training. TSCL's implementation depends on its specification for each problem and requires defining several items, such as the sub tasks or the reward. Such definitions and implementations, as we previously discussed, may lead to different results. In addition to this, the bibliography purposing and testing this framework is scarce. So, we can not draw general conclusions about it from this work.

That said, we can note that answering why and how it works, depends on what is the dataset being used and how the sub tasks are defined. Since there is no prior definition on the pace function or the dynamics for the selection of the sub tasks to train, TSCL will be able to consider only indirectly the criteria used to create the sub tasks. For instance, if the dataset is noisy, it may or may not try to tackle the mislabelled instances problem, depending if the definition of sub tasks consider somehow the level of noise in its examples. If noisy labels are distributed among all sub tasks, it seems hard that the Teacher's decision could consider them when choosing the examples used to train. In Guo et al., 2018 for instance, despite of not implementing TSCL, the authors split the dataset into three sub tasks using clustering strategies, so that each cluster contains examples with the same label quality within it. This could be considered into the sub task definition to allow the Teacher to choose learning paths that may consider them, in a harder to easy, easy to harder, or whatever it finds better to apply them.

The main change behind scenes introduced by TSCL is that it allows to dynamically oversample or undersample the input space as needed by just choosing whatever makes the largest changes over the reward function, and this looks like the key in how TSCL works. Also as validated in experiments, the usage of absolute value for Q function plays a role to mitigate catastrophic forgetting. In addition to this, it is still true the observation that, on early epochs the model is more sensitive to learn.

5.2. Summary

This work focused on exploring TSCL and its application to an OCR task. We compared the performance of TSCL with the traditional training regime in this specific application. We first conducted a theoretical analysis, showing the underlying principles behind the effectiveness of CL in general, and TSCL in particular. Moreover, it juxtaposes the theoretical analysis with experimental results that, despite not demonstrating significant improvements in the task, provide valuable evidence contributing to the understanding of the functioning of the TSCL framework. It is worth noting that a detailed study regarding the decision-making process for framing a supervised problem within TSCL is lacking in the existing literature. As a result of this, we share some insights into the reasons why TSCL works.

5.3. Contributions

A preliminar but important aspect of this work is the strengthening proposed over Chavat's work. While this was not an objective itself, we were able to improve his proposal by adding beam search to the inference, training using image augmentation over more epochs, surpassing the metrics reported by Chavat in more than 16% for CER. In fact, the results obtained in our work suggest that training with a cleaner version of the dataset could reach better results than those obtained by Calamari, the OCR tool currently used by the LUISA team. This work also contributes to the specific OCR task by proposing novel augmentation functions that resulted in the most significant improvements compared to previous models for this particular dataset. However, the related parameters of these functions were not empirically validated to their optimal values, suggesting further room for improvement in this direction.

While this work does not provide an extensive exploration of TSCL applied to different problems, it represents a step forward in understanding the decision-making process for framing supervised problems within the TSCL framework. This aspect constitutes a contribution to future implementations of the framework in supervised tasks, extending beyond OCR. To the best of our knowledge, this is the first attempt to implement the TSCL framework in an OCR task.

All the code produced in this work is publicly available[1]; however, it is important to note that the code should be considered experimental and may require further refinement and validation for production-level usage. For privacy reasons, the dataset used in this work is not released.

[1]https://gitlab.fing.edu.uy/rodrigo.laguna/pro-pos-grad-luisa-cl

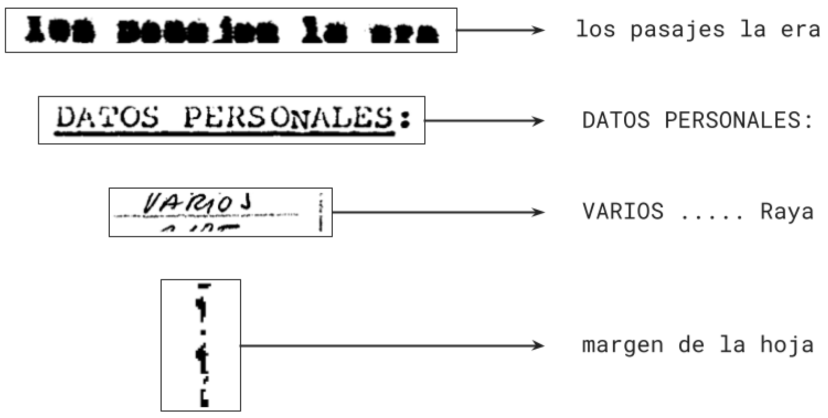


Figure 5.1: Examples from the validation dataset, each shown with its corresponding label. The first image contains 18 characters, while the others contain 17. Therefore, according to the criteria defined in this work, all belong to the same sub task and are expected to share a similar level of difficulty. However, the second example is clearly more legible—at least for humans—than the first. The third image is visually clear but handwritten, and its label contains two errors: the sequence of dots should not be present based on the labeling criteria established in Luisa, and "Raya" refers to the underlining rather than the text content. Finally, the fourth example is completely mislabeled, as the black vertical line corresponds merely to the page margin. Such variations in difficulty should be considered when defining sub tasks. An additional pre-filtering step might help exclude instances like the last one from the dataset.

5.4. Future work

One key point in TSCL framed as for supervised tasks, is the sub-tasks definition. In this work we follow a simple approach after trying some alternatives. However, none of them addressed a key point which later in the error analysis showed up: noise in the labels. The dataset used does not provide any information about it. We could re-define the sub tasks in this point, so that the teacher could consider, at least indirectly, noise in the data when choosing the tasks to train, as showed in Figure 5.1. The effect of noise in the labels is not particularly studied in the literature, and how to capitalize it in TSCL frame work may be a good direction for further research. In addition to this, the reward function may also play an important, and sub-explored, role. The goal of our system is to minimize CER metric, which is non-differenciable, so we fit the model using SGD to minimimize an alternative loss. Then, we choose our TSCL to guide its behaviour based on changes over the loss function. So, the relation between the metric being optimized by SGD and the one being jointly optimized by the Teacher and Student seems another interesting point to reconsider.

One limitation of this work, and perhaps its most significant weakness, is that all experiments and conclusions were based on a single training session as discussed in 4.1. This implies that the reported metrics and comparisons lack robustness and fail to account for potential variations across multiple training runs. Conducting experiments with repeated runs and analyzing the performance variability would provide more robust and reliable insights. Additionally, the improvements in terms of CER and LCS metrics were relatively small, and the error analysis did not show major differences among them. Although TSCL outperformed traditional training in these experiments, it remains uncertain whether TSCL would consistently yield the best results if the experiments were repeated multiple times. This work does not address this question.

The most practical next step to follow is to apply the improvements proposed on the base model, that is, Image Augmentation, changes in the charset and using Beam Search, with comparable settings and dataset to seriously challenge and probably beat the model that is being used in LUISA: Calamari, as discussed in Section 4.3.7.

5.5. Conclusions

We successfully implemented TSCL for a supervised problem, specifically OCR. However, the results obtained with this technique, when compared to traditionally trained methods, show no significant improvement in terms of performance. Moreover, a qualitative error analysis did not reveal any substantial differences.

Additionally, TSCL did not reduce data usage, so it cannot be concluded that it is more efficient in this regard. Furthermore, TSCL requires more computational resources, as discussed in this work, and demands an ad-hoc implementation with many more parameters to tune compared to traditional algorithms like SGD. In fact, all improvements observed in relation to Chavat's report were due to other techniques that are simpler and more established in the field, rather than TSCL.

References

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., et al. (2017). A closer look at memorization in deep networks. *34th International Conference on Machine Learning, ICML 2017*, *1*, 350–359.

Avramova, V. (2015). *Curriculum Learning with Deep Convolutional Neural Networks* (Master's thesis). KTH ROYAL INSTITUTE OF TECHNOLOGY. http://www.diva-portal.org/smash/get/diva2:878140/FULLTEXT01.pdf

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. *Proceedings of the 26th annual international conference on machine learning*, 41–48.

Chang, H.-S., Learned-Miller, E., and McCallum, A. (2017). Active bias: Training more accurate neural networks by emphasizing high variance samples. *Advances in Neural Information Processing Systems*, *30*.

Chavat Pérez, F. (2022). Modelos seq2seq para la transcripción de documentos del archivo berrutti.

Cichon, J., and Gan, W.-B. (2015). Branch-specific dendritic ca 2+ spikes cause persistent synaptic plasticity. *Nature*, *520*(7546), 180–185.

Cruzar, sistematización de archivos militares. (2019). https://cruzar.uy/ (accessed: 30.05.2024)

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

El-Bouri, R., Eyre, D., Watkinson, P., Zhu, T., and Clifton, D. (2020). Student-Teacher Curriculum Learning via Reinforcement Learning: Predicting Hospital Inpatient Admission Location. http://arxiv.org/abs/2007.01135

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, *48*(1), 71–99.

Etcheverry, L., Agorio, L., Bacigalupe, V., et al. (2021). A computational framework for the analysis of the uruguayan dictatorship archives. *Qurator 2021-Conference on Digital Curation Technologies, Berlin, Germany, 8-12 feb, page 1-15, 2021.*

Glorot, X., and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). PMLR. http://proceedings.mlr.press/v9/glorot10a.html

Gong, M., Li, H., Meng, D., Miao, Q., and Liu, J. (2018). Decomposition-based evolutionary multiobjective optimization to self-paced learning. *IEEE Transactions on Evolutionary Computation*, *23*(2), 288–302.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2014). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. https://arxiv.org/abs/1312.6211

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*.

Guo, S., Huang, W., Zhang, H., Zhuang, C., Dong, D., Scott, M. R., and Huang, D. (2018). CurriculumNet: Weakly supervised learning from large-scale web images. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11214 LNCS*, 139–154. https://doi.org/10.1007/978-3-030-01249-6_9

Hacohen, G., and Weinshall, D. (2019). On the power of curriculum learning in training deep networks. *arXiv preprint arXiv:1904.03626*.

Hendrycks, D., and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Jiang, L., Meng, D., Mitamura, T., and Hauptmann, A. G. (2014). Easy samples first: Self-paced reranking for zero-example multimedia search. *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, 547–556. https://doi.org/10.1145/2647868.2654918

Jiang, L., Meng, D., Yu, S. I., Lan, Z., Shan, S., and Hauptmann, A. G. (2014). Self-paced learning with diversity. *Advances in Neural Information Processing Systems*, *3*(January), 2078–2086. https://papers.nips.cc/paper/5568-self-paced-learning-with-diversity.pdf

Jiang, L., Meng, D., Zhao, Q., Shan, S., and Hauptmann, A. G. (2015). Self-paced curriculum learning. *Proceedings of the National Conference on Artificial Intelligence*, *4*(February 2019), 2694–2700.

Jiang, L., Zhou, Z., Leung, T., Li, L. J., and Fei-Fei, L. (2017). MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels. *arXiv*.

Kakerbeck, V. (2018). *Progressively Growing Neural Networks for Scene* (Master's thesis) [Retrieved from https://github.com/vkakerbeck/Progressively-Growing-Networks/blob/master/Masterthesis_Final.pdf]. University of Osnabrück, Germany.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv*, 1–26.

Kay, A. (2007). Tesseract: An open-source optical character recognition engine. *Linux J.*, *2007*(159), 2.

Khan, F., Zhu, X., and Mutlu, B. (2011). How do humans teach: On curriculum learning and teaching dimension. *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*, 1–9.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., and Desjardins, G. (2016). Overcoming catastrophic forgetting in neural networks - supporting information. *Pnas*, 1–6. http://www.pnas.org/content/suppl/2017/03/14/1611835114.DCSupplemental/pnas.201611835SI.pdf

Kumar, M. P., Packer, B., and Koller, D. (2010). Self-Pace Latent Variables. *NeurIPS*, 1–9. https://papers.nips.cc/paper/3923-self-paced-learning-for-latent-variable-models.pdf

Leyendo unidos para interpretar los archivos. (2019). https://mh.udelar.edu.uy/luisa/ (accessed: 30.05.2024)

Li, H., and Gong, M. (2017). Self-paced convolutional neural networks. *IJCAI*, 2110–2116.

Loshchilov, I., and Hutter, F. (2015). Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*.

Mai, Z., Li, R., Jeong, J., Quispe, D., Kim, H., and Sanner, S. (2021). Online continual learning in image classification: An empirical survey. *arXiv preprint arXiv:2101.10423*.

Mallol-Ragolta, A., Liu, S., Cummins, N., and Schuller, B. (2020). A curriculum learning approach for pain intensity recognition from facial expressions. *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020) (FG)*, 534–538. https://doi.org/10.1109/FG47880.2020.00083

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. (2019). Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, *102*(3), 419.

Meng, D., Zhao, Q., and Jiang, L. (2015). What Objective Does Self-paced Learning Indeed Optimize? http://arxiv.org/abs/1511.06049

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nesmachnow, S., and Iturriaga, S. (2019). Cluster-uy: Collaborative scientific high performance computing in uruguay. *Supercomputing: 10th International Conference on Supercomputing in Mexico, ISUM 2019, Monterrey, Mexico, March 25–29, 2019, Revised Selected Papers 10*, 188–202.

NVIDIA. (2023). *Train with mixed precision*. https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Platanios, E. A., Stretcu, O., Neubig, G., Poczos, B., and Mitchell, T. M. (2019). Competence-based curriculum learning for neural machine translation. *arXiv preprint arXiv:1903.09848*.

Sangineto, E., Nabi, M., Culibrk, D., and Sebe, N. (2018). Self paced deep learning for weakly supervised object detection. *IEEE transactions on pattern analysis and machine intelligence*, *41*(3), 712–725.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Shrivastava, A., Gupta, A., and Girshick, R. (2016). Training region-based object detectors with online hard example mining. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 761–769.

Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. (2022). Curriculum learning: A survey. *International Journal of Computer Vision*, *130*(6), 1526–1565.

Spitkovsky, V. I., Alshawi, H., and Jurafsky, D. (2010). From baby steps to leapfrog: How "Less is more" in unsupervised dependency parsing. *NAACL HLT 2010 - Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, (June), 751–759.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhuber, J. (2013). Compete to compute. *Advances in Neural Information Processing Systems*, (June).

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tan, M., and Le, Q. V. (2021). EfficientNetV2: Smaller Models and Faster Training. *1*, 1–12. http://arxiv.org/abs/2104.00298

Thai, A., Stojanov, S., Rehg, I., and Rehg, J. M. (2021). Does Continual Learning = Catastrophic Forgetting? http://arxiv.org/abs/2101.07295

Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J. (2019). An Empirical Study of Example Forgetting During Deep Neural Network Learning. *Iclr2019*, 1–19.

Tsvetkov, Y., Faruqui, M., Wang, L., MacWhinney, B., and Dyer, C. (2016). Learning the curriculum with Bayesian optimization for task-specific word representation learning. *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, *1*, 130–139. https://doi.org/10.18653/v1/p16-1013 van Hasselt, H., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning values across many orders of magnitude. *arXiv preprint arXiv:1602.07714.*

Wang, W., Caswell, I., and Chelba, C. (2019). Dynamically composing domain-data selection with clean-data selection by" co-curricular learning" for neural machine translation. *arXiv preprint arXiv:1906.01130.*

Wang, X., Chen, Y., and Zhu, W. (2021). A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(9), 4555–4576.

Weinshall, D., Cohen, G., and Amir, D. (2018). Curriculum learning by transfer learning: Theory and experiments with deep networks. *arXiv preprint arXiv:1802.03796.*

Wick, C., Reul, C., and Puppe, F. (2020). Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition. *Digital Humanities Quarterly*, *14*(1).

Wu, J., Li, L., and Wang, W. Y. (2018). Reinforced co-training. *arXiv preprint arXiv:1804.06035.*

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144.*

Xu, B., Zhang, L., Mao, Z., Wang, Q., Xie, H., and Zhang, Y. (2020). Curriculum learning for natural language understanding. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 6095–6104.

Xu, C., Tao, D., and Xu, C. (2015). Multi-view self-paced learning for clustering. *Twenty-Fourth International Joint Conference on Artificial Intelligence.*

Zhang, X., Kumar, G., Khayrallah, H., Murray, K., Gwinnup, J., Martindale, M. J., McNamee, P., Duh, K., and Carpuat, M. (2018). An empirical exploration of curriculum learning for neural machine translation. *arXiv preprint arXiv:1811.00739.*

Zhao, Q., Meng, D., Jiang, L., Xie, Q., Xu, Z., and Hauptmann, A. G. (2015). Self-paced learning for Matrix factorization. *Proceedings of the National Conference on Artificial Intelligence*, *4*, 3196–3202.

Zhou, Y., Yang, B., Wong, D. F., Wan, Y., and Chao, L. S. (2020). Uncertainty-aware curriculum learning for neural machine translation. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 6934–6944.

APPENDICES

Appendix 1

Base Model Experiments: traditional training

This appendix includes further experiments, details and discussion with respect to the improvements proposed in our work with respect to the base model with traditional SGD training.

1.1. Vocabulary and Image Augmentation

This experiment expands the one presented at Section 4.2.2, showing how those changes in the vocabulary interact with the Image Augmentation strategy proposed. The results for vocabulary experiments with and without augmentation are shown in Table 1.1.

When augmentation is not used, the more changes in the vocabulary, to make it more robust, is the better for the final performance of the model, as previously discussed. However, when we use augmentation, the behaviour is the other way around: instead of improving the performance, as the vocabulary

Augment	Vocabulary	Epoch	LCS	CER	Train loss	Eval loss
0	Fixed charset	8	0.713	0.305	0.647	2.082
	Computed charset + accents	10	0.716	0.304	**0.542**	2.115
	Computed charset + accents + min_count=2	9	**0.721**	**0.298**	0.587	**2.061**
1	Fixed charset	10	**0.735**	**0.284**	0.685	**1.963**
	Computed charset + accents	10	0.728	0.290	**0.678**	1.979
	Computed charset + accents + min_count=2	10	0.722	0.296	0.682	2.055

Table 1.1: This table shows the results of running the base model for 10 epochs under different settings with respect to the vocabulary being used, as the incremental improvements proposed over base, with and without Image Augmentation.

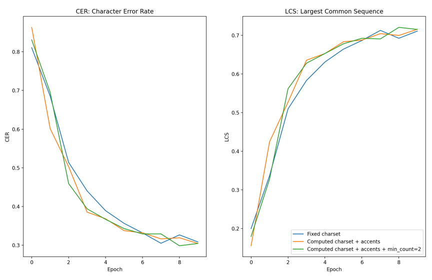


Figure 1.1: CER and LCS over time for non augmented scenario.

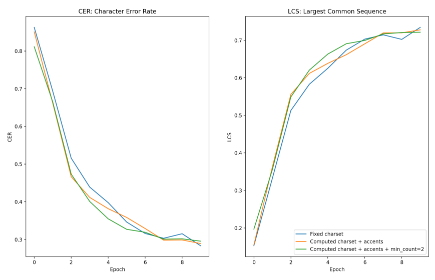


Figure 1.2: CER and LCS over time for augmented scenario.

Figure 1.3: Figures 1.1 and 1.2 show the evolution of CER and LCS for these scenarios. It can be seen that the trajectories are similar across all scenarios

become smaller, the performance decreases when image augmentation is used.

In summary: our incremental improvements proposed and validated without augmentation are not true when image augmentation is used, and in fact, looks like a deterioration, which is counter intuitive since image augmentation should improve all scenarios.

However, when comparing the training trajectories in Figure 1.3 they are really similar in all cases, specially for the augmented scenario. Maybe, the reason for these unexpected results in the augmented dataset scenario, is just because the models' training did not converged yet, which takes longer for the augmented scenario, so, maybe if models are run for larger sessions, these differences are more visible. This is particularly important for the fix on accent marks, that only applied to a small percentage of examples: since the model is not completely fit after 10 epochs, there are yet too much errors and such small improvement can not be noticed. This reasoning can not be applied to the reduction on the vocabulary size: the big cut from the original fixed one to this smaller computed on the fly, should have more effect early on the training, when the model makes a lot of errors. Since at the very beginning these errors should be more common, then it is more likely that the model assigns a character that is not even present in the dataset. However, as the training goes on, the model learns not to chose these out of actual vocabulary chars. During the training it did not see a single example that encourages the model to output this char. So, as the model improves, it probably will not choose a character that never saw, and representing it or not in the output sequence has probably no effect in the long term.

Max epochs	Beam size	Augmentation rate	epoch	LCS	CER	train loss	val loss
50	1	0	46	0.752	0.265	**0.082**	3.367
	1	1	35	0.767	0.252	0.349	2.167
	3	1	35	**0.781**	**0.236**	0.349	nc
20	1	0	19	0.746	0.273	0.268	2.371
	1	1	20	0.755	0.262	0.474	**2.02**
	3	1	20	0.769	0.247	0.474	nc
	1	0 (Chavat)	11	0.74	0.282	-	-

Table 1.2: Metrics over validation dataset. This table shows that just running for 20 epochs without augmentation already improves the results compared to what is reported by Chavat, shown in the last row. This is mainly due to the changes in the vocabulary. If image augmentation is included, then the improvement is even greater, and it gets better when beam search is used. Those rows with a beam size of 1 correspond to not using beam search, as discussed earlier. When more epochs are allowed, all alternatives improve up to 4.5%, which is for the CER in the beam search scenario. In summary, for CER the strengthened base model reports an improvement of near 16.3% (from 0.282 to 0.236), and almost 6% for the LCS (from 0.74 to 0.781).

Model	LCS	CER	Δ LCS	Δ CER	cumulative Δ LCS	cumulative Δ CER
Chavat	0.74	0.282	0	0	0	0
+ Changes in vocabulary	0.746	0.273	0.006 (0.8%)	0.009 (3.2%)	0.006 (0.8%)	0.009 (3.2%)
+ Image Augmentation	0.755	0.262	0.009 (1.2%)	0.011 (4.0%)	0.015 (2.0%)	0.020 (7.1%)
+ Beam Search	0.769	0.247	0.014 (1.9%)	0.015 (5.7%)	0.029 (3.9%)	0.035 (12.4%)
+ More Epochs	0.781	0.236	0.012 (1.6%)	0.011 (4.5%)	0.041 (5.5%)	0.046 (16.3%)

Table 1.3: The table illustrates the contributions of each improvement to enhance the base model for various metrics. Each row represents a single incremental change compared to the previous row. Δ column shows how the performance changes in relation to the preceding row, showcasing the individual contribution of each row. The cumulative Δ columns depict the accumulated improvements relative to Chavat's work, which is shown in the first row.

1.2. Strongest base model: credit blame

This section proposes to show how each of the proposed changes discussed in Section 4.2 contributes to increase the performance of the model.

The results in Table 1.2 show different scenarios with respect to the configurations tested, and this table can be seen as an extension of Table 4.6. To illustrate the improvement that each modification yields, Table 1.3 presents the metrics following a constructive narrative of the changes purposed to the base model.

In summary, it can be seen that, when running for the same number of epochs, our results surpass those reported by Chavat, with or without augmentation, which validates the changes in the vocabulary. It is also clear that the greatest improvement comes from the beam search mechanism. Finally, the best results are achieved when 50 epochs are used, leading to a 0.236 in CER and 0.781 in LCS resulting in an improvement of near 16.3% and almost 6%, respectively, compared to what Chavat reports for the validation set.