An Examination of High-Entropy Alternatives of Connectionist Temporal Classification Loss for Optical Music Recognition Using Convolutional Recurrent Neural Networks

by

Hritik Saynganthone

THESIS

Presented to the Faculty of the Department of Computer Science

Golisano College of Computer and Information Sciences

Rochester Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

Rochester Institute of Technology

December 2025

An Examination of High-Entropy Alternatives of Connectionist Temporal Classification Loss for Optical Music Recognition Using Convolutional Recurrent Neural Networks

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Richard Zanibbi, Supervisor

Dr. Richard Lange, Reader

Dr. Joe Geigel, Observer

Acknowledgments

Thank you to the DPRL and its members Patrick, Bryan, Abishek, Ayush, Jacob and especially my advisor Dr. Zanibbi, all of whom contributed to improving and refining my work. Thank you to the RIT Game Symphony Orchestra and its incredible Eboard, for being my home away from home and giving me something to look forward to each week. And finally, thank you to my wife, Alyssa for being with me and loving me unconditionally.

Abstract

An Examination of High-Entropy Alternatives of Connectionist Temporal Classification Loss for Optical Music Recognition Using Convolutional Recurrent Neural Networks

Hritik Saynganthone, B.S.
Rochester Institute of Technology, 2025

Supervisor: Dr. Richard Zanibbi

The Connectionist Temporal Classification (CTC) loss function is the most commonly used loss function in the field of Optical Music Recognition (OMR). However, OMR suffers from a massive class imbalance problem, exacerbated by the fact that CTC loss is subject to the spiky distribution problem, wherein the blank token introduced by CTC is vastly overpredicted and appears in timesteps where it would make more sense to predict a non-blank token, since CTC will collapse repeated tokens into a single token. This work posits that alternative loss functions to CTC that optimize for an increase in entropy of the prior probability distribution output of the model will lead to better generalization and lower error rates. The three main loss functions tested are FocalCTC, SR-CTC, and EnCTC, each of which optimize for increased entropy for different aspects of the estimated prior distribution. Experiments are conducted on all three. Both FocalCTC and EnCTC show an improvement over baseline CTC.

Table of Contents

Acknowledgments **iii**

Abstract **iv**

List of Tables **viii**

List of Figures **ix**

Chapter 1. Introduction **1**

- 1.1 Problem Statement 2
- 1.2 Thesis Statement 3
- 1.3 Key Information Detection Tasks 4
- 1.4 Overview 4

Chapter 2. Music Background **6**

- 2.1 Musical Scores 6
- 2.2 Encoding Schemes of Musical Scores 8
- 2.3 The GrandStaff Dataset 13
  - 2.3.1 Ground Truth Images 14
  - 2.3.2 Ground Truth Encoding 15
  - 2.3.3 Tokenization of Humdrum Kern 17
  - 2.3.4 Conversion to MusicXML 19
- 2.4 Other Datasets 21
- 2.5 Summary 22

Chapter 3. Recognition Models & Loss Functions **24**

- 3.1 Convolutional Recurrent Neural Networks 24
  - 3.1.1 Other CRNN Alternatives 27
- 3.2 Connectionist Temporal Classification Loss 30

3.2.1 Implementation of CTC Loss 33
3.3 CTC Loss & the Spiky Distribution Problem 35
3.4 High Entropy Alternatives to CTC Loss 40
3.4.1 FocalCTC Loss 40
3.4.2 Smooth-Regularized CTC Loss (SR-CTC) 41
3.4.3 Maximum Conditional Entropy Regularization for CTC (EnCTC) 43
3.5 Other CTC Alternatives 44
3.5.1 Re-weighted CTC 44
3.5.2 Aggregation Cross Entropy 46
3.5.3 Wildcard CTC 48
3.5.4 Variational CTC 48
3.5.5 Entropy Regularized and Equal Spacing CTC 49
3.6 Summary 49

Chapter 4. Experimental Design & Results **52**
4.1 Evaluation Protocols and Metrics 52
4.2 Implementation Details 54
4.3 Stability and Control Experiment 56
4.3.1 Design 57
4.3.2 Results 57
4.3.3 Discussion 59
4.4 FocalCTC Hyperparameter Search 62
4.4.1 Design 62
4.4.2 Results 63
4.4.3 Discussion 70
4.5 Direct Entropy Modification Experiments 72
4.5.1 Design 73
4.5.2 Results 73
4.5.3 Discussion 76

Chapter 5. Conclusions **81**

Bibliography **83**

Appendix **89**

Appendix 1. Glossary **90**

Vita **105**

List of Tables

2.1 A set of datasets being considered for OMR research. 22

4.1 Stability test results. 57

4.2 Summary of the CER, SER, and LER on the GrandStaff test set for the FocalCTC experiment. The smallest result for each is bolded. 63

4.3 Summary of the CER, SER, and LER for the baseline CTC results present in Table 4.1, equivalent to FocalCTC when $\alpha = 1.00$ and $\gamma = 0$. 63

4.4 Summary of the number of epochs it took each FocalCTC condition for early stopping to occur. 63

4.5 Results for both SR-CTC and EnCTC presented alongside the baseline results as well as the conditions from FocalCTC that minimized each error metric. The smallest result for each error metric is bolded. 73

List of Figures

2.1 Examples of a single instrumental score (SIS), a multi instrumental score (MIS), and a pianoform score (PFS). 8

2.2 (a) and (b) show snippets of monophonic (single voice) SISs. (c) and (d) show the semantic encodings of Fig. 2.2a and Fig. 2.2b using the grammar from Calvo-Zaragoza et al. [6]. Anything in the image that would be played differently has a different symbolic representation, such as the D major key sign and the note with a sharp accidental. (e) and (f) show the agnostic encodings of Fig. 2.2a and Fig. 2.2b using the grammar from Calvo-Zaragoza et al. [6]. Each symbol has some kind of shape information such as a note type, accidental, or clef, and where on the staff that shape is located. `L` followed by a number is used to indicate a line and `S` followed by a line is used to indicate a space, indexing from the lowest visible line and the space above it as the first line and first space respectively. (g) and (h) show the split-agnostic encodings of Fig. 2.2a and Fig. 2.2b using the split-sequence grammar from Calvo-Zaragoza et al. [6,26]. These use the same idea as the agnostic encodings, except the heights are separated from their corresponding symbols. 11

2.3 An example image of a score from the GrandStaff dataset, rendered using the *Verovio Humdrum Viewer*. It consists of measures 12 through 15 of *Piano Sonata No. 2 in A Major, Op. 2, No. 2* by Ludwig van Beethoven. A lower quality JPG of this score is what is actually present in the dataset and used as testing data. Transposed versions of this snippet are used as training data. 15

2.4 A partial Kern transcription of the score in Fig. 2.3. The score has been rotated by 90 degrees so that visually aligns with the transcription. The barlines in Kern have been highlighted to show how the transcription aligns with the image. 16

2.5 Kern (left) and the semantic encoding from Calvo-Zaragoza et al. [6] (right) of the bass clef staff of Fig. 2.3. The encoding from Calvo-Zaragoza et al. [6] is presented in a vertical format in order to show how it aligns with the corresponding Kern. . . 18

2.6 The Kern (left) and Bekern (right) representations of the first two measures of the score presented in Fig. 2.3. The character "·" is used as a separation delimiter for tokenization and does not appear in any actual model output. 19

2.7 Bekern (left) and the agnostic encoding from Calvo-Zaragoza et al. [6] (right) of the bass clef staff of Fig. 2.3. The encoding from Calvo-Zaragoza et al. [6] is presented in a vertical format in order to show how it aligns with the corresponding Bekern. . 20

2.8 The Kern (left) and MusicXML (right) representation of a clef and key sign declaration for Fig. 2.3. 20

3.1 The CRNN architecture found in Ríos-Vila et al. [31]. 25

3.2 An example of flattening. The left shows an example of how the feature map of a score may be discretized, and the right shows the result of the concatenation. Note that this flattening happens on $f(\mathbf{x})$ – for clarity, an image from the ground truth has been used instead. 27

3.3 (a) shows the raw sequence π of length T whereas (b) shows the first part of function $\mathcal{B}$ where all duplicate tokens not separated by blanks are removed, and (c) shows the sequence from (b) but with all blanks removed. (c) is the final result of applying $\mathcal{B}$ to the original π from (a). 31

3.4 An example snippet of a correct output sequence (a), an example valid path that would generate it (b), and an example invalid path that would not (c). 32

3.5 All the possible transition states that can produce token s at time t, given possible states $t-1$. (a) and (b) show the cases where a transition from $s-2$ is not possible, whereas (c) shows the case where a transition from $s-2$ is possible. 35

3.6 The full computation array for $\alpha_{s,t}$. Each circle represents a token, colored blue if it contributes to the final sum in a meaningful way. Outgoing arrows represent the possible transitions that can occur from the current state, and incoming arrows represent all possible previous states that could have lead to the current state. The dotted arrows represent transitions that should be possible but the to/from states have been omitted for brevity. 36

3.7 The probability over time (left) and error signal graphs (right) at three different stages of model training when using CTC from Graves et al. [15]. Each colored line represents the signal from one token in the vocabulary. The dashed line represents that of the "blank" token. (a) represents the model early on in training, showing a roughly uniform distribution across all tokens. (b) is the model in the midst of training, where probability spikes are beginning to form. (c) is the model at the end of training, where there are distinct probability spikes at very short intervals. 37

3.8 Two example alignments for the third measure of the bass clef staff of Fig. 2.3. For the sake of clarity, a simple note name is used in place of a more descriptive token. On the left is the "spiky" alignment where blank tokens are predicted even at timesteps at which a symbol is present, such as in the timestep to the right of the "A". On the right is a non-spiky distribution, where non-blank tokens are predicted wherever it is reasonable to consider a symbol present. Note that alignment such as this is done on an image from the ground truth for illustrative purposes and would normally be done on the reshaped feature map. 38

4.1 The metrics CER (a), SER (b), and LER (c) for the baseline and all four random seeds shown across three bar graphs. This mirrors the results found in Table 4.1. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent. 58

4.2 The ground-truth (a) and baseline CTC (b) final model output of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is. 60

4.3 The ground truth (a) and baseline CTC (b) transcriptions used to render the scores in Fig. 4.2. The errors highlighted in red on the right reflect the errors highlighted in red within the rendering. Note that there are other transcription errors present on the right, where certain symbols (such as the *f* on the same line as the mistranscribed *d*) are missing an `n` modifier, representing a ♮. However, because of the key sign, these transcription errors are not shown in the rendering since having the natural and not having the natural are treated the same in this case. 61

4.4 A 3D mesh plot of the CER, showing all twelve conditions run and what their corresponding CER was. 64

4.5 A 3D mesh plot of the SER, showing all twelve conditions run and what their corresponding SER was. 64

4.6 A 3D mesh plot of the LER, showing all twelve conditions run and what their corresponding LER was. 65

4.7 A 3D mesh plot of the convergence time, showing all twelve conditions run and what their corresponding number of epochs was. 65

4.8 The metrics CER (a), SER (b), and LER (c) for the baseline and all twelve FocalCTC conditions. This mirrors the results found in Table 4.2. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent. 68

4.9 The ground truth (a), $\alpha = 0.25, \gamma = 0.5$ (b), and $\alpha = 0.25, \gamma = 2$ (c) final model output transcriptions of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is. 70

4.10 The metrics CER (a), SER (b), and LER (c) for the baseline, conditions from FocalCTC that minimized each error metric, SR-CTC, and EnCTC. This mirrors the results found in Table 4.5. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent. 74

4.11 Bar graphs showing the CER (a), SER (b), LER (c) of SR-CTC and the FocalCTC results of $\alpha = 0.75, \gamma = 2.0$. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent. 75

4.12 An example of SR-CTC incorrectly predicting a token due to discontinuous timesteps having influence on one another. Three spines are shown as the original input image had two voices in the top staff. The `*` token represents a spine placeholder, which in this case simply indicates no change from the previous clef and a new clef does not need to be rendered. 77

4.13 The baseline CTC (a) and EnCTC (b) final model output of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is. Note that the final model output for EnCTC did not produce any errors for this example. 79

4.14 The baseline CTC (a) and EnCTC (b) final model output of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is. 79

4.15 The value of the SER metric on the validation set for EnCTC over each training step. Since the model will stop after the SER has not improved in five epochs, this serves as an indication of when the model reached several potential convergence points before continuing to decrease the SER. 80

Chapter 1

Introduction

The broad field we are conducting research in is Optical Music Recognition (OMR), a subfield of computer vision. Much like how Optical Character Recognition (OCR) concerns itself with using image processing techniques to transcribe characters of natural language contained within images, OMR is defined as the transcription of existing sheet music contained within images and converting them into a computer-processable format such as MusicXML or Humdrum Kern.

Our motivation for studying OMR comes from both a historical and practical context. On the historical side, having digital versions of public domain pieces allows for greater ease of preservation since they would no longer be bound by a physical medium. While digital images of the scores would also achieve this effect, digital transcription also allows for another feature – the ability to modify the underlying music.

This directly leads us to the more practical side of OMR. Most scores in the modern age are engraved using software rather than by hand, by using notation tools such as MuseScore[1], Dorico[2], and Sibelius[3] among others, and being able to take an existing piece of sheet music and quickly have a digitally editable version allows for greater use of the source material in both professional and academic environments, such as rearranging or reorchestrating an existing musical work for a

---

[1] https://musescore.org/en

[2] https://www.steinberg.net/dorico/

[3] https://www.avid.com/sibelius

different genre or collection of instruments.

1.1 Problem Statement

Current OMR systems suffer from two class distribution problems. The first is the number of classes – due to its inherent complexity, modern music notation has a very high number of symbols, especially those that look visually similar. This means that when a model attempts to predict what may be in an image, the probability distribution is spread across many labels for the model to choose from. In other words, the confidence of the model that a symbol is correct generally decreases as the number of probable choices increases, as is the case with OMR, where many symbols are visually similar.

The second is class imbalance, wherein some symbols are far less likely to appear than some other symbols and therefore are unlikely to be predicted as a transcription in general, such as barlines being far more common than ties or slurs. This problem is exacerbated due to the Connectionist Temporal Classification (CTC) loss function [15], the most common loss function used in OMR. CTC loss allows models with recurrent output to not have the same length as the target sequence, achieved in part by introducing a special label known as a "blank". This loss suffers from the spiky distribution problem [8, 15, 20, 22, 41], which causes the model to overpredict the blank token introduced through CTC since it can validly appear several times in between every single "real" token from the output transcription.

Specifically, the research question we are studying is how do alternatives to CTC loss that optimize for greater entropy of the token and timestep probability distributions affect the performance of convolutional recurrent neural networks (CRNNs) designed for OMR? Although greater entropy would imply a greater amount of uncertainty within the model, optimizing for greater entropy allows the model to better represent the current amount of knowledge that it has as well as encourage the model to better explore the probabilistic space it occupies.

1.2 Thesis Statement

Several increased entropy losses have all demonstrated their ability to resolve the spiky distribution problem [11, 22, 40] due to two main principles. The first is the principle of maximum entropy which tells us that by optimizing for increased entropy given some stated prior knowledge, the least amount of bias is introduced into an estimated probability distribution [17, 18]. Therefore, the model is learning the probability distribution of the OMR tokens that best describes the amount of knowledge it has at any given time during training. The second is the fact that increased entropy also indicates to the model that it should continue exploring the probabilisitic space it occupies, due to the increased uncertainty of any one path. This leads to less overfitting to specific paths overall [22].

These results have yet to be demonstrated in OMR, which has a much more aggressive class imbalance problem even outside of blanks versus non-blanks. As such, it is worth evaluating their performance in OMR and seeing what, if any, tradeoffs there are. Our core hypothesis and claim is that alternatives to CTC loss that increase the entropy of a CRNNs output probability distribution in some fashion (i.e., on the level of timesteps, tokens, or paths) will increase the generalizability of CRNNs designed for OMR and lead to lowered error rates when compared to the baseline CTC implementation.

1.3 Key Information Detection Tasks

OMR can be broken up into two sub-tasks: a visual symbol recognition task and a symbol sequence transcription task. The former is typically done via an encoder, such as a convolutional neural network (CNN) [2,6,27,29,31,38], and latter via a decoder, such as a recurrent neural network (RNN) [2,6,31,38] or transformer [27,29,31]. Dependent on the dataset being used, the exact set of symbols may change but should include these at a bare minimum: noteheads, barlines, accidentals, key sign, time signature, clef, note rests, multimeasure rests, grace notes, and fermati [6]. Very few also attempt to recognize articulations [38], lyrics [12], or dynamics [43].

1.4 Overview

The rest of this document is broken down as follows. Chapters 2 and 3 are background chapters that discuss relevant details for understanding our research. Chapter 2 will discuss the relevant background needed to understand the information extraction tasks for OMR. It will go over what a musical score is, how they are encoded for computer use, and the dataset being used for all conducted experiments. Chapter 3 will discuss the most common model architecture for OMR, that being a convolutional recurrent neural network (CRNN) – inputs and outputs of each layer of the architecture will be discussed. Additionally, we will go over how the Connectionist Temporal Classification (CTC) loss function works as well as the spiky distribution problem which stems from its implementation. Three alternatives to CTC loss are then presented, each with their respective argument on how entropy of the estimated prior probability distribution is increased. These are FocalCTC, Smooth-Regularized CTC (SR-CTC), and Maximum Conditional Entropy Regularization for CTC (EnCTC).

Chapter 4 contains all the experiments conducted used to confirm our hypothesis. The evaluation protocols used and implementation details of the model architecture are also provided. Three experiments are conducted. The first is to ascertain the nature of nondeterminism present with PyTorch's implementation of the CTC loss function. The second is to perform a hyperparameter search for FocalCTC. The third is a direct comparison of the baseline CTC result, the best FocalCTC results, SR-CTC result, and EnCTC result. All three experiments have design, results, and discussion presented.

Chapter 5 contains the conclusions drawn from the experiments conducted in Chapter 4. We confirm if our initial hypothesis was correct and provide further clarifications as to why the results were achieved. We additionally provide notes regarding future work that may be conducted regarding our same research question.

Chapter 2

Music Background

This chapter contains the relevant musical background needed to understand the information extraction tasks used for OMR. We will cover what a musical score is, how they are usually encoded beyond symbolic notation, as well as what dataset we chose for our research and why we chose it. Additionally, we will describe how each of these aspects is relevant to the input and outputs of an OMR model. At the end, we will briefly discuss alternatives to the dataset chosen, and go over why they were not chosen instead.

2.1 Musical Scores

To begin, a musical score is a document that contains the notation of a piece of music. In essence, it is a complete symbolic description of a piece of music that provides direct instructions on how to recreate the piece in question by a musician. The contents of a score are usually broken down into three categories:

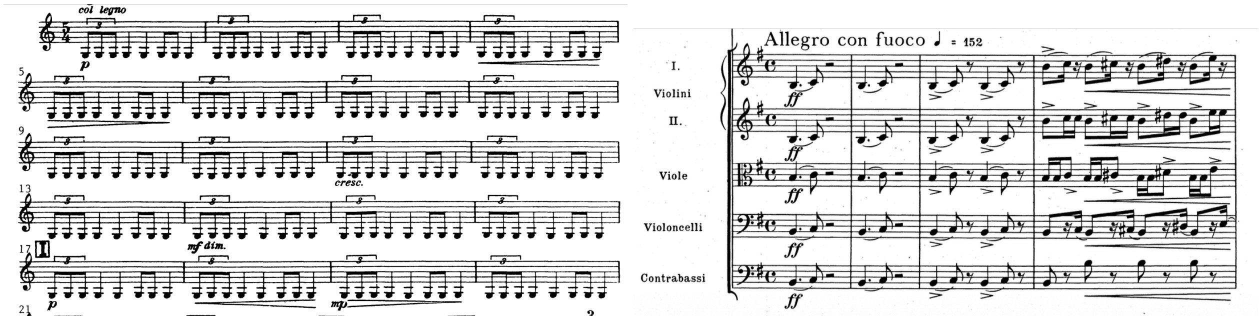
- **Single Instrumental Scores (SIS)**: these only contain a part for a single instrument separated across multiple staff systems – i.e., there are multiple horizontal sequences of measures per page (seen in Fig. 2.1a).
- **Multi Instrumental Scores (MIS)**: these contain several instruments playing at the same time, though typically with different rhythms. Parts may or may not contain multiple voices.

It is typical to see one staff system per page, i.e. though there are multiple lines of measures per page, they represent the same series of measures but played across different instruments simultaneously (seen in Fig. 2.1b).

- **Pianoform Scores (PFS)**: these parts only contain the Piano but usually have multiple voices spread across a Grand Staff – a special kind of staff in music that usually consists of a staff with a treble clef and a staff with a bass clef, linked together by middle C (as a note, the clefs on each staff are sometimes subject to change depending on the piece). These scores often have more complexity due to the having multiple voices and inherently having more than one staff per system. Otherwise, this is the same as a single instrumental score (seen in Fig. 2.1c).

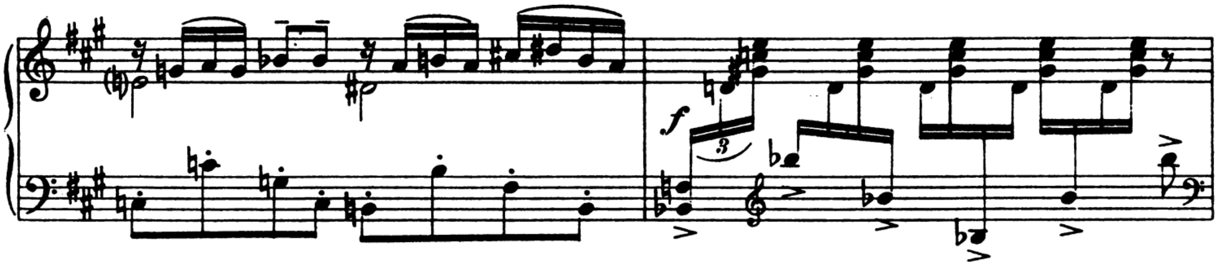
To clarify, the complexity of a score can be thought of as an indicator of how much information is contained within. More complex scores often ask the musician to do more on a per-beat or per-measure basis, such as playing multiple notes or rhythms simultaneously on a polyphonic instrument, like the Piano. An example of this can be seen in Fig. 2.1c, where we see the second voice in the first measure of the top staff (i.e. the E♮ with the stem going down) having a different rhythm and note to that of the first voice (i.e., the phrase starting on the 16th rest followed by a G♮) in the same bar.

In the research literature, images of these scores are the input into OMR models, which may be in either traditional western [6,13,24,25,30,35] or mensural (medieval choir) notation [7,25,32]. In many OMR datasets, this is compressed and stored in a JPG format [6,25,31] to reduce dataset size, though there are examples of images with no compression applied [5]. Which of these is used depends on what the model itself was designed to handle, although some models may opt to use



(a) Example of an SIS, an excerpt from the Violin I part of *Mars, Bringer of War* composed by Gustav Holst.

(b) Example of an MIS, an excerpt from the string section of *Symphony No.9 "From the New World", IV: Allegro* by Antonín Dvořák.



(c) Example of a PFS, an excerpt from the Piano part of *Rhapsody in Blue* by George Gershwin.

Figure 2.1: Examples of a single instrumental score (SIS), a multi instrumental score (MIS), and a pianoform score (PFS).

some combination of the above when performing their training. Although JPG allows the image file size to be smaller it often creates compression artifacts that may introduce inaccuracies when training.

2.2 Encoding Schemes of Musical Scores

Since a computer cannot read symbolic notation directly, the symbols must be encoded into a string of symbols that represent some musical score. Notation software such as MuseScore use such encoding schemes as their internal representation of a score, such as MSCZ[1] or MusicXML[2]. As such, some form of encoding is also the ground truth transcription of datasets, which is used by models and thus are also what the model outputs, save for any post processing. Methods as detailed as MusicXML are often too complex for OMR models to use directly [24] since they are meant to capture any and all possible kinds of notation, common or not, for the purposes of interoperability between different notation software. As a result, datasets often employ simpler encodings.

Several encoding schemes have been proposed, largely being separated into two classes: *semantic encodings* and *agnostic encodings* [6]. Each describes a different kind of tokenization scheme, specifically regarding the amount of information being contained within a given encoding token. Semantic encodings are less "primitive" than agnostic ones since tokens therein represent all the information contained within some piece of notation, such as what the duration and pitch of a note is rather than just one or the other, or an entire key signature (semantic) rather than a disjoint set of sharps or flats (agnostic). This distinction can be thought of symbols characterizing represented information versus visual information. Semantic encoding symbols describe what a piece of notation actually means to a musician, while agnostic encoding symbols describe what some notation visually looks like on the page.

The reason these schemes matter is because the manner in which a score is encoded directly impacts the transcription accuracy. As has been shown, using a purely semantic encoding scheme tends to lead to lack of overall learning since music is semantically dense [38]. Even for scores with certain details omitted such as articulations or dynamics, there tends to be hundreds if not thousands of unique symbols, many of which visually appear exactly the same when viewed in

---

[1] https://musescore.org/en/handbook/3/file-formats#musescore-native-format

[2] https://www.w3.org/2021/06/musicxml40/tutorial/introduction/

isolation, such as a sharp ($\sharp$) appearing on a key sign versus it appearing as an accidental next to note, as seen in Fig. 2.2a and 2.2b.

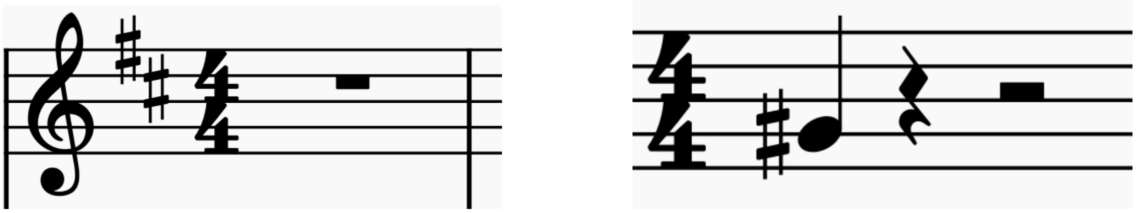
As such, some form of agnostic encoding has generally been shown to be the most effective method of encoding a transcription [31, 38], since it reduces the number of tokens the model must learn and therefore lowers the variance in predicted tokens. For example, we can predict a sharp as a sharp regardless of context since there is only a single symbol for it. In that same vein, encodings that reduce the symbol set further, such as in split-agnostic encodings where shape and height information are encoded separately (described in more detail below), have been shown to increase the transcription accuracy [2, 3, 26].

Semantic Encoding In a semantic encoding, every kind of symbol is given a unique token, regardless of any similarities regarding the shape composition tokens or their vertical positioning. Such a scheme can be seen in Fig. 2.2c and 2.2d, where the key of D major and the note G$\sharp$ are separate tokens in the vocabulary despite both containing a common element, a sharp, as seen in Fig. 2.2a and 2.2b.

Agnostic Encoding In an agnostic encoding, all visual symbols are given their own token, paired with some indication of positioning within the musical staff. This position information is needed, since that was lost when going from a semantic representation to an agnostic one. As we can see in Fig. 2.2e and 2.2f, this manifests as the token for key-sign no longer being present and the token for a notehead no longer containing any information about a sharp. Instead, the sharp is given its own token appended with a height indicator token.

---

[3] Scores were created by hand using MuseScore 4.



(a) The key of D Major, represented by two sharps – one on the F staff line and one on the C staff space.

(b) The note G sharp (G♯), where the sharp symbol is placed before the notehead and the G staff line goes through the midpoint of the symbol.

`keySignature-DM, timeSignature-4/4, rest-whole`

(c)

`timeSignature-4/4, note-G#4_quarter, rest-quarter, rest-half`

(d)

`clef.G2-L2, accidental.sharp-L5, accidental.sharp-S3,`
`digit.4-L4, digit.4-L2, rest.whole-L4, barline-L1`

(e)

`digit.4-L4, digit.4-L2, accidental.sharp-L2,`
`note.quarter-L2, rest.quarter-L3, rest.half-L3`

(f)

`clef.G2, L2, accidental.sharp, L5, accidental.sharp, S3,`
`digit.4, L4, digit.4, L2, rest.whole, L4, barline, L1`

(g)

`digit.4, L4, digit.4, L2, accidental.sharp, L2,`
`note.beamedRight0, L2, rest.quarter, L3, rest.half, L3`

(h)

Figure 2.2: (a) and (b) show snippets of monophonic (single voice) SISs. (c) and (d) show the semantic encodings of Fig. 2.2a and Fig. 2.2b using the grammar from Calvo-Zaragoza et al. [6]. Anything in the image that would be played differently has a different symbolic representation, such as the D major key sign and the note with a sharp accidental. (e) and (f) show the agnostic encodings of Fig. 2.2a and Fig. 2.2b using the grammar from Calvo-Zaragoza et al. [6]. Each symbol has some kind of shape information such as a note type, accidental, or clef, and where on the staff that shape is located. `L` followed by a number is used to indicate a line and `S` followed by a line is used to indicate a space, indexing from the lowest visible line and the space above it as the first line and first space respectively. (g) and (h) show the split-agnostic encodings of Fig. 2.2a and Fig. 2.2b using the split-sequence grammar from Calvo-Zaragoza et al. [6, 26]. These use the same idea as the agnostic encodings, except the heights are separated from their corresponding symbols.

3

The practical difference between this and the semantic encoding is that if a symbol has the same shape and height in two different contexts, then the same token from the vocabulary can be used in both scenarios. For example, if there was an F♯ on the top line of the staff, the same symbol used within the key sign (a sharp on the F staff line) can be used in output representation. We can see this in Fig. 2.2e and 2.2f as all sharp symbols in both are represented by the same token in both encodings (i.e. `accidental.sharp`), just with a different height attached (i.e. `L5` and `S3` for Fig. 2.2a and `L2` for Fig.2.2b).

Split-agnostic encoding (also known as split-sequence encoding) is a kind of agnostic encoding, but instead of each symbol having both shape and height information, each symbol only contains one or the other [2,3,26]. In an actual encoding this manifests in an alternating sequence (i.e. pairs) of shapes and heights, as shown in Fig. 2.2g and 2.2h, where we begin with a shape followed by a height, followed by a shape, followed by a height and so on. We can see that the aforementioned figures resemble their counterparts in 2.2e and 2.2f but with a split at every instance of `-`, the character the separates the shape and height information.

Furthermore, symbols that look the same on a page are encoded via the same token, since usually the only distinction between them would be the vertical position which is not encoded with the shape. We can see this in Fig. 2.2g and 2.2h, in which sharps in both figures use the same shape token but have different height tokens following them.

Though this encoding method is incredibly similar to non-split agnostic, it has been shown to provide better results, because of reduced variation in the tokens being predicted, which stems from a smaller vocabulary [2,3,26]. For example, Fig. 2.2g shows we no longer have multiple tokens for sharps depending on their physical position on the staff – they are simply followed with a different height token.

2.3 The GrandStaff Dataset

The primary dataset being used for our research is the GrandStaff dataset [31]. Other datasets such as Capitán [7] were considered as a part of our preliminary work since they are relatively small (around 900 samples) and allowed me to gain a deeper understanding of the underlying model architecture we were using at the time.

GrandStaff was chosen for three main reasons, the first being the notation type – all images in the GrandStaff dataset are in modern western notation, commonly used and the most broadly applicable to how music is read in the western world. Corollary to this is the encoding type of the dataset, which uses Humdrum Kern. Kern is a format that can be easily translated in and out of more widely used formats such as MusicXML, unlike the formats of Calvo-Zaragoza et al. [6] wherein no translation is readily available between it and formats that can actually be used by software. Ergo, samples in GrandStaff can be easily viewed and analyzed, used when performing analysis of experimental results.

Finally, there is the data itself and the model associated with it. GrandStaff contains a very large amount of training data, over 40,000 sample images [31], which is sufficient enough to obtain meaningful results. Moreover, the model proposed in Ríos-Vila et al. [31] was designed to directly use the GrandStaff dataset and provided a good baseline implementation of the model architecture we wanted to test. Thus, this dataset and the model implementation designed for it were chosen for our experiments.

This section will therefore describe the two main aspects of the GrandStaff dataset – the ground truth images and transcriptions for each image. It will also go into detail on the encoding scheme for the transcriptions and the properties contained therein.

2.3.1 Ground Truth Images

At a high level, the ground truth images for GrandStaff consist of several pianoform score snippets from well-known composers such as Beethoven and Chopin. These snippets are around three to six bars (measures) in length, such that all images were all a single system (i.e. they did not span multiple lines of grand staff measures). "Nongraphic" symbols were removed from the scores such as dynamics, expression text, slurs, and lyrics as to not clutter the images unnecessarily [31].

In order to augment this dataset, all snippets were transposed by three different intervals, meaning each snippet produced seven samples: an original, three transposed up a major second, a minor third, and a major third respectively, and three transposed down by the same intervals. This had the effect of introducing new accidentals (i.e., some notes that previously had no accidental may now have one, or have their existing one changed or removed), changing key signs (e.g., moving a score up by a major second has the effect of adding two sharps to the key sign), and causing certain notes to be re-beamed as they went from having their stems pointed in one direction to another [31].

Finally, all scores were rendered using the freely available *Verovio Humdrum Viewer*[4] and converted to JPG for space. Distorted versions of all samples also exist that are meant to emulate how images of scores may be captured through a camera [31], but they were not used in any of our experiments. An example score from the dataset can be seen in high quality in Fig. 2.3.

---

[4] https://verovio.humdrum.org/

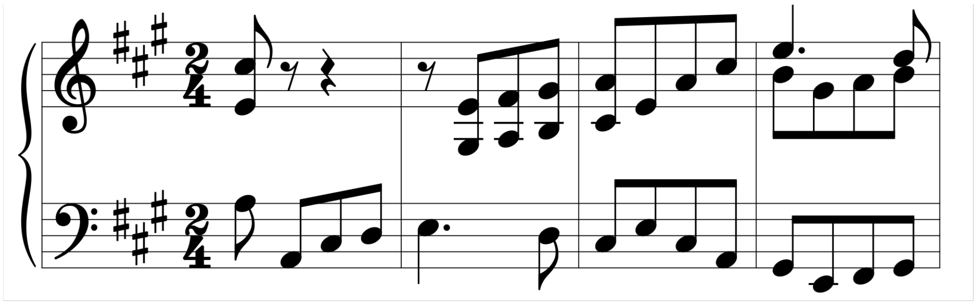
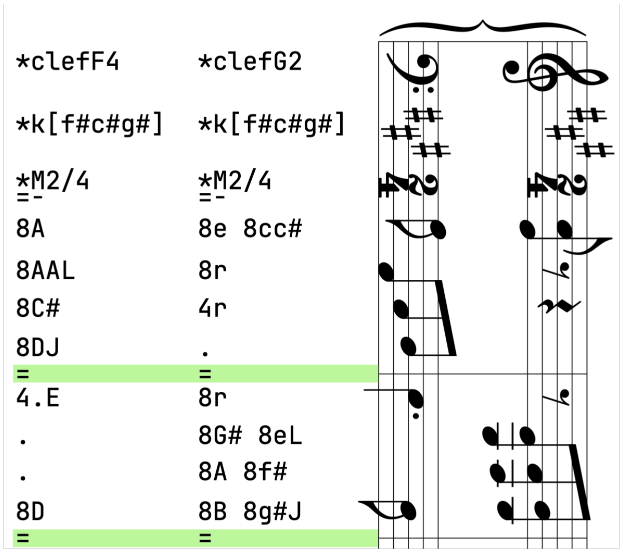


Figure 2.3: An example image of a score from the GrandStaff dataset, rendered using the *Verovio Humdrum Viewer*. It consists of measures 12 through 15 of *Piano Sonata No. 2 in A Major, Op. 2, No. 2* by Ludwig van Beethoven. A lower quality JPG of this score is what is actually present in the dataset and used as testing data. Transposed versions of this snippet are used as training data.

2.3.2 Ground Truth Encoding

As mentioned previously, the ground truth transcriptions for GrandStaff are in an encoding format known as Humdrum Kern, or just "Kern" for short. Kern starts by assigning each staff of music it encodes as a "spine", wherein the bottommost staff corresponds to the leftmost spine. Each spine is separated using the *tab* character, allowing multiple spines to be present within a Kern file. Additional spines may be inserted to represent either additional staves, or additional voices within a single staff. An example of how a Kern file corresponds to Fig. 2.3 can be seen in Fig. 2.4.

Each line within a Kern file contains sets of characters separated by spaces and potentially tabs if there are multiple spines. Anything that is on the same line can be conceptualized as having the same vertical position within a staff, corresponding to events that are happening at the same moment in time. If multiple notes are happening at the same time within a staff (usually as a part of a chord or multiple voices happening in parallel), then the symbols that represent those notes will be separated via the *space* character within a single spine. For example, on the first beat of Fig. 2.4 the bottom staff has an eighth note of the note A and the top staff has two eighth notes



```
*clefF4      *clefG2
*k[f#c#g#]   *k[f#c#g#]
*M2/4        *M2/4
=-           =-
8A           8e 8cc#
8AAL         8r
8C#          4r
8DJ          .
=            =
4.E          8r
.            8G# 8eL
.            8A 8f#
8D           8B 8g#J
=            =
```

Figure 2.4: A partial Kern transcription of the score in Fig. 2.3. The score has been rotated by 90 degrees so that visually aligns with the transcription. The barlines in Kern have been highlighted to show how the transcription aligns with the image.

playing different pitches, that being E and C♯. The A is represented via `8A` where the `8` represents the eighth note and the `A` represents the note A in this octave. Similarly, `8e` and `cc#` represent the E and C♯ of the top staff, and they are separated by a space to indicate they should be played at the same time. In the same vein, the *tab* is used to distinguish notes that happen at the same time but on different staves, and as such is placed between the `8A` and `8e`.

When moving to the next division of time, a *newline* is placed. However, each newline does not correspond with the same interval of time. Instead, the amount that is moved from one line to the next is entirely dependent on what the previous line contained. Generally, a newline will move the next line forward in time by a duration equal to whatever the shortest duration was in the previous line. For example, a line that contains an eighth note as its shortest duration will mean the next line will be one eighth note ahead, such as going from lines five to six in Fig. 2.4, whereas a line that contains a half note as its shortest duration will mean the next line is a half note ahead.

Each symbol within in Kern (i.e., the units separated by whitespace characters) represent some kind of musical notation, such as a key sign (seen in line 2 of Fig. 2.4), rests (seen in line 5 of Fig. 2.4), or notes. Furthermore, the symbols for notes consist of certain characters that represent characteristics such as pitch, duration, accidentals, and beaming. The Kern specification[5] does not enforce any kind of canonical ordering of these symbols which creates the issue of multiple different encodings for the same visual notation. Ríos-Vila et al. [31] therefore does indeed enforce an ordering of duration, pitch, and optionally accidental and beaming, which is present in all transcriptions of all GrandStaff images.

2.3.3 Tokenization of Humdrum Kern

The most straightforward way to tokenize kern is to separate by whitespace (i.e., spaces, tabs, and newlines). This yields a semantic encoding, since every notational symbol gets its own symbol based on both its pitch and duration. For example, G♯ and F♯ would be two different tokens if Kern is tokenized on whitespace alone. A comparison of this tokenization alongside the semantic encoding from Calvo-Zaragoza et al. [6] can be seen in Fig. 2.5. Note that since these whitespaces are also important to how Kern is encoded, each is also given their own token in the vocabulary to be predicted like any other token. These are decoded into their functional equivalents at the end of prediction. For the sake of clarity, they have been omitted from figures.

As mentioned before, agnostic encodings outperform their semantic equivalents [31,38], and thus an agnostic equivalent to the semantic tokenization is desired. Ríos-Vila et al. [31] achieved this by decomposing the symbols for notes into separate elements that could be used by multiple

---

[5] https://www.humdrum.org/rep/kern/

```
*clefF4                         clef-F4,
*k[f#c#g#]                      keySignature-AM,
*M2/4                           timeSignature-2/4,
=-
8A                              note-A3_eighth,
8AAL                            note-A2_eighth,
8C#                             note-C#3_eighth,
8DJ                             note-D3_eighth,
=                               barline,
4.E                             note-E3_quarter.,
.
.
8D                              note-D3_eighth,
=                               barline
```

(a) (b)

Figure 2.5: Kern (left) and the semantic encoding from Calvo-Zaragoza et al. [6] (right) of the bass clef staff of Fig. 2.3. The encoding from Calvo-Zaragoza et al. [6] is presented in a vertical format in order to show how it aligns with the corresponding Kern.

different kinds of notes, such as how a sharp can be used in both a G♯ and F♯. If separate tokenization occurs on duration, pitch, accidental, and beaming we get Basic Extended Kern [31] or "Bekern" for short. The "·" delimiter is used within an Bekern file in order to indicate what needs to be separated when tokenizing, as seen in Fig. 2.6b.

This gives an agnostic encoding, albeit not one that is as non-discriminative as the agnostic encoding from Calvo-Zaragoza et al. [6] since this encoding does not allow the accidentals within a key sign to be treated the same as those on notes, or the numbers within a time signature to be reused if they are duplicated. A comparison of these two schemes is provided in Fig. 2.7.

```
*clefF4      *clefG2
*k[f#c#g#]   *k[f#c#g#]
*M2/4        *M2/4
=-           =-
8A           8e 8cc#
8AAL         8r
8C#          4r
8DJ          .
=            =
4.E          8r
.            8G# 8eL
.            8A 8f#
8D           8B 8g#J
=            =
```

(a)

```
*clefF4      *clefG2
*k[f#c#g#]   *k[f#c#g#]
*M2/4        *M2/4
=-           =-
8·A          8·e 8·cc·#
8·AA·L       8·r
8·C·#        4·r
8·D·J        .
=            =
4·.·E        8·r
.            8·G·# 8·e·L
.            8·A 8·f·#
8·D          8·B 8·g·#·J
=            =
```

(b)

Figure 2.6: The Kern (left) and Bekern (right) representations of the first two measures of the score presented in Fig. 2.3. The character "·" is used as a separation delimiter for tokenization and does not appear in any actual model output.

2.3.4 Conversion to MusicXML

For some background, MusicXML, as the name implies, uses several sets of nested tags in order to define what a score contains and what it looks like. Since it is designed with interoperability between different kinds of notation software, it uses a great deal of verbosity in order to communicate even simple or common notations. This is made clear in Fig. 2.8, where something as simple as a clef, key sign, and time signature takes not only significantly more lines, but far more actual symbols.

As such, it is not a commonly used notation for ground truth transcription [24], and hence why formats like Kern and others are used for OMR tasks. However, it does have one major advantage – unlike Kern, MusicXML can be read directly by notation software. Renderers such as

```
*clefF4                        clef.F4-L4,
*k[f#c#g#]                     accidental.sharp-L4,accidental.sharp-S2, accidental.sharp-S4,
*M2/4                          digit.2-L4,digit.4-L2,
=-
8·A                            note.eighth-L5,
8·AA·L                         note.beamedRight1-S1,
8·C·#                          note.beamedBoth1-S2,
8·D·J                          note.beamedLeft1-L3,
=                              barline-L1,
4·.·E                          note.quarter-S3, dot-S3,
.
.
8·D                            note.eighth-S3,
=                              barline-L1
```

(a) (b)

Figure 2.7: Bekern (left) and the agnostic encoding from Calvo-Zaragoza et al. [6] (right) of the bass clef staff of Fig. 2.3. The encoding from Calvo-Zaragoza et al. [6] is presented in a vertical format in order to show how it aligns with the corresponding Bekern.

```
*clefF4       *clefG2
*k[f#c#g#]    *k[f#c#g#]
*M2/4         *M2/4
```

```
<attributes>
  <divisions>2</divisions>
  <key>
    <fifths>3</fifths>
  </key>
  <time>
    <beats>2</beats>
    <beat-type>4</beat-type>
  </time>
  <staves>2</staves>
  <clef number="1">
    <sign>G</sign>
    <line>2</line>
  </clef>
  <clef number="2">
    <sign>F</sign>
    <line>4</line>
  </clef>
</attributes>
```

(a) (b)

Figure 2.8: The Kern (left) and MusicXML (right) representation of a clef and key sign declaration for Fig. 2.3.

the *Verovio Humdrum Viewer* are good when scores are being processed one at a time, but lack the ability to process scores en masse or modify specific elements such as notehead color – useful for when highlighting the differences between scores.

Fortunately, Kern has a one-to-one conversion to MusicXML, wherein all elements present within a Kern file have an equivalent translation in MusicXML. Using a command line tool known as "music21[6]," scores represented in the Kern format can easily be translated into MusicXML. Since tools such as MuseScore offer command line APIs, it is feasible to batch convert scores into MusicXML from Kern followed by rendering them in MuseScore for analysis – something used quite extensively when analyzing the results of our experiments.

2.4 Other Datasets

To date, several other datasets have been created for the purposes of OMR and other related fields. Most consist of a set of images and a set of annotated ground truths – a text transcription of whatever the image may contain. As mentioned before, not all datasets use the same form of semantic or agnostic encoding. Table 2.1 summarizes several of these alternatives that were discovered over the course of our research.

Though other datasets do exist for the purposes of OMR (such as those used on the musical object detection task), the ones described in Table 2.1 are particularly suited to the method we want to use since all the encoding schemes present are easy to use and understand whilst still being robust enough to handle the most common cases. As such, they were all datasets we initially explored before finally landing on GrandStaff.

---

[6] https://www.music21.org/music21docs/

[7] 2018 Semantic and 2018 Agnostic refer to the original encoding schemes put forth in Calvo-Zaragoza et al. [6].

Name	**Notation**	**Encoding**[7]	**Encoding Type**
PrIMuS [6]	Western SIS	MEI, Kern, 2018 Semantic, 2018 Agnostic	Semantic, Agnostic
Camera PrIMuS [5]	Western SIS	MEI, Kern, 2018 Semantic, 2018 Agnostic	Semantic, Agnostic
Capitán [7]	Mensural SIS	2018 Agnostic	Agnostic
Grandstaff-LMX [24]	Western PFS	Linearized MusicMXL	Agnostic
OLiMPiC [24]	Western PFS	Linearized MusicMXL	Agnostic
FMT [30]	Western SIS	2018 Agnostic, Kern	Semantic, Agnostic
FP-GrandStaff [28]	Western PFS	Ekern	Agnostic
SEILS [25]	Mensural SIS	MusicXML, MEI, Kern, 2018 Semantic, 2018 Agnostic	Semantic, Agnostic

Table 2.1: A set of datasets being considered for OMR research.

2.5 Summary

To summarize, a musical score is a document that contains notation, which provides direct instructions to the musician on how to recreate a piece of music. They can be categorized as a single instrumental score (SIS), multi-instrumental score (MIS), and pianoform score (PFS). Since machines are not able to read the notation from a document directly, an encoding scheme must be used that represents a score as text. These are broken down into semantic and agnostic encodings, where semantic encodings use a tokenization that is based on represented information within a musical symbol and agnostic encodings tokenize based on visual information of a musical symbol. The degree to which an encoding is agnostic can change, such as the case with split-agnostic encoding.

The dataset being used for the conducted experiments is the GrandStaff dataset [31], which consists of several thousand PFS training images, each of which is a transposed snippet of an existing classical work in the public domain. Each image is encoded in Humdrum Kern, an encoding format that aligns with a 90° rotation of the ground truth image. If tokenized based on whitespace, the resulting vocabulary would be a semantic encoding. However, Ríos-Vila et al. [31] proposed Bekern, an alternative tokenization scheme that separates based on duration, pitch, accidental, and beaming, which results in an agnostic encoding. Kern itself is unable to be directly read via notation software such as MuseScore, but has a one-to-one conversion to MusicXML, a much more standardized format that can be read by most notation software.

From this, we put forth that the input to the OMR model is an image from the GrandStaff dataset, that has some canonical transcription in Kern. The model obtains a vocabulary by preprocessing the Bekern encodings of all images in the dataset in order to make a list of all tokens that can appear. When the model is attempting to transcribe the image, it is these Bekern tokens that are output. Since Bekern is simply a way to tokenize Kern, the output of the model is meant to be a Kern representation of the score. As such, the model must learn how to recognize patterns within the input images in order to output the correct token sequence.

Chapter 3

Recognition Models & Loss Functions

This section will go over the relevant mathematical background regarding the models and loss functions used in OMR. Alternatives to the most commonly used arhcitecture will be presented for the sake of completeness. We will then discuss how the most commonly used loss function in OMR, that being the Connectionist Temporal Classification (CTC) loss function, suffers from the spiky distribution problem, and how it may be alleviated by optimizing for high entropy within the loss. Three alternatives to CTC will then be presented, each with their corresponding argument on how entropy is increased through their use. Additional loss functions discovered through review of literature will also be presented to ensure comprehensive coverage to alternatives to CTC loss.

3.1 Convolutional Recurrent Neural Networks

The most common kind of model used in OMR is the Convolutional Recurrent Neural Network (CRNN) [1–3, 6, 24, 31, 32]. The reason for this is because it is uniquely suited to the underlying tasks (feature detection and sequence generation) present in the transcription problem for images [34] due to each portion of the model specializing in one of these two tasks and information being passed between them. An example diagram of the architecture with loss function can be found in Ríos-Vila et al. [31] and can be seen in Fig. 3.1.

To begin, the input to an OMR model is an image of a score. In our work, this is a single-system PFS such as the one shown previously in Fig. 2.3. The output to the model is a text-based

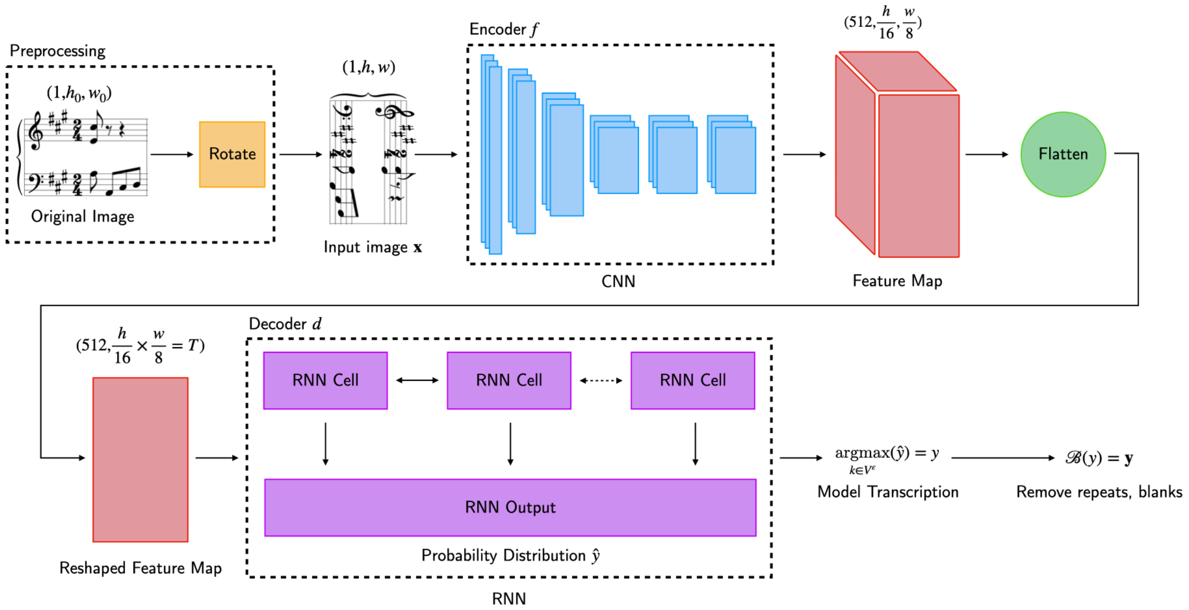


Figure 3.1: The CRNN architecture found in Ríos-Vila et al. [31].

transcription of notation in the image. This text output is based on a vocabulary obtained by the model and follows some kind of encoding scheme such as Kern as previously shown in Fig. 2.4. More formally, we define an input image $\mathbf{x}$ as having target output transcription l, where we define the model as being successful if it can correctly produce the output l only given $\mathbf{x}$ and the valid set of tokens that could be in l. We define this valid set of tokens as vocabulary V. We also introduce a special token ε, which will be used by the Connectionist Temporal Classification (CTC) loss function, as described later. Adding this to V gives us augmented vocabulary V^{ε} and a valid token k, where $k \in V^{\varepsilon}$.

The architecture of the model can be broken down into three main parts: preprocessing, a convolutional encoder, and a recurrent decoder. Before an input image comes into the model, some amount of preprocessing is done which can vary from implementation to implementation. This includes but is not limited to: image rotation to better align with the transcription [31], separation of the notes and staff lines [23], or even no preprocessing at all [6]. This preprocessed image is the input image $\mathbf{x}$ into the model.

Next, we define the encoder of our model f as taking $\mathbf{x}$ as input. This encoder consists of several convolutional and activation layers that compute visual features. The pooling and stride values as well as the number of convolutional and activation layers vary from implementation to implementation. This gives us a feature map $f(\mathbf{x})$, the size of which is dependent on the exact structure of f. For the model in Ríos-Vila et al. [31], $f(\mathbf{x})$ results in 512 feature channels, shown in Fig. 3.1. Each value in $f(\mathbf{x})$ is a set of features describing some fixed height and width cell of $\mathbf{x}$.

This output is then flattened, such that it may be used by the decoder. The method used to flatten differs depending on implementation [29, 31], but the result can be conceptualized as taking the vectors in $f(\mathbf{x})$ that correspond to features of vertical slices in the original sample image (not necessarily $\mathbf{x}$ depending on the preprocessing) and concatenating them together into a single dimension. This reshaped feature map has length T, where each element is a feature vector. T is determined based on the properties of each convolutional layer, as well as the height and width of $\mathbf{x}$. An example of this flattening step can be seen in Fig. 3.2.

We define the decoder of the model d as taking in the reshaped feature map as input. It is most often implemented as one or more bidirectional long short term memory units (BLSTMs) [6, 31]. Each cell in d corresponds to probability distribution $\hat{y}^t$, where t corresponds to a timestep, T is the total number of timesteps, and $1 \leq t \leq T$. $\hat{y}^t$ is defined then as the probability distribution over V^ε where $\hat{y}^t_k$ is the probability of outputting token k at timestep t. d results in a set of T probability distributions, defined as $\hat{y}$, equal to the set $\{\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^T\}$.

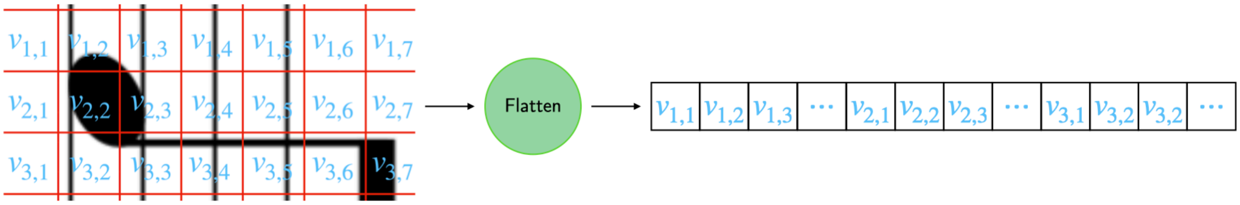

$v_{1,1}$ $v_{1,2}$ $v_{1,3}$ $v_{1,4}$ $v_{1,5}$ $v_{1,6}$ $v_{1,7}$
$v_{2,1}$ $v_{2,2}$ $v_{2,3}$ $v_{2,4}$ $v_{2,5}$ $v_{2,6}$ $v_{2,7}$
$v_{3,1}$ $v_{3,2}$ $v_{3,3}$ $v_{3,4}$ $v_{3,5}$ $v_{3,6}$ $v_{3,7}$
Flatten
$v_{1,1}$ $v_{1,2}$ $v_{1,3}$ ⋯ $v_{2,1}$ $v_{2,2}$ $v_{2,3}$ ⋯ $v_{3,1}$ $v_{3,2}$ $v_{3,2}$ ⋯

Figure 3.2: An example of flattening. The left shows an example of how the feature map of a score may be discretized, and the right shows the result of the concatenation. Note that this flattening happens on $f(\mathbf{x})$ – for clarity, an image from the ground truth has been used instead.

When inference occurs, the argmax over this sequence of probability tokens is then taken, wherein for each of the T probability distributions $\hat{y}^t$, we take the most likely token to be output at that timestep. The output of this model transcription is y, which has length T. This, in turn, is passed to $\mathcal{B}$, a function that comes from CTC that removes any repeat tokens in the output transcription not separated by blanks, followed by removing the blanks themselves. The output of this action results in the final model output $\mathbf{y}$, from which the loss is calculated. This output will have length $\leq T$ due to the process of removing symbols from y.

3.1.1 Other CRNN Alternatives

Although CRNNs with CTC is the basis for the conducted work, there exist other kinds of architectures that are worth mentioning as a point of comparison. No significant experiments were conducted on the following models, but are included to provide a robust background on what other models are used for OMR.

InterRNN+CTC This model, in essence, uses the same idea as a CRNN. The inputs and outputs as well as the use of convolutional layers, recurrent layers, and the CTC Loss Function are the same. The key difference is that the shape and height of engravings are encoded separately within different alphabets (i.e., a split-agnostic encoding) and models trained on just those alphabets are trained separately to begin with.

InterRNN is defined as follows: instead of there being one set of convolutional layers, there is a copy made. Additionally, a copy is made of the first recurrent layer as well. We then have both sets of the convolutional layers plus one recurrent layer train in parallel: one for the shape and one for the height. After the recurrent layer in these parallel networks, their output is fed into a concatenation block, which is sent to the final recurrent layer. $\mathcal{B}$ then removes duplicates and blanks as described before, the output of which is the prediction of the model $\mathbf{y}$ and used to calculate loss [2,3].

Sheet Music Transformer with Cross Entropy Sheet Music Transformer with Cross Entropy is somewhat of a departure from the above structures, as an autoregressive transformer is used instead of an RNN and cross entropy (CE) loss is used instead of CTC Loss (Eq. 3.1, 3.2, 3.3). The inputs and outputs are the same as above.

Once given a score, several convolutional layers perform feature detection on the image. A two-dimensional positional encoding is then appended to last layer's output, since music inherently has both a vertical and horizontal offset. That is sent to an autoregressive transformer consisting of self-attention, cross-attention, and feed-forward layers. When performing the recurrence, the previous predictions have a one-dimensional positional encoding applied based on their location within the output transcription. As one would imagine, the output of the autoregressive transformer (a tensor) is turned into a most-likely symbol prediction [24,29].

The motivation to continue using a CRNN as opposed to the transformer-based approach is due to a much larger body of work being present within OMR that utilizes the CRNN architecture as opposed to the transformer-based one. This allows us to better compare the results of our experiments with a broader set of OMR models. Moreover, since transformer-based approaches tend to use cross-entropy [24, 29] and the integration of CTC requires modifications to both the vocabulary as well as the model transcription, usage of the transformer-based approach would be contingent on being able to change the model architecture, something that was deemed beyond the scope of the research we wanted to conduct.

CNN-Based Music Object Detection Unlike the above mentioned architectures, models that fall into this category are not necessarily interested in the transcription task but the pure object detection task, where an image of a score is provided that could be any of the major types (SIS, MIS, or PFS) and the output is to provide a bounding box describing the location of musical objects such as noteheads, rests, accidentals, etc.

Most typically this is implemented using variations on R-CNNs, such as Faster R-CNN and Cascade R-CNN [9, 42]. The datasets used for models such as these also differ, since they need to provide some form of bound box ground truth as opposed to the music transcription of the image. Examples include DeepscoresV2 [36], MUSCIMA++ [16], and DoReMi [33], all of which include ground truth annotations for bounding boxes. These models have been able to perform object detection on many more classes that are being transcribed by the CRNN approach [42] and have even been able to become more highly specialized detectors for specific symbols such as accidentals [9].

For the research we want to conduct, it is atypical for the model architecture to provide the bounding box information in addition to the transcription, and as such it is difficult to compare the performance of the CRNN model to object detection models such as these.

3.2 Connectionist Temporal Classification Loss

At the core of this model is the Connectionist Temporal Classification (CTC) loss function which is meant to calculate loss when there is a recurrent output. Specifically, it solves the issue of transcription alignment, such that the recurrent output does not need to match the length of the target output. In the CRNN architecture with OMR, the output of our encoder $f(\mathbf{x})$ can be thought of as a list of extracted features, where the length of the list will be based on the dimensions of the input image $\mathbf{x}$. Since the dimensions of $\mathbf{x}$ do not necessarily have any bearing on how long the target transcription will be, it makes sense to use CTC since it allows for this misalignment to be present [6, 15, 19].

In order to make model output compatible with CTC, we first introduce a "blank" symbol (denoted ε) within the valid vocabulary V, denoted as V^{ε} [15, 39] as mentioned before. A "blank" in this instance corresponds to points within an image where no other symbol occurs, such as between notes in a single measure where visually there is only staff lines present. We can then formally define CTC as follows: if we consider the model transcription y, it has a length of T and is a generated sequence of tokens from V^{ϵ} (denoted as ${V^{\epsilon}}^{T}$). Recall that $\hat{y}_k^t$ is the probability of label k being selected at time t, where $k \in V^{\epsilon}$ and $1 \leq t \leq T$. We can define the likelihood of any label sequence of that length (denoted as π) [15] as:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} \hat{y}_{\pi_t}^{t}, \forall \pi \in {V^{\epsilon}}^{T} \tag{3.1}$$

From here, we define a many-to-one function $\mathcal{B}$ that takes our path π and maps it to another sequence in $V^{\leq T}$: the set of sequences over V of a length less than or equal to T. To do this, two operations are performed on π. The first is removing all tokens that appear more than once without a blank in between them and the second is removing all blanks in the resulting sequence [15]. We `a, a, <blank>, a, <blank>, b, <blank> b`

(a)

`a, <blank>, a, <blank>, b, <blank> b`

(b)

`a, a, b, b`

(c)

Figure 3.3: (a) shows the raw sequence π of length T whereas (b) shows the first part of function $\mathcal{B}$ where all duplicate tokens not separated by blanks are removed, and (c) shows the sequence from (b) but with all blanks removed. (c) is the final result of applying $\mathcal{B}$ to the original π from (a).

can see this operation in Fig. 3.3. It is worth noting that these two operations are performed in the order defined above.

Given model transcription y, we apply function $\mathcal{B}$ on this sequence in order to get our final label sequence, referred to as $\mathbf{y}$. A direct consequence of $\mathcal{B}$ is allowing the recurrent output be misaligned with the target output, since this allows the target sequence to be captured without the risk of label repetitions. Repeats indicate that the same object in $f(\mathbf{x})$ is present in multiple timesteps, meaning that it only occurs once in the original image $\mathbf{x}$ and thus all repeat instances can be removed. Removing blanks eliminates "empty" spaces in $\mathbf{x}$ that are not normally transcribed, such as between notes and staves or in moments of silence for automatic speech recognition (ASR) [15]. Actual repeats within $\mathbf{x}$ are still captured, since they occur on either side of a blank [15].

Using $\mathcal{B}$, we define the probability of generating a target sequence l (i.e., the sequence from ground truth that we want the model to predict) from all paths π that generate it as $\pi \in \mathcal{B}^{-1}(l)$ [15]:

$$p(l|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|\mathbf{x}) \tag{3.2}$$

This probability, in turn, is used to calculate the loss of the model. A question here is that if the

```
*M4/4
=-
4g#
4r
2r
```

(a) A partial Kern transcription of Fig. 2.2b

```
*M4/4
=-
4εgε#
4rεε
22rr
```

(b) An example path π that would generate the correct label sequence if passed through $\mathcal{B}$.

```
*M4/4
=-
4gεg#
4r
2r
```

(c) An example path π that would generate an incorrect label sequence if passed through $\mathcal{B}$.

Figure 3.4: An example snippet of a correct output sequence (a), an example valid path that would generate it (b), and an example invalid path that would not (c).

model only produces a single path as its canon sequence (i.e. the model only chooses the most likely next token in the sequence), then what exactly is being summed over? For our RNN, the output is a sequence of probability distributions of length T where each distribution is over V^{ϵ}. The idea is to compute the sum over all paths π that could be used to generate l and ignore all paths that could not.

As an example, consider the case presented in Fig. 3.4. If we consider Fig. 3.4a to be the target label sequence l, then the path π shown in 3.4b would be valid since $\mathcal{B}$ would convert it into l if the repeat labels are removed, followed by the removal all blanks. On the contrary, the path in Fig. 3.4c would be invalid since applying $\mathcal{B}$ would remove the ε on line three, resulting in `4gg#` instead of `4g#`.

Finally, to calculate the actual loss of the model $\mathbf{L}_{\text{CTC}}$, we define [15]:

$$\mathbf{L}_{\text{CTC}} = -\ln(p(l|\mathbf{x})) \tag{3.3}$$

and use standard backpropagation methods such as stochastic gradient descent to differentiate the loss with respect to the weights in the network. Optimizers such as Adam may also be used in order to dynamically set the learning rate based on gradients of the loss [6, 31].

3.2.1 Implementation of CTC Loss

Looking at the high level definition of CTC in Eq. 3.2 and Eq. 3.3, we see that we need to calculate the probability of all potential paths π that generate l. The simplest way to do this would be to calculate each probability individually and then sum them up as we compute them, but this would be very slow and computationally expensive for long paths. Instead, we can utilize dynamic programming to sum the probabilities of paths that reach the same point together at each timestep [15].

Since blank tokens are valid before after every token in the ground truth transcription l of length U, it is easier to work with sequences that include them. Let l^{ε} be the sequence l but with the blank character ε before and after each preexisting token:

$$l^{\varepsilon} = [\varepsilon, l_1, \varepsilon, l_2, \varepsilon, \ldots, \varepsilon, l_U, \varepsilon] \tag{3.4}$$

where $|l^{\varepsilon}| = 2U + 1$.

Next we define a two-dimensional array α indexed by s and t. Let $\alpha_{s,t}$ be the probability of paths reaching subsequence $l^{\varepsilon}_{1:s}$ after t timesteps, where s indexes l^{ε} and $1 \leq t \leq T$. The value of $\alpha_{s,t}$ is determined by observing the current state of the array at t, that being s, and seeing what other states at $t-1$ could have generated the state s. This array is initialized with two possible starting states at $t = 1$ since this sequence can either start with $s = 1$ or $s = 2$, equal to the probability of the first token being a blank or the first token being the first “real” token l_1 , respectively. We can do this because starting the path with a blank or l_1 will both lead to a sequence beginning with l_1 if $\mathcal{B}$ is applied, meaning they both generate the correct output label l_1 to start the sequence.

From here, we need to determine from what states can we transition from in only a single timestep from $t-1$ to t. Since repeats are removed, this means it is possible to transitions from state s to s, since this simply indicates that the same token was predicted multiple times in a row and we have not actually moved forward in the sequence. Next, it is possible to go from state $s-1$ to s since this indicates that we have moved from either a blank to a real token, or from a real token to a blank, moving forward in the sequence. Lastly, since we are allowed to skip over blanks, there are certain cases where we can go from $s-2$ to s in a single timestep. However, it is of more help to consider the cases where this *cannot* happen.

Say that we cannot go from $s-2$ to s, which means it is not possible to skip over $s-1$. Therefore one of two cases is happening. The first is that $l^{\varepsilon}_{s} = l^{\varepsilon}_{s-2}$, which only happens when there is distinct repetition of tokens in l, meaning that $l^{\varepsilon}_{s-1} = \varepsilon$ and it cannot be skipped. The second is when $l^{\varepsilon}_{s} = \varepsilon$, in which case l^{ε}_{s-1} a real token from l and cannot be skipped. In both cases:

$$\alpha_{s,t} = (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \times \hat{y}^{t}_{l^{\varepsilon}} \tag{3.5}$$

where $\hat{y}^{t}_{l^{\varepsilon}}$ is the probability of outputting token l^{ε}_{s} at time t as derived from Eq. 3.1. This case is shown in Fig. 3.5a and Fig. 3.5b. If we are able to validly go from state $s-2$ to s in addition to states $s-1$ and s, then $l^{\varepsilon}_{s-1} = \varepsilon$ and l^{ε}_{s-1} must be located in between real and unique tokens of l. As a result:

$$\alpha_{s,t} = (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \times \hat{y}^{t}_{l^{\varepsilon}} \tag{3.6}$$

This case is shown in Fig. 3.5c.

After we compute these probabilities for all possible valid values of t, we end up with two final values – one corresponds to the sequence ending with the last real token of l, and the other corresponds to the sequence ending with a blank. We take the sum of these two values in order

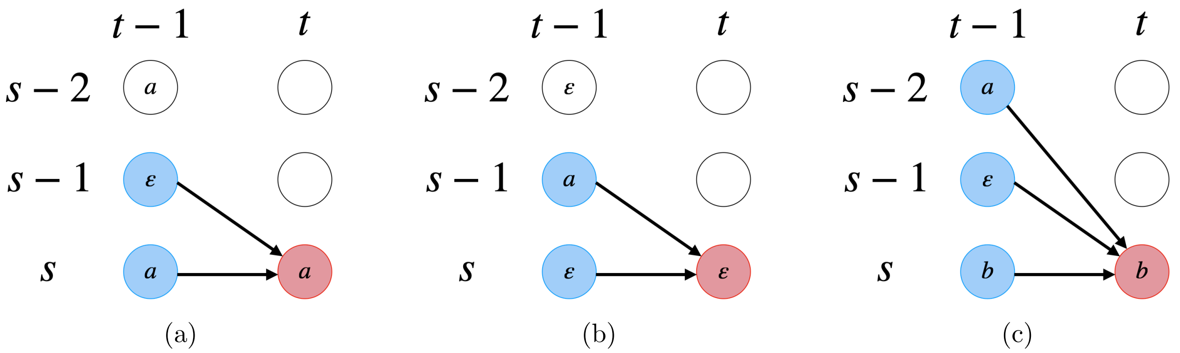


Figure 3.5: All the possible transition states that can produce token s at time t, given possible states $t-1$. (a) and (b) show the cases where a transition from $s-2$ is not possible, whereas (c) shows the case where a transition from $s-2$ is possible.

to get our final sum, equivalent to the calculation in Eq. 3.2. A diagram of the full computation array of $\alpha_{s,t}$ is shown in Fig. 3.6.

On most modern hardware, doing the above as stated will usually lead to underflow since we are repeatedly multiplying probabilities together. One way to alleviate this issue is to perform these calculations in log-space by taking the natural log of all quantities and converting operations to their logarithmic equivalent. The result can then be exponentiated at the end to undo the natural log.

3.3 CTC Loss & the Spiky Distribution Problem

As previously mentioned, most systems that use CTC loss suffer from the "spiky distribution problem" [8, 15, 20, 22, 41]. In essence, this is the problem of blank overprediction – since blank tokens can validly appear in between all true labels from l and multiple times in a row at that [15], the introduction of the blank token causes a massive class imbalance where, by the end of training, blanks are far more likely to be predicted than any other token at most timesteps, except for extremely short time intervals.

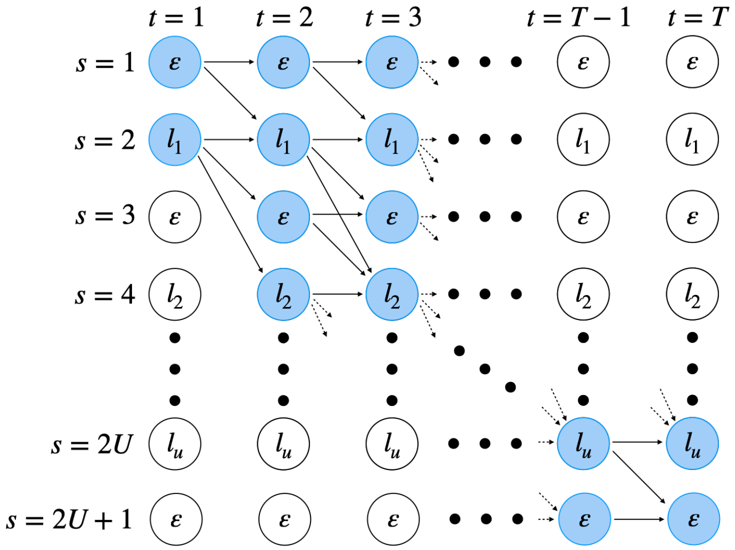


Figure 3.6: The full computation array for $\alpha_{s,t}$. Each circle represents a token, colored blue if it contributes to the final sum in a meaningful way. Outgoing arrows represent the possible transitions that can occur from the current state, and incoming arrows represent all possible previous states that could have lead to the current state. The dotted arrows represent transitions that should be possible but the to/from states have been omitted for brevity.

We can see this behavior in Fig. 3.7. Early on in training (Fig. 3.7a) the probability of predicting any token, blank or not, is about equal. However, by the time we get to the end of training (Fig. 3.7c) we see that blanks are consistently predicted with high probability except for very short intervals where another token is predicted instead. These “spikes” are where the spiky distribution problem gets its name.

These spikes manifest as a result of the blank token being predicted even in timesteps where a symbol is present. In other words, there are timesteps where the blank token is predicted even though it would make more sense and lead to better generalization if the model predicted the same non-blank token repeatedly rather than more blanks before and after some non-blank symbol. This is exemplified in Fig. 3.8a, in which the blank token is predicted in timesteps even where a note is

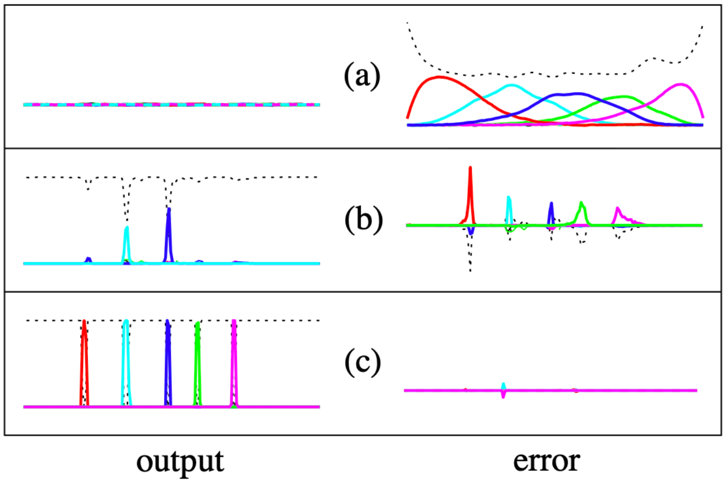
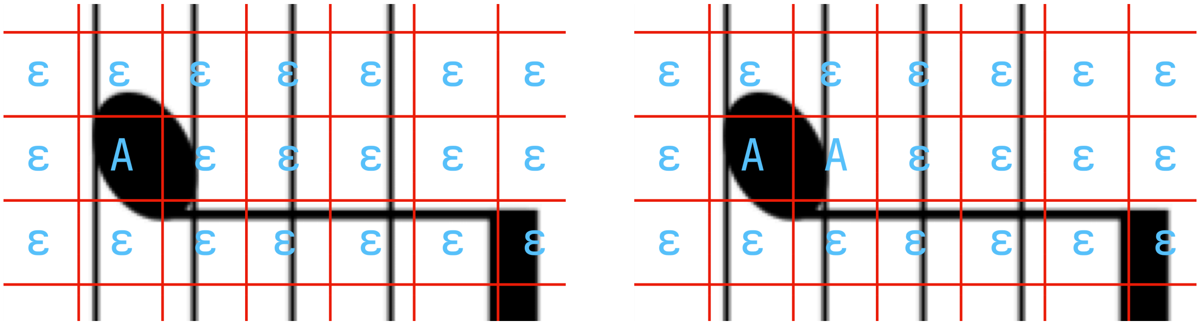


Figure 3.7: The probability over time (left) and error signal graphs (right) at three different stages of model training when using CTC from Graves et al. [15]. Each colored line represents the signal from one token in the vocabulary. The dashed line represents that of the "blank" token. (a) represents the model early on in training, showing a roughly uniform distribution across all tokens. (b) is the model in the midst of training, where probability spikes are beginning to form. (c) is the model at the end of training, where there are distinct probability spikes at very short intervals.

present. In contrast, a non-blank is predicted in nearly all timesteps where a note is located within Fig. 3.8b. As one may expect, models that do not account for this learn a probability distribution that is biased towards the blank token in nearly all timesteps, except for very small durations where symbols occur. This manifests as blanks being present in nearly all paths, which dominate training and skew the prior probability estimation towards predicting blanks more often than they should. Paths that contain an excess of blanks are thus known as "dominant paths".

Nonetheless, there is a way to alleviate both the spiky distribution problem and the issues of class imbalance as a whole – namely, the optimization of high entropy within loss calculation itself. Although it may seem counterintuitive, optimizing for higher entropy gives the model information on how much it has explored the probabilistic space it occupies and prevents the prior probability of the blank token from overpowering any features that may be present.



(a) (b)

Figure 3.8: Two example alignments for the third measure of the bass clef staff of Fig. 2.3. For the sake of clarity, a simple note name is used in place of a more descriptive token. On the left is the "spiky" alignment where blank tokens are predicted even at timesteps at which a symbol is present, such as in the timestep to the right of the "A". On the right is a non-spiky distribution, where non-blank tokens are predicted wherever it is reasonable to consider a symbol present. Note that alignment such as this is done on an image from the ground truth for illustrative purposes and would normally be done on the reshaped feature map.

This is due to the principle of maximum entropy. Consider the current state of some system that is probabilistic in nature, where there is is some amount of precisely stated prior knowledge on how the system achieved this state (e.g., knowing what has occurred and with what frequency each event occurred at). Then consider all the probability distributions that could have lead to the current state, where it is unknown what the exact probabilities of any one outcome are. The distribution that best represents the current amount of knowledge of the system is the one that contains the most entropy, since maximizing entropy can be thought of as having the least amount of "bias" towards any specific result [17, 18].

For example, consider a six-sided die that is rolled and lands on a six. There are many possible probabilities we could assign to each face, such as the six having greater odds. However, in the absence of other information, the best probability distribution to assume of the die is that all faces have the same probability of occurring, since it is the most representative of the current state of knowledge. If this die continued being rolled and it lands on a six three more times in a row, this changes the amount of knowledge we have and we may consider assigning the face with a six as having greater probability than that of the other faces. However, it would be ill-advised of us to assign a nearly 100% probability of a landing on a six since that biases the die in a way that has not yet been demonstrated. As such, it is still best for us to assume that the probability distribution of the die contains the maximum amount of entropy that is reasonable given the new information we have.

Therefore, by optimizing for high entropy, the model is optimizing for having a probability distribution that best represents the amount of knowledge it currently has. This means that the distribution itself provides the model with information. In particular, high entropy indicates that the model has not sufficiently explored the space and should try and remain as unbiased towards a single result as possible, whereas low entropy indicates the model is very confident in the results it is giving and may be prone to overfitting. Because of this, the high entropy optimization additionally encourages the model to explore the space longer than it otherwise may have [22].

As such, there are three main alternatives to CTC loss that, in some way, optimize for higher entropy. These are FocalCTC, Smooth-Regularized CTC (SR-CTC), and Maximum Conditional Entropy Regularization for CTC (EnCTC). However, the distribution wherein the entropy is actually being accounted for differs in each method. FocalCTC optimizes for increased entropy in each probability distribution $\hat{p}^t$ for each timestep across all tokens. SR-CTC optimizes for increased entropy in each probability distribution $\hat{p}_k$ for each token across all timesteps. And EnCTC optimizes for increased entropy in the probability distribution of paths $\pi \in \mathcal{B}^{-1}(l)$.

3.4 High Entropy Alternatives to CTC Loss

3.4.1 FocalCTC Loss

FocalCTC [11] is a method that borrows ideas from focal loss, a variation on cross entropy loss. To begin, focal loss is defined as:

$$\mathbf{L}_{\mathrm{F}} = -\alpha(1 - \hat{p}_t)^{\gamma} \ln(\hat{p}_t) \tag{3.7}$$

where $\hat{p}_t$ is the probability of predicting the true class.

With focal loss (shown in Eq. 3.7 [21]), the idea is to introduce two hyperparameters α and γ. The purpose of α is to help with class imbalance – it can either be set as a hyperparameter or can be dynamically determined based on the inverse class frequency and will scale the loss accordingly. It is usually set in the range $\alpha \in [0, 1]$ [21] but in theory can be set outside of that range. We can think of α as controlling the loss on a global level – as alpha increases, all probabilities not very near 1 will result in a loss increase.

The purpose of γ is to adjust the loss based on "easy" or "hard" examples. We can see that as γ increases, high values of $\hat{p}_t$ will result in comparatively lower loss values. Specifically, γ causes the loss to exponentially get smaller, and when the model is confident in its prediction $1-\hat{p}_t$ is very small, leading to $(1 - \hat{p}_t)^{\gamma}$ tending towards 1 and leaving the base loss unaffected [19, 21]. Comparatively, the model being less confident causes $1 - \hat{p}_t$ to be very large and thus the loss to be greater. That is, more focus is placed on examples that are "hard" with low $\hat{p}_t$ values [21]. As such, probabilities near 1 are more controlled by γ than α. An increase in γ corresponds to the loss allowing for lower values of $\hat{p}_t$ to result in lower loss values than they otherwise would have.

This same idea can be applied to CTC loss, with a simple substitution of the base equation from cross-entropy to CTC loss, as shown in Eq. 3.8 [11].

$$\mathbf{L}_{\text{FC}} = -\alpha(1 - p(l|\mathbf{x}))^{\gamma} \ln(p(l|\mathbf{x})) \tag{3.8}$$

Using the definition of CTC Loss (Eq. 3.3), we can redefine FocalCTC in terms of CTC [11]:

$$\mathbf{L}_{\text{FC}} = \alpha(1 - \mathbf{exp}(-\mathbf{L}_{\text{CTC}}))^{\gamma} \mathbf{L}_{\text{CTC}} \tag{3.9}$$

The same logic for the purposes of α and γ apply. It is worth mentioning that in practice α is set to a static hyperparameter for the duration of training when used with FocalCTC instead of being dynamic [11].

Entropy is increased in this method because the loss causes the model to focus on harder samples and therefore increase the weight of the tokens within these harder samples. The tokens therein inherently have lower probability of being predicted due to how uncommon they are or difficult they are to predict, hence why they appear in the harder samples. By having the model focus on these tokens, their probability of prediction will increase. In other words, this has the effect of causing tokens whose probability was initially quite low have been increased, leading to the distribution of the estimated prior probabilities to become more evenly distributed (i.e., to have greater entropy).

3.4.2 Smooth-Regularized CTC Loss (SR-CTC)

SR-CTC [41] posits that given the probability distribution $\hat{y}$, a "smoothed" version of $\hat{y}$ along the time axis for each token is a more accurate description of the "true" distribution, and should thus be taken into account when calculating the loss. Consider the other indexing variable of $\hat{y}$, that being k. We can define $\hat{y}_k$ as the probability distribution for some token $k \in V^{\varepsilon}$, where each element within this distribution describes what the likelihood of producing label k is at any timestep t. This means we can redefine $\hat{y}$ as a sequence $\{\hat{y}_{k_1}, \hat{y}_{k_2}, \ldots, \hat{y}_{k_{|V^\varepsilon|}}\}$ over all labels $k \in V^\varepsilon$. We define this smoothed distribution $\hat{y}_k^{(s)}$ as a convolution of each prior distribution $\hat{y}_k$ with kernel K:

$$\hat{y}_k^{(s)} = \hat{y}_k * K \tag{3.10}$$

Although K could be set to any smoothing kernel, for the experiments conducted in Yao et al. [41], $K = [0.25, 0.50, 0.25]$, which is the standard length-three one-dimensional smoothing kernel. Applying K to each token's output probability distribution $\hat{y}_k$ results in the smoothed distribution $\hat{y}^{(s)}$. Each timestep t in this new distribution has distribution $\hat{y}^{t(s)}$, which consists of the smoothed probabilities obtained from calculating $\hat{y}_k^{(s)}$ for each token at the given timestep. From here, SR-CTC is defined as:

$$\mathbf{L}_{\text{SC}} = \mathbf{L}_{\text{CTC}} + \beta \mathbf{L}_{\text{SR}} \tag{3.11}$$

where

$$\mathbf{L}_{\text{SR}} = \sum_{t=1}^{T} D_{\text{KL}}(\hat{y}^{t(s)} \mid\mid \hat{y}^t) \tag{3.12}$$

and β is a linear coeffcient that acts as a tuning parameter and is set in the range $[0, 1]$. $D_{KL}(Q \mid\mid P)$ is the Kullback-Leibler (KL) divergence, which measures the difference between two probability distributions Q and P. It can be conceptualized as the amount of entropy needed to describe the differences in distribution from P to Q.

In essence, the core idea behind SR-CTC is to enforce a degree of consistency between the non-smoothed and smoothed distributions. This method attempts to keep the entropy of the distribution high by accounting for it in the loss function. We can think of the act of smoothing in two different scenarios. When the probability distribution is already "smooth" (i.e. for each token, the probabilities across all timesteps are close) such as upon first model initialization, the entropy is high and the smoothing has little effect. In this instance, the smoothed distribution does not possess a great deal amount more entropy than the original.

However, when the probability distribution is spiky (as seen in left of Fig. 3.7c), the smoothed distribution does indeed possesses more entropy than that of the original estimated distribution, since application of the kernel at these points can be envisioned as spreading the high probability spikes out across more timesteps. By enforcing consistency between the original distribution and this higher-entropy distribution within the loss, the model is being optimized for keeping the entropy relatively high while it is training.

3.4.3 Maximum Conditional Entropy Regularization for CTC (EnCTC)

The idea of EnCTC is to directly incorporate an entropy regularization term into the loss calculation, such that the entropy of feasible paths (i.e., paths π that could generate l) does not drop too quickly [22]. That is, given the current probability distribution of paths, EnCTC ensures that the entropy remains higher than it otherwise would, for a longer amount of time. It is defined as:

$$\mathbf{L}_{\text{EN}} = \mathbf{L}_{\text{CTC}} - \beta H(p(\pi|\mathbf{x}, l)) \tag{3.13}$$

where β is a tuning parameter much like that in SR-CTC [41]. $H(p(\pi|\mathbf{x}, l))$ is the entropy regularization term with respect to all feasible paths π, defined as:

$$H(p(\pi|\mathbf{x}, l)) = -\frac{1}{p(l|\mathbf{x})} \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|\mathbf{x}) \ln p(\pi|\mathbf{x}) + \ln p(l|\mathbf{x}) \tag{3.14}$$

By keeping the entropy high, the loss is trying to steer the model away towards a single path or type of path. As explained before, these so-called “dominant paths” tend to be ones with an excess of blanks. By considering alternatives to these dominant paths, we are alleviating the spiky distribution problem since alternatives that better the input are considered, that being ones that do not have the excess of blanks [22].

This loss function is most direct with its optimization of high entropy within the loss itself. By keeping the entropy relatively high on a per valid path basis, the model is ensuring that it continues to explore alternative paths for longer. This can be understood by considering what happens when this regularization term is not present, i.e., in baseline CTC loss. Without the term present, the learned distribution produces a low entropy, resulting in overfitting to a few dominant paths. This low entropy is exemplified in the spiky distribution problem, as mentioned previously. This can lead to an overall lack of generalization, since other feasible paths may have better described the symbols features contained with each timestep but were not selected since the model was not being punished for having low entropy. Without a term such as the one above, those paths were very unlikely to be selected, as there was no incentive for the model to do so.

3.5 Other CTC Alternatives

While researching alternatives to CTC, we found several whose purpose was to solve an issue that has not been necessarily demonstrated in OMR or whose implementation would require the baseline CRNN architecture of the model to be modified in some way. Nevertheless, they are included here for comprehensive coverage of CTC alternatives.

3.5.1 Re-weighted CTC

Re-weighted CTC (RE-CTC) is defined as follows [19]:

$$\mathbf{L}_{\text{RE}} = -\mathbf{W} \ln(p(l|\mathbf{x})) = \mathbf{W}\mathbf{L}_{\text{CTC}} \tag{3.15}$$

where $\mathbf{W}$ is a weighting vector of length N where N is the total number of samples. N is used to weight every sample individually. Though strategies such as FocalCTC have proportionally greater loss for harder samples, the metric of "loss" to measure the difficulty of a sample may not be a good metric to be trained on since they cannot accurately determine the importance of any individual sample [19]. Instead, we utilize the metrics used in the evaluation calculations of the model. For OMR, we use the number of substitution errors, deletion errors, and insertion errors on a character, symbol, or line level. These will each be individually referred to as a "count".

$\mathbf{W}$ is initialized to 1 for all samples and then updated after every epoch. To calculate the update, we first define $N(i, e)$ as some partial or complete sum of the aforementioned counts, for sampple $\mathbf{x}_i$ in the e-th epoch. Then for sample $\mathbf{x}_i$ at the end of the e-th epoch, we update the corresponding weight $w(i, e)$ via:

$$w(i, e) = N \times \frac{w'(i, e)}{\sum_{j=1}^{N} w'(j, e)} \tag{3.16}$$

$$w'(i, e) = \alpha w(i, e-1) + \beta N(i, e) \times \frac{1}{N_{total}(i, e) \times e} \tag{3.17}$$

$$N_{total}(i, e) = \sum_{f=1}^{e} N(i, f) \tag{3.18}$$

In Eq. 3.17, the values α (> 0) and β (≥ 0) are tunable hyperparameters that determine the amount of sway the previous weight should have versus the contribution from this epoch. Furthermore, Eq. 3.16 provides us with a normalization term, ensuring that all weights sum up to N. Once the update is complete, the next epoch can begin, the loss of which will use the new weights determined from the previous term [19].

If we let $\mathbf{L}_{\mathrm{CTC}}(i, e)$ be the loss of sample $\mathbf{x}_i$ in epoch e, then [19]:

$$\mathbf{L}_{\mathrm{RE}} = w(i, e)\mathbf{L}_{\mathrm{CTC}}(i, e) \tag{3.19}$$

Since the metric we are evaluating is a physical count of the errors being produced, we can see that as the number of errors decreases, the weight of the term for that sample tends towards 0. Furthermore, we have the normalization from Eq. 3.16, meaning that for samples where the number of errors is high, we will have a proportionally greater weighting term, causing a greater loss to be calculated compared to those with few errors.

This loss was not used simply because allowing the loss access to the evaluation metrics would require the architecture of the model to be changed since the baseline CRNN architecture does not account for needing this information during the loss calculation [31].

3.5.2 Aggregation Cross Entropy

The core idea behind aggregation cross entropy (ACE) is to convert the sequence problem that is needed for CTC into a counting problem for the purposes of calculating loss. To illustrate the function more easily, we can redefine Eq. 3.3 as:

$$\mathbf{L}_{\mathrm{CTC}} = -\sum_{i=1}^{U} \ln(p(l_i|i, \mathbf{x})) \tag{3.20}$$

where U is the length of the target output sequence, and i describes a position within the output sequence [39].

The idea behind ACE is that instead of calculating loss relative to the correct output label l itself, we calculate it relative to the number of times a symbol appears:

$$\mathbf{L}_{\mathrm{ACE}} = -\sum_{j=1}^{|V^{\epsilon}|} \ln(p(N_{k_j}|\mathbf{x})) \tag{3.21}$$

where N_{k_j} is the number of times the symbol k_j appears in sequence l and $|V^{\epsilon}|$ is the length of the alphabet with the blank symbol included [39]. Recall $\hat{y}_k^t$ as the probability of outputting label k at timestep t. This is used to define aggregation $\hat{y}^k$:

$$\hat{y}^k = \sum_{t=1}^{T} \hat{y}_k^t \tag{3.22}$$

From here, Xie et al. [39] proposes two different methods based on how we can use $\hat{y}^k$ to predict the count for each class.

Regression Based ACE Based on the observation that:

$$max \sum_{j=1}^{|V^\epsilon|} \ln(p(N_{k_j}|\mathbf{x}) \Leftrightarrow min \sum_{j=1}^{|V^\epsilon|} (N_{k_j} - \hat{y}_{k_j})^2 \tag{3.23}$$

it can be derived that:

$$\mathbf{L}_{\mathrm{ACE}} = \frac{1}{2} \sum_{j=1}^{|V^\epsilon|} (N_{k_j} - \hat{y}_{k_j})^2 \tag{3.24}$$

This gives us an approximation of Eq. 3.21 from the perspective of regression [39].

Cross Entropy Based ACE As was observed by the original authors, regression based ACE suffered from an apparent gradient vanishing problem, and as a result proposed an alternative that borrowed the idea of cross entropy:

$$\mathbf{L}_{\mathrm{ACE}} = -\sum_{j=1}^{|V^\epsilon|} \overline{N_{k_j}} \ln(\overline{\hat{y}_{k_j}}) \tag{3.25}$$

where $\overline{N_{k_j}} = N_{k_j}/T$, $\overline{\hat{y}_k} = \hat{y}_k/T$. This allows us to calculate the effective difference between the predicted distribution and the correct distribution and base our loss on that difference.

Neither of these losses were used since this does not necessarily solve the spiky distribution problem nor class distribution problem of OMR. However, it has achieved better results than CTC in CRNN architectures for similar problems (such as text transcription from images) and thus would be worth testing in the future [39].

3.5.3 Wildcard CTC

In addition to the blank symbol, an “all” (denoted $*$) is also included within the alphabet. It is defined such that the probability of it being the correct token is always 1 regardless of what is present within the timestep. This method was initially developed for datasets in which the ground truth labels were incomplete in some capacity, such that this symbol could match degraded portions of the labels [4]. The loss calculation is otherwise unchanged, although sometimes a per-sample weighting term is also added [4]. In most datasets for OMR, ground truth labels are very complete and as such, the inclusion of a wildcard token would likely only degrade generated outputs.

3.5.4 Variational CTC

Variational CTC loss was designed very directly to solve the issue of class imbalance and spiky distribution problem by asserting that the distribution of blanks and non-blank tokens are not from the same distribution at all and should thus be predicted separately. As such, Var-CTC makes token predictions in two stages: first it sees if it should output a blank to begin with and if it determines that it should not, only then does it try to determine what non-blank should be predicted. In order to determine these two different distributions, three multi-layer perceptrons (MLPs) are used alongside a learned embedding of the vocabulary [8].

Chao et al. [8] proposes two variants of the above. Var-CTC uses an evidence-based lower bound with a statistical variational inference strategy in order to estimate the above probability distributions. An alternative, Mml-CTC, utilizes a direct method of calculation rather than estimation.

Although it would have been interesting to see how this would have compared since it tries to solve the spiky distribution problem so directly, there were two main factors that ultimately lead to this loss not being used. The first is the experimental results – Var-CTC was tested against several other CTC variants as well as the baseline and rarely outperformed even that [8]. Second is the implementation itself – implementing Var-CTC would require a significant holistic model change [8].

3.5.5 Entropy Regularized and Equal Spacing CTC

Also known as EnEsCTC, this loss function takes the EnCTC loss mentioned previously and adds an additional assumption, that being one of equal spacing. This states that elements in the input will be evenly spaced between one another – this somewhat does the opposite of entropy regularization since it is designed to prune out infeasible paths sooner [22]. Although the first part of the loss function could make sense since overfitting is an issue in OMR, the second assumption does not hold when recognizing sheet music as the spacing between elements can vary widely.

3.6 Summary

The most common type of model in OMR is convolutional recurrent neural network (CRNN). The convolutional layers act as an encoder f of the input image $\mathbf{x}$ and perform feature detection on defined two dimensional cells of the input image. The recurrent layers act as a decoder d of convolutional output $f(\mathbf{x})$ after flattening, and output a sequence of T probability distributions over the set of all labels in the vocabulary V^{ε}. At inference, sequence has the argmax taken wherein for each timestep t, the most likely token k is output at that timestep. This label sequence y is passed to CTC function (B) in order to remove repeat tokens and blanks, producing model output $\mathbf{y}$. There also exist alternatives to this architecture used in OMR, but were not chosen for the experiments conducted.

The Connectionist Temporal Classification (CTC) loss function is the most commonly used loss function in OMR. CTC loss is used in scenarios that involve recurrent output, specifically so that the length of the recurrent output does not necessarily need to match that of the target output. To implement it, a dynamic programming array is used, often calculated in log-space to avoid underflow.

CTC loss suffers from what is known as the "spiky distribution problem," where the blank token is overpredicted since it can validly appear between all true labels of ground truth transcription l and multiple times in row at that. By the end of training, the estimated prior probability distribution assigns the probability of blanks to be too high for nearly all timesteps, except for very short intervals. Paths that contain an overabundance of blanks are therefore more likely to be predicted, and known as "dominant paths".

One way this problem can be solved is by optimizing for higher entropy of the estimated prior probability distribution within the loss function itself. This utilizes the principle of maximum entropy, which states that the probability distribution that best represents the amount of current knowledge within a system is the one that maximizes the amount of entropy given the prior knowledge that is known at the time. As such, there are three alternatives to CTC loss that optimize for greater entropy, being FocalCTC, Smooth-Regularized CTC (SR-CTC), and Maximum Conditional Entropy Regularization for CTC (EnCTC).

FocalCTC optimizes for increased entropy within each probability distribution $\hat{p}^t$ by increasing the estimated prior probability distribution of tokens within harder samples, which inherently have a lower prior probability estimate. SR-CTC forces consistency between each estimated prior distribution $\hat{y}^t$ and $\hat{y}^{t(s)}$, where $\hat{y}^{t(s)}$ is determined based on $\hat{y}_k^{(s)}$, the smoothed distribution of each distribution $\hat{y}_k$. $\hat{y}_k^{(s)}$ has greater entropy than $\hat{y}_k$, ergo so does $\hat{y}^{t(s)}$ compared to $\hat{y}^t$. Finally, EnCTC incorporates an entropy regularization term directly within the loss calculation itself, optimizing for the increased entropy of all paths $\pi \in \mathcal{B}^{-1}(l)$. Since the dominant paths are the ones with an excess of blanks, increasing the entropy of all paths has the effect of decreasing the estimated probability of dominant paths while increasing the estimated probability of alternatives.

Based on this analysis of FocalCTC, SR-CTC, and EnCTC, we believe that they provide the best alleviation from the spiky distribution problem of baseline CTC due to their incorporation of greater entropy. As such, each of these loss functions was the primary focus of an experiment conducted to see if they were able to achieve lower error rates compared to that of baseline CTC.

Chapter 4

Experimental Design & Results

4.1 Evaluation Protocols and Metrics

This section describes what metrics the trained models will be evaluated on as well as what the core outcome of the research is and how we will know if our original hypothesis is correct. As is standard practice in the field, we will be evaluating on the following three metrics:

- **Character Error Rate % (CER)**: a count of incorrect characters from the final label predicted sequence compared to the ground truth transcription is taken, which is then divided by the total number of characters in the ground truth transcription. This gives the CER for a single sample. The CER of the model can be calculated by doing this same procedure cumulatively, where the counts are taken over the entire test set instead of a single sample.
- **Symbol Error Rate % (SER)**: same as above but for symbols contained within the obtained vocabulary.
- **Line Error Rate % (LER)**: same as above but for lines of the output. In this case, lines of Kern.

These metrics are calculated upon the test set after the model is done training.

Furthermore, since the implementation we used [31] only stops via early stopping and not via a maximum number of epochs, we will also include the number of epochs until convergence as another metric (also referred to as "convergence time"). Doing this demonstrates how efficiently the model trains, since a faster convergence (i.e., fewer epochs until early stopping occurs) means that the model learned at a faster rate compared to that of a slower convergence. These four metrics are the primary dependent variables for all experiments conducted.

It is worth noting the absence of metrics such as precision, recall, intersection-over-union (IoU), and F1. For visual detection tasks and related disciplines, it is common to include these metrics as a way to visually ascertain how correct the output of a model was – for approaches focused on musical object detection, these metrics are used [9,42]. However, since there is no visual component to the output of a CRNN model, metrics such as these do not hold much meaning. Moreover, since CRNNs are interested in the transcription task and not the pure object recognition one, these metrics are not considered relevant.

The core outcome of this research is to determine whether or not there exists a better loss function than standard CTC for the OMR transcription task, given the CRNN architecture and the above mentioned constraints. Ergo, the set of possible outcomes is enumerated as which loss function and their chosen hyperparameters provides the least error rate for the three evaluation methods listed above. These may or may not come from the same loss function.

Our hypothesis will be confirmed if at least one of the high entropy alternatives to CTC outperforms the baseline CTC implementation when evaluated on one of the aforementioned metrics.

4.2 Implementation Details

The baseline architecture for our research is the CRNN implementation from [31][1]. As mention before, this CRNN architecture differs slight than that of more traditional OMR systems [2, 6, 32] in that it rotates the initial input image (as seen Fig. 3.1) such that it may better align with how Kern transcribes images (as seen in Fig. 2.4). This implementation uses Python 3.9 and PyTorch Lightning 2.5.0 using PyTorch 2.5.1. Logging is performed through the platform *Weights and Biases*.

Several modifications have been made to the baseline implementation in order to facilitate the experiments we wanted to perform, two of which are worth mentioning. The most substantial of these was the addition of a *loss selector* – instead of using the built-in CTC loss function provided by PyTorch, we implemented a way for the loss to be selected through configuration parameters, allowing me to replace loss functions without needing holistic model architecture changes.

Second to this was the inclusion of intermediate saving, wherein the model will save a certain number of transcription predictions after each validation epoch. These consist of some that are consistently saved each epoch (i.e., the same samples are being saved after each validation at every validation) and some that are randomly saved (i.e., a sample is randomly chosen to be saved during each validation). The purpose of this was for the study of models as training was occurring.

After training, these files are analyzed by a command line tool known as *musicdiff*, which allows one to take two music files and "diff" them in a way not dissimilar to that of Git. This tool, in turn, utilizes another tool known as *music21*[2], which provides many command-line APIs for processing different encodings of sheet music. Useful for *musicdiff* is its ability to convert

---

[1] https://github.com/multiscore/e2e-pianoform

[2] https://www.music21.org/music21docs/

from formats like Kern to MusicXML, and modify the contents of files like MusicXML – such as highlighting notes in different colors and adding staff text. Using this, *musicdiff* is able to process two Kern files (such as a ground truth file and a prediction) and annotate the differences between the two files.

As such, a modified version of *musicdiff* was used alongside a custom made Python script to directly process the output files produced by the model. These modifications were made in order to fix certain bugs within *musicdiff*, such as the bottom note of a chord always being highlighted as incorrect if there was any issue within the chord, rather than the offending note within the chord being selected. This code can be found here, `https://gitlab.com/dprl/modified-musicdiff`, and the modified CRNN code can be found here, `https://gitlab.com/dprl/e2e-pianoform-high-entropy-loss`.

The control variables for all experiments consist of the model hyperparameters that do not directly impact the loss function. These variables remained constant throughout all runs unless otherwise stated and include a batch size of 1, an initial learning rate of 0.0001 using the Adam optimizer, and a patience of 5 evaluated on the symbol error rate with a minimum delta of 0.01. The maximum number of epochs is set to 10,000 as an arbitrarily high number such that only the early stopping criteria can signal the trainer to stop running. With the exception of the stability experiment detailed below, all experiments were conducted using a set seed. All experiments were run on the same model architecture, and on one or more NVIDIA GeForce RTX 2080 Ti's with 11264MiB memory. No more than four of these cards were used in parallel to train a single model.

The training and validation sets are the same and contain 46,081 undistorted samples from the GrandStaff dataset stored in JPG. There is no separate validation set for GrandStaff, and one was not created for these experiments in order to have the baseline experiments match as closely as possible to that of Ríos-Vila et al. [31]. The test set is a separate set of 7,800 samples not included in the training/validation set and consists of all the un-transposed piano snippets [31].

The independent variable(s) for each experiment will be discussed on a per-experiment basis and largely (though not entirely) consist of a modification to the baseline CTC loss function. The dependent variables are the error metrics and number of epochs discussed above. Additionally, any trends noticed in the types of errors made by the models will be provided based on the analysis of the diffs provided by *musicdiff*. These diffs can be obtained by running the music visualization script in the provided code repository.

4.3 Stability and Control Experiment

As discussed in the PyTorch documentation[3], their implementation of CTC loss is subject to a degree of variability. Although we did try to resolve this, we were ultimately unable to get PyTorch's CTC loss function to behave in a deterministic way. As such, we wanted to establish how stable this implementation really was, and make sure that any variability that we observe in our actual loss experiments is not happening due to variance in the method used to calculate the loss. Additionally, the implementation from Ríos-Vila et al. [31] did not use a set seed, and so we wanted to ensure that the model itself was stable. Finally, we wanted to use these experiments to also establish a baseline understanding on how the model performs when the loss is set to CTC.

Our hypothesis is that PyTorch's implementation, though non-deterministic, will produce results close enough to one another for the purposes of our loss function experiments, such that differences in those results will be substantive and not because of non-determinism in the implementation. It is worth noting that to do a statistically robust test would require hundreds if not

---

[3] https://docs.pytorch.org/docs/main/generated/torch.nn.CTCLoss.html

thousands of runs, which is beyond the scope of our work.

4.3.1 Design

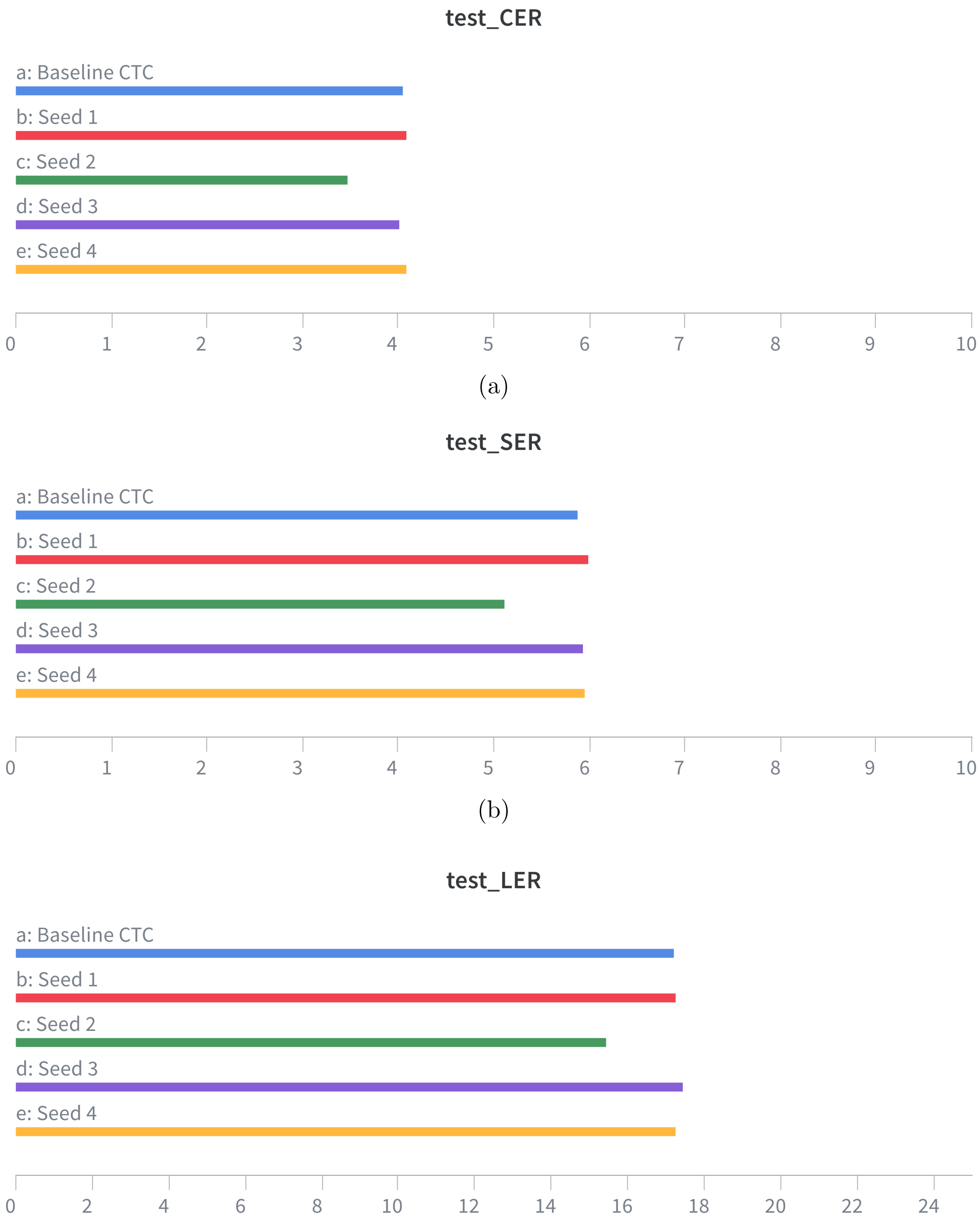
The independent variables for this experiment were the seeds used to initialize the weights of the model and other random number generators (such as those that determine the processing order of training samples). To our understanding, this seed has no bearing on the non-determinism of PyTorch's CTC. Four seeds were selected at random, and an addition seed was chosen to serve as the baseline for all other seeds to compare against. The dependent variables are the metrics listed above. Each of the four random seeds was assigned to one 2080 card and run simultaneously. The baseline seed was trained using four 2080 cards in parallel.

4.3.2 Results

Seed	**CER %**	**SER %**	**LER %**	**No. Epochs**
Baseline	4.05014	5.86764	17.23378	32
Seed 1	4.08220	5.98617	17.28009	44
Seed 2	3.47855	5.12068	15.41811	39
Seed 3	4.00907	5.93292	17.43519	36
Seed 4	4.08103	5.95251	17.26580	35

Table 4.1: Stability test results.

Table 4.1 shows the results of the experiment, computed on the aforementioned evaluation protocols. Additionally, Fig. 4.1 show all of the same error metrics in bar-graph form. We observe that compared to the baseline, seeds 1, 3, and 4 had less than a 0.05%, 0.70%, and 0.25% difference in the CER, SER, and LER respectively. Seed 2 however did outperform all other seeds, having the lowest error rates out of the four random seeds and the baseline. This is especially the case with the LER where seed 2 made 1.81% fewer errors when compared to the baseline. We additionally



(a)

(b)

(c)

Figure 4.1: The metrics CER (a), SER (b), and LER (c) for the baseline and all four random seeds shown across three bar graphs. This mirrors the results found in Table 4.1. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent.

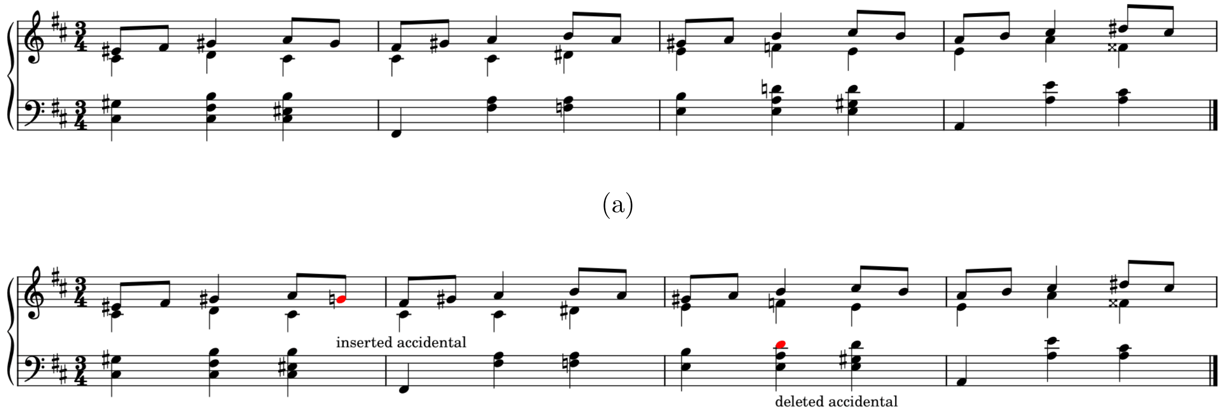
observe that all random seeds took more epochs to reach convergence than the baseline. This is where the biggest differences between the seeds can be observed, where seed 1 ran for 37.5% more epochs than the baseline.

4.3.3 Discussion

From this, we can conclude that certain seeds do indeed outperform one another, as demonstrated by seed 2. This is most apparent in the LER, but is present with the CER and SER as well. This is likely the result of random chance, where the weight initialization performed by seed 2 was better than that of seeds 1, 3, 4, and the baseline. Furthermore, the number of epochs does not appear to have much correlation to the ultimate error rates because similar error rates were achieved with greatly different convergence times.

This showed that set seed needed to be used in order to ensure that variations in the error rates produced by our experiments were the result of actual differences, not because of differences in random weight initialization. From this, we continued using the same seed that was used in the "baseline" run in all future experiments, both since the error rates produced are roughly representative of the other seeds and because it had the lowest number of epochs to reach convergence. This will be the result that subsequent experiments are compared against when baseline metrics for CTC loss are needed.

We also analyze the kinds of errors that were common for the baseline model to make, as a point of future reference. By far, the most common kind of transcription mistake was incorrect accidentals, specifically either those derived from the key sign or those derived from an accidental note of the same pitch earlier in the bar. An example rendering of a transcription performed by the model is seen in Fig. 4.2.



(a)

(b)

Figure 4.2: The ground-truth (a) and baseline CTC (b) final model output of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is.

The reason for these kinds of errors can be ascertained by considering the visual information contained with the symbol itself. Consider the second G♯ seen in measure one of Fig. 4.2a. If the first G♯ was instead just a G or some other note, then what used to be the second G♯ would now just be a G. In other words, whether or not this specific note is a G or G♯ is context dependent and requires coordination from timesteps earlier on in the sequence for this context to be effectively detected. It appears that such coordination did not occur when the baseline model was transcribing this score, as seen by the G natural in measure one of Fig. 4.2b, transcribed simply as *g* in the output Kern, as seen in Fig. 4.3. Although BLSTMs are able to capture context by selectively retaining both information earlier in the sequence at later points in the sequence as well as later information earlier in the sequence, it appears that there are certain, albeit somewhat predictable, cases where OMR transcription fails due to the recurrent cells forgetting the context of key signs and note accidentals within the same measure.

```
**kern         **kern
*              *^
*clefF4        *clefG2        *clefG2
*k[f#c#]       *k[f#c#]       *k[f#c#]
*M3/4          *M3/4          *M3/4
=-             =-             =-
4C# 4G#        8e#L           4c#
.              8f#J           .
4C# 4F# 4B     4g#            4d
4C# 4E# 4B     8aL            4c#
.              8g#J           .
=              =              =
4FF#           8f#L           4c#
.              8g#J           .
4F# 4A         4a             4c#
4Fn 4A         8bL            4d#
.              8aJ            .
=              =              =
4E 4B          8g#L           4e
.              8aJ            .
4E 4A 4dn      4b             4fn
4E 4G# 4d      8cc#L          4e
.              8bJ            .
=              =              =
4AA            8aL            4e
.              8bJ            .
4A 4e          4cc#           4a
4A 4c#         8dd#L          4f##
.              8cc#J          .
=              =              =
*              *v             *v
*-             *-
```

(a)

```
**kern         **kern
*              *^
*clefF4        *clefG2        *clefG2
*k[f#c#]       *k[f#c#]       *k[f#c#]
*M3/4          *M3/4          *M3/4
=-             =-             =-
4C# 4G#        8e#L           4c#
.              8f#J           .
4C# 4F# 4B     4g#            4d
4C# 4E# 4B     8aL            4c#
.              8gJ            .
=              =              =
4FF#           8f#L           4c#
.              8g#J           .
4F# 4A         4a             4c#
4F 4A          8bL            4d#
.              8aJ            .
=              =              =
4E 4B          8g#L           4e
.              8aJ            .
4E 4A 4d       4b             4f
4E 4G# 4d      8cc#L          4e
.              8bJ            .
=              =              =
4AA            8aL            4e
.              8bJ            .
4A 4e          4cc#           4a
4A 4c#         8dd#L          4f##
.              8cc#J          .
=              =              =
*              *v             *v
*-             *-
```

(b)

Figure 4.3: The ground truth (a) and baseline CTC (b) transcriptions used to render the scores in Fig. 4.2. The errors highlighted in red on the right reflect the errors highlighted in red within the rendering. Note that there are other transcription errors present on the right, where certain symbols (such as the *f* on the same line as the mis-transcribed *d*) are missing an `n` modifier, representing a ♮. However, because of the key sign, these transcription errors are not shown in the rendering since having the natural and not having the natural are treated the same in this case.

4.4 FocalCTC Hyperparameter Search

This experiment was to test the conditions in Feng et al. [11] but in the context of OMR rather than OCR. That is to say, we wanted to see for what values of α and γ from Eq. 3.7 is the CER, SER, and LER minimized. This was implemented using Eq. 3.9 with PyTorch's CTC as the calculation for $\mathbf{L}_{\text{CTC}}$. Our hypothesis is that the lowest error rates will be achieved when $\alpha = 0.50$ and $\gamma = 1.0$. A value of $\alpha = 0.50$ is hypothesized to do the best since lower values of α are not very punishing and will lead to smaller values of $\hat{p}_t$ to not result in great enough loss values. Furthermore, greater values of α will cause the loss incurred of common OMR symbols (of which $\hat{p}_t$ is high) to be great, causing the model to overfit to those same symbols rather than trying to learn the rarer ones [11]. Moreover, a value of $\gamma = 1.0$ is predicted to do the best since it allows for probabilities further from 1 of $\hat{p}_t$ to result in low loss values but not as much as $\gamma = 2.0$, where even probabilities as low as 0.80 result in losses less than 0.01 for the values of α we are considering.

4.4.1 Design

The independent variables for this experiment were the values of α and γ used in FocalCTC loss. The values tested are as follows: $\alpha = [0.25, 0.50, 0.75, 0.99]$ and $\gamma = [0.5, 1.0, 2.0]$ [11]. Each condition combination was tested, resulting in twelve total. The dependent variables are the metrics listed above. Each of the four 2080's was assigned an α value, and then performed three runs in total – one for each value of γ. A modified version of PyTorch's CTC loss function was used as the basis of implementation for FocalCTC.

4.4.2 Results

CER%, SER%, LER%												
	$\alpha = 0.25$			$\alpha = 0.50$			$\alpha = 0.75$			$\alpha = 0.99$		
$\gamma = 0.5$	4.24846	6.28695	18.75513	3.91863	**5.80615**	17.73719	**3.86134**	5.86597	**17.58368**	4.42040	6.58667	19.56450
$\gamma = 1.0$	4.31349	6.38923	19.10212	4.45195	6.60709	19.76513	4.20023	5.83715	18.89197	3.86476	5.83715	17.80494
$\gamma = 2.0$	4.71595	7.00882	21.19543	4.52268	6.74115	20.51257	4.93676	7.33675	21.83039	4.65502	6.90181	21.12132

Table 4.2: Summary of the CER, SER, and LER on the GrandStaff test set for the FocalCTC experiment. The smallest result for each is bolded.

CER%, SER%, LER%			
	$\alpha = 1.00$		
$\gamma = 0$	4.05014	5.86764	17.23378

Table 4.3: Summary of the CER, SER, and LER for the baseline CTC results present in Table 4.1, equivalent to FocalCTC when $\alpha = 1.00$ and $\gamma = 0$.

No. Epochs					
	$\alpha = 0.25$	$\alpha = 0.50$	$\alpha = 0.75$	$\alpha = 0.99$	$\alpha = 1.00$
$\gamma = 0$	N/A	N/A	N/A	N/A	TODO
$\gamma = 0.5$	44	51	58	31	N/A
$\gamma = 1.0$	41	41	38	52	N/A
$\gamma = 2.0$	51	58	45	42	N/A

Table 4.4: Summary of the number of epochs it took each FocalCTC condition for early stopping to occur.

Results for this experiment are presented in Table 4.2 and Table 4.4. Additionally, the baseline CTC results from Table 4.1 are presented in terms of FocalCTC parameters $\alpha = 1.00$ and $\gamma = 0$ in Table 4.3 and Table 4.4. Mesh plot and bar graph visualizations containing the CER, SER, and LER for all FocalCTC conditions as well as the baseline CTC run are shown in Fig. 4.4-4.8. We first observe that the shape of all three error rate meshes are all roughly the same. This means that for all conditions tested, there were few cases of a specific error rate increasing or decreasing while the other error rates did not do the same. That is to say, the error rates roughly increased

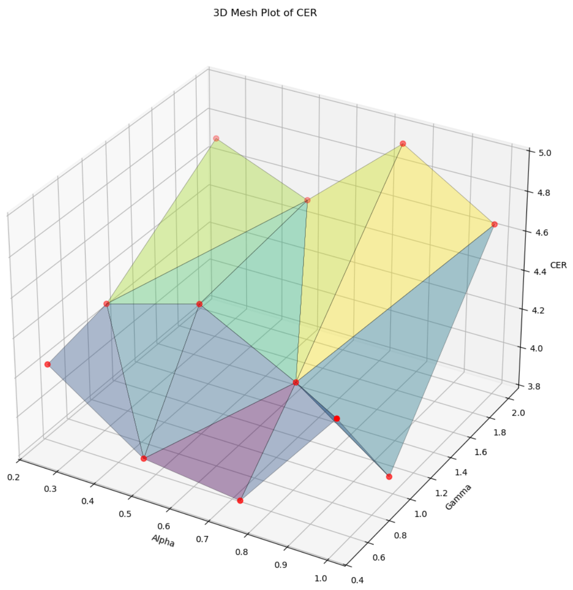


Figure 4.4: A 3D mesh plot of the CER, showing all twelve conditions run and what their corresponding CER was.

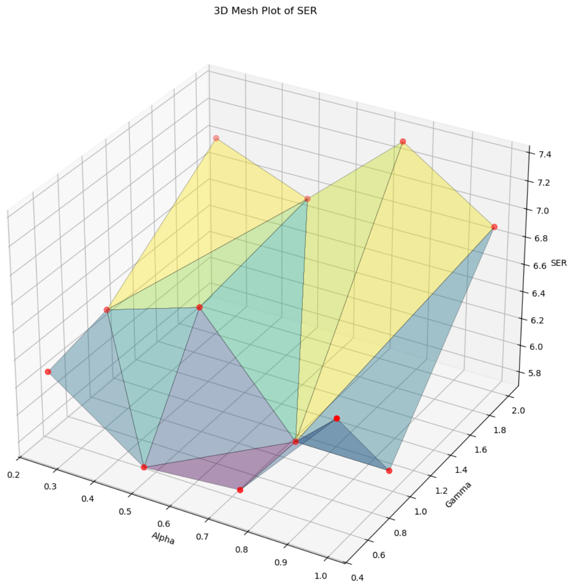


Figure 4.5: A 3D mesh plot of the SER, showing all twelve conditions run and what their corresponding SER was.

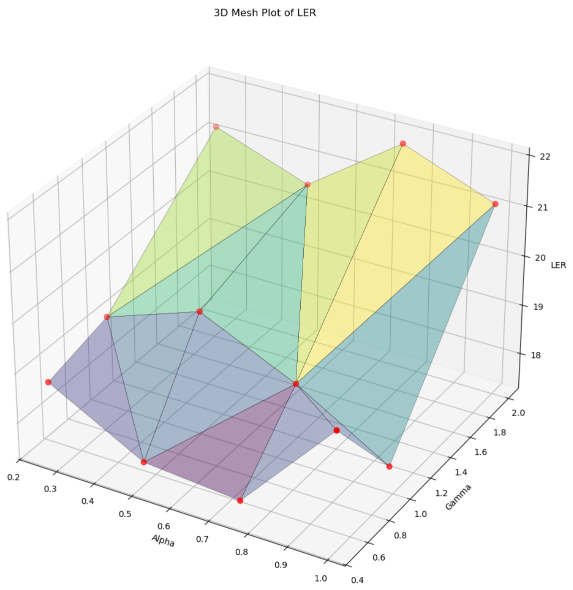


Figure 4.6: A 3D mesh plot of the LER, showing all twelve conditions run and what their corresponding LER was.

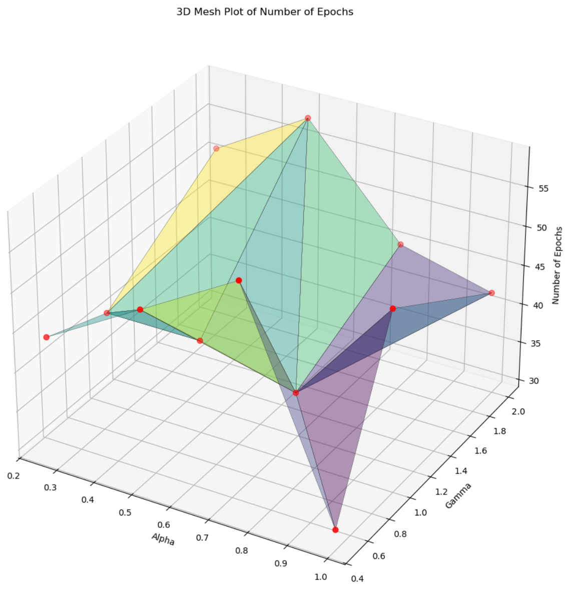
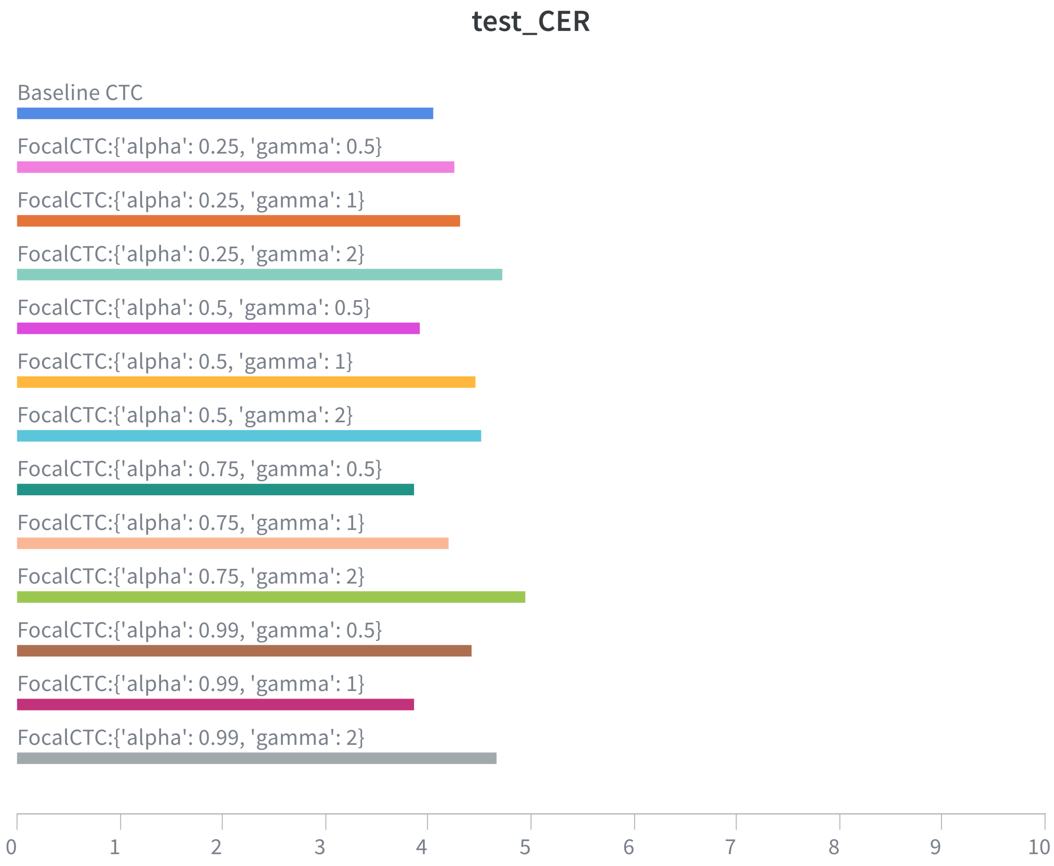
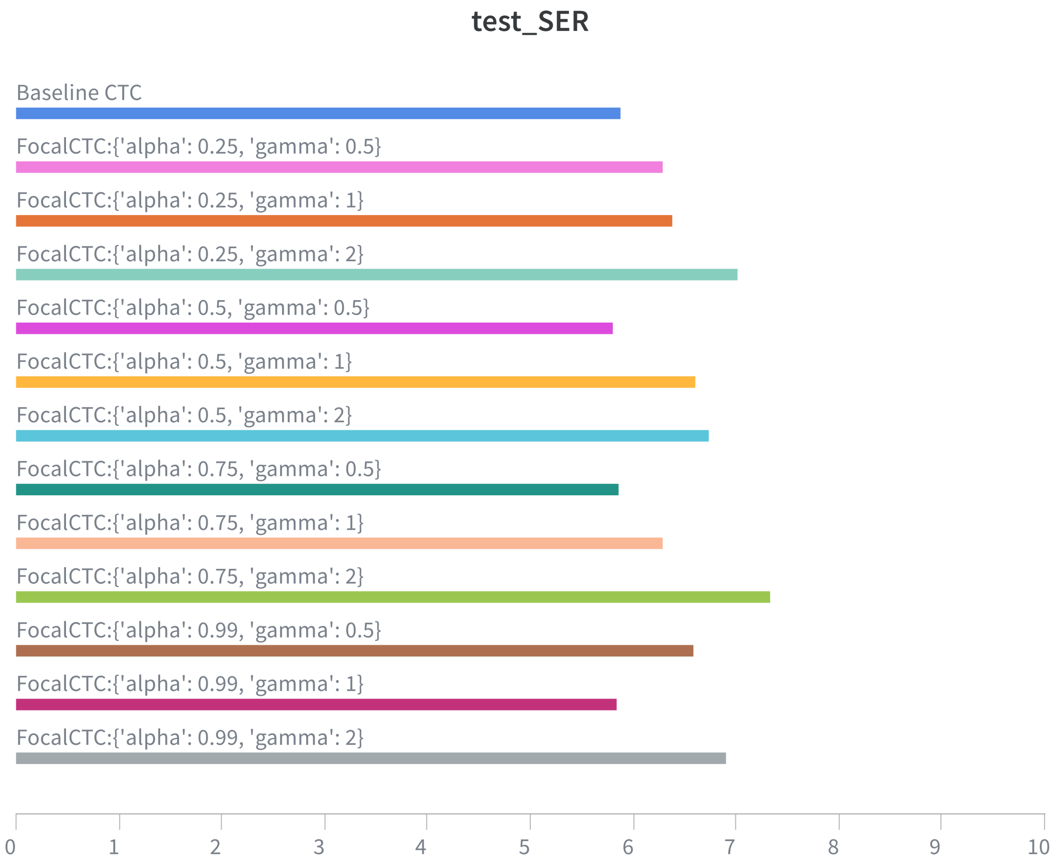


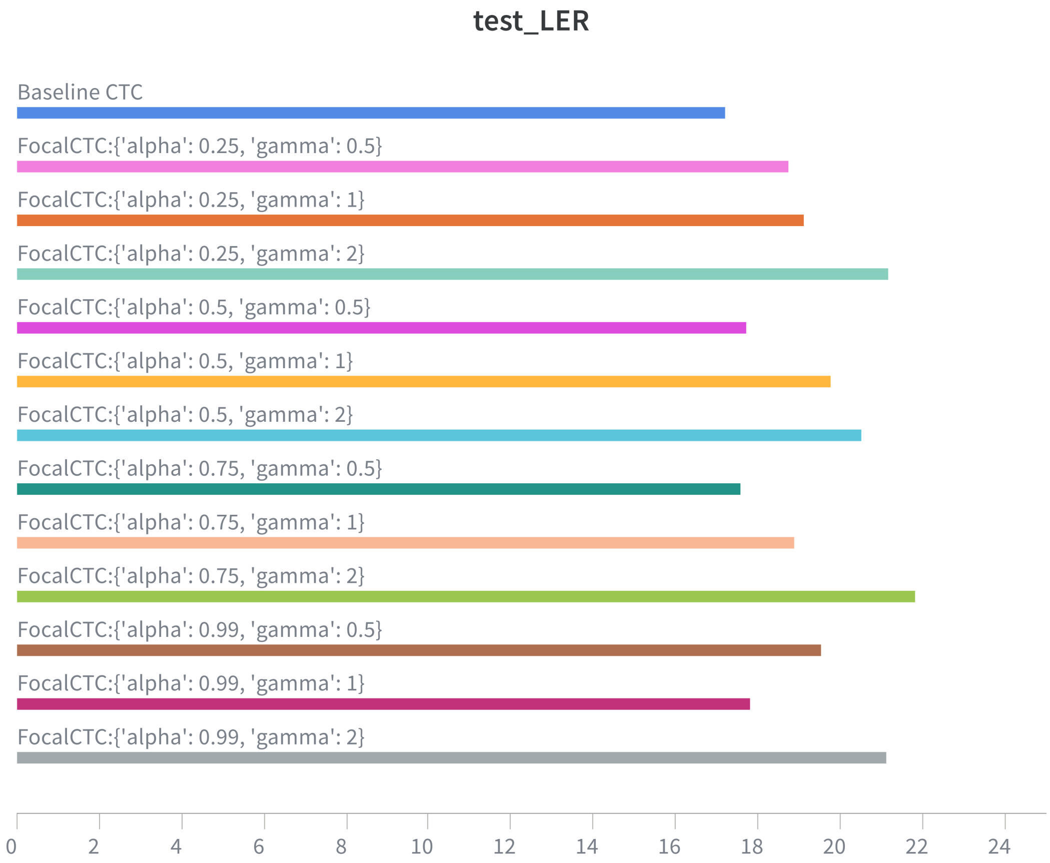
Figure 4.7: A 3D mesh plot of the convergence time, showing all twelve conditions run and what their corresponding number of epochs was.


test_CER
Baseline CTC
FocalCTC:{'alpha': 0.25, 'gamma': 0.5}
FocalCTC:{'alpha': 0.25, 'gamma': 1}
FocalCTC:{'alpha': 0.25, 'gamma': 2}
FocalCTC:{'alpha': 0.5, 'gamma': 0.5}
FocalCTC:{'alpha': 0.5, 'gamma': 1}
FocalCTC:{'alpha': 0.5, 'gamma': 2}
FocalCTC:{'alpha': 0.75, 'gamma': 0.5}
FocalCTC:{'alpha': 0.75, 'gamma': 1}
FocalCTC:{'alpha': 0.75, 'gamma': 2}
FocalCTC:{'alpha': 0.99, 'gamma': 0.5}
FocalCTC:{'alpha': 0.99, 'gamma': 1}
FocalCTC:{'alpha': 0.99, 'gamma': 2}
0
1
2
3
4
5
6
7
8
9
10

(a)


test_SER
Baseline CTC
FocalCTC:{'alpha': 0.25, 'gamma': 0.5}
FocalCTC:{'alpha': 0.25, 'gamma': 1}
FocalCTC:{'alpha': 0.25, 'gamma': 2}
FocalCTC:{'alpha': 0.5, 'gamma': 0.5}
FocalCTC:{'alpha': 0.5, 'gamma': 1}
FocalCTC:{'alpha': 0.5, 'gamma': 2}
FocalCTC:{'alpha': 0.75, 'gamma': 0.5}
FocalCTC:{'alpha': 0.75, 'gamma': 1}
FocalCTC:{'alpha': 0.75, 'gamma': 2}
FocalCTC:{'alpha': 0.99, 'gamma': 0.5}
FocalCTC:{'alpha': 0.99, 'gamma': 1}
FocalCTC:{'alpha': 0.99, 'gamma': 2}
0
1
2
3
4
5
6
7
8
9
10

(b)



(c)

Figure 4.8: The metrics CER (a), SER (b), and LER (c) for the baseline and all twelve FocalCTC conditions. This mirrors the results found in Table 4.2. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent.

and decreased in unison. The primary exception to this is going from $\alpha = 0.50$ to $\alpha = 0.75$ at $\gamma = 0.5$, where both the CER and LER decrease but the SER increases. There is also the case of $\alpha = .75$ to $\alpha = 0.99$ at $\gamma = 1.0$ where the CER and LER once again decrease, but the SER stays constant.

We then observe what happens to the error rate as α stays constant but γ increases. When $\alpha = 0.25, 0.50, 0.75$, the greatest error rates occurred when $\gamma = 2.0$, and the lowest error rates occurred when $\gamma = 0.50$, implying a direct correlation between an increase in γ and greater error. However, for $\alpha = .99$, $\gamma = 1.0$ achieved the lowest error rates, seen in the dips present in the meshes in Fig. 4.4, 4.5, and 4.6 at that point.

We additionally observe what happens to the error rate as γ stays constant but α increases. At $\gamma = 0.5$, the lowest values achieved for the CER and LER were at $\alpha = 0.75$, and for SER was at $\alpha = 0.50$. These were in fact the lowest error rates achieved for this experiment. Furthermore, the CER and SER were smaller than that of the baseline experiment (seen in Fig. 4.8a and Fig. 4.8b) and was a marginal improvement of 0.19% and 0.06% respectively. However, the lowest LER achieved in the FocalCTC experiment was greater (seen in Fig. 4.8c) than that of baseline CTC, showing an error increase of 0.35%, indicating that although there were fewer errors overall, they were spread out over more lines. For $\gamma = 1.0$ the lowest values for all three metrics were achieved at $\alpha = 0.99$, although it is worth noting that the same value was achieved for the SER at $\alpha = 0.75$. Finally, for $\gamma = 2.0$, all three error metrics achieved their lowest point at $\alpha = 0.50$.

Although all models did eventually converge, there does not appear to be any long-range trend between α nor γ and the number of epochs it takes for the model to converge. This observation comes from the fact that there is no consistency between what happens to the number of epochs (i.e. if it increases, decreases, or stays the same) as α and γ increase or decrease – something which can be verified visually by seeing the mesh plot in Fig. 4.7.

4.4.3 Discussion

We first discuss the trend of the error as α stays constant but γ increases. We posit the that the reason the CER, SER, and LER all increase is because the model learns to put too much emphasis on learning less common symbols such as accidentals and not enough in learning more fundamental ones, such as notes. This is exemplified in Fig. 4.9, where we can see that when α is fixed to 0.25 and γ is allowed to vary, the corresponding models make more mistakes when identifying notes when $\gamma = 2.0$ as compared to when $\gamma = 0.5$.

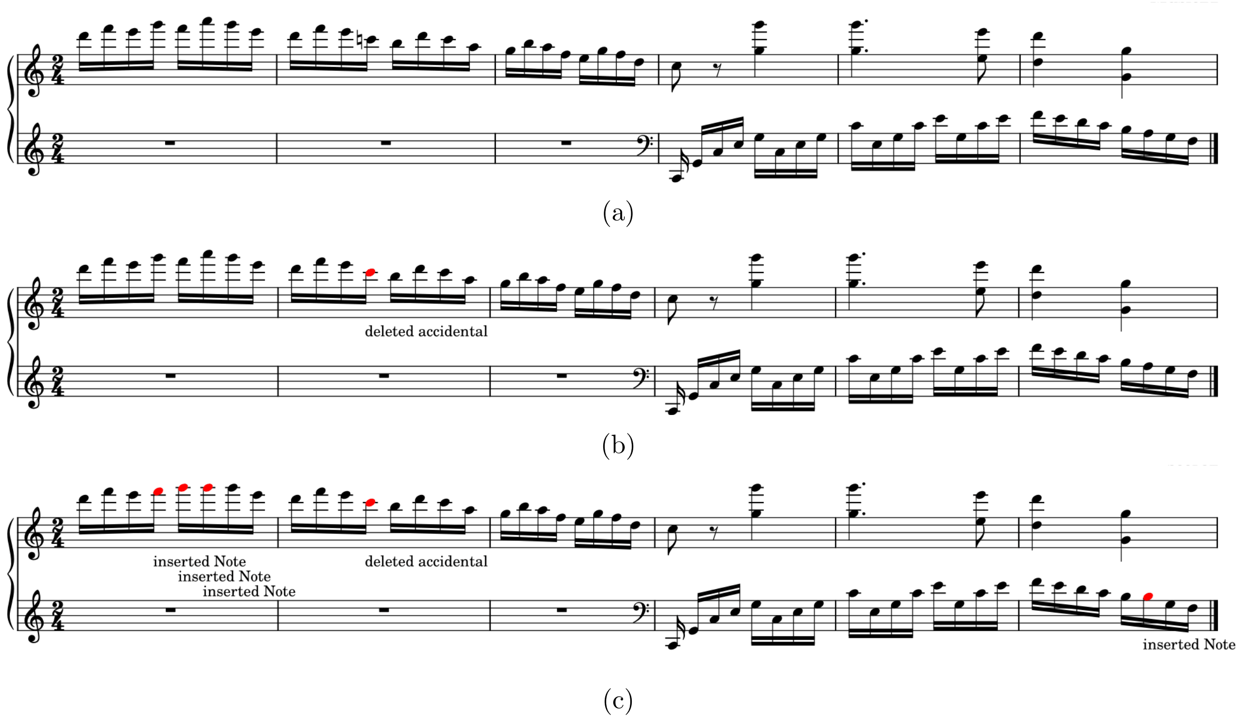


Figure 4.9: The ground truth (a), $\alpha = 0.25, \gamma = 0.5$ (b), and $\alpha = 0.25, \gamma = 2$ (c) final model output transcriptions of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is.

This does beg the question as to why the $\gamma = 0.5$ model perfectly identified the clef change in the third measure, despite clef changes being much rarer than that of accidental. In fact, clef changes were actually quite recognizable by nearly all models and rarely got misidentified despite their rarity. We believe this occurs due to the distinct visual nature of each clef – all three (F, G, and C) do not resemble each other in almost any way and there is no other notation that resembles them either. This also appears to be the case with ties and slurs, since basically all models were able to identify them easily. From this, we can conclude that greater values of γ appear to misclassify symbols that, despite their commonality, were less visually distinct than that of other symbols when compared to lower γ models.

It is worth noting that there appears to be a pocket of low error at $\alpha = .99, \gamma = 1.0$, where at this α value, $\gamma = 1.0$ outperformed both alternative γ values. It is not surprising that it outperformed $\gamma = 2.0$ for the reasons mentioned above, but for it to achieve lower error than $\gamma = 0.5$ suggests that a finer grid search of this space may be warranted. This dip does correspond to an increase in accuracy found in Feng, et al. [11], suggesting that it may be inherent to FocalCTC itself.

There is also the matter of what effect increasing α has if γ stays constant. This appears to have more variability on the error rates on a per-γ basis compared to when α is kept static. No consistent trend was noticed in the kinds of errors made by the models as α increased or decreased and γ stayed fixed for any of the tested values of γ. However, the two exceptions where increasing α increased the CER and LER but not the SER does seem to suggest that as α increases, the errors tend to spread out over more lines and that there are more errors per symbol, rather than the number of symbols with errors increasing.

Finally, we consider the reason as to why there appears to be little to no correlation between increases in α or γ and the number of epochs it took for the model to converge. We suspect this is simply due to the non-determinism and randomness of PyTorch's CTC implementation rather than any inherent property of FocalCTC itself.

Based on these results, we conclude that part of our initial hypothesis was incorrect, since the lowest error metrics were achieved in all cases when $\gamma = 0.5$. The reason $\gamma = 1.0$ did not give the lowest error is likely the same reason initially given for why $\gamma = 2.0$ would not give the lowest error: greater values of γ allow for lower probability values to result in a near-zero loss, and as such, it is likely that $\gamma = 1.0$ simply also exhibited this property. Moreover, we were partially correct about $\alpha = 0.50$ being the smallest error that would be achieved, since this was the case for the SER, though not the CER nor LER as previously mentioned. This implies that in the case of OMR, the loss needs to be more punishing than initially thought for the model to achieve the best results.

4.5 Direct Entropy Modification Experiments

Our final set of experiments were to directly test the methods for SR-CTC [41] and EnCTC [22] in OMR. Although FocalCTC does increase the entropy of a prior probability distribution, it does so as more of a consequence of what it actually is trying to achieve – that being to have greater loss be attributed to tokens found in samples it misclassifies, rather than modify the entropy directly. Contrast this with SR-CTC and EnCTC, which are far more direct in their incorporation of entropy.

SR-CTC was implemented using Eqs. 3.10 - 3.12, using the direct model probability output $\hat{y}$ as well as PyTorch's CTC as the calculation for $\mathbf{L}_{\text{CTC}}$. Additionally, β was set to 0.2. Since EnCTC requires values to be kept track of on a per-path basis (Eq. 3.13 and 3.14), PyTorch's

CTC implementation could not be used. Instead, a custom dynamic programming array (like that described in Section 3.1.1) is used to store both the CTC and path entropy values [22]. β was set to 0.2 for this loss as well. Our hypothesis is that out of all the experiments run, the one that will perform the best is EnCTC, since it most directly penalizes the model for not having high entropy.

4.5.1 Design

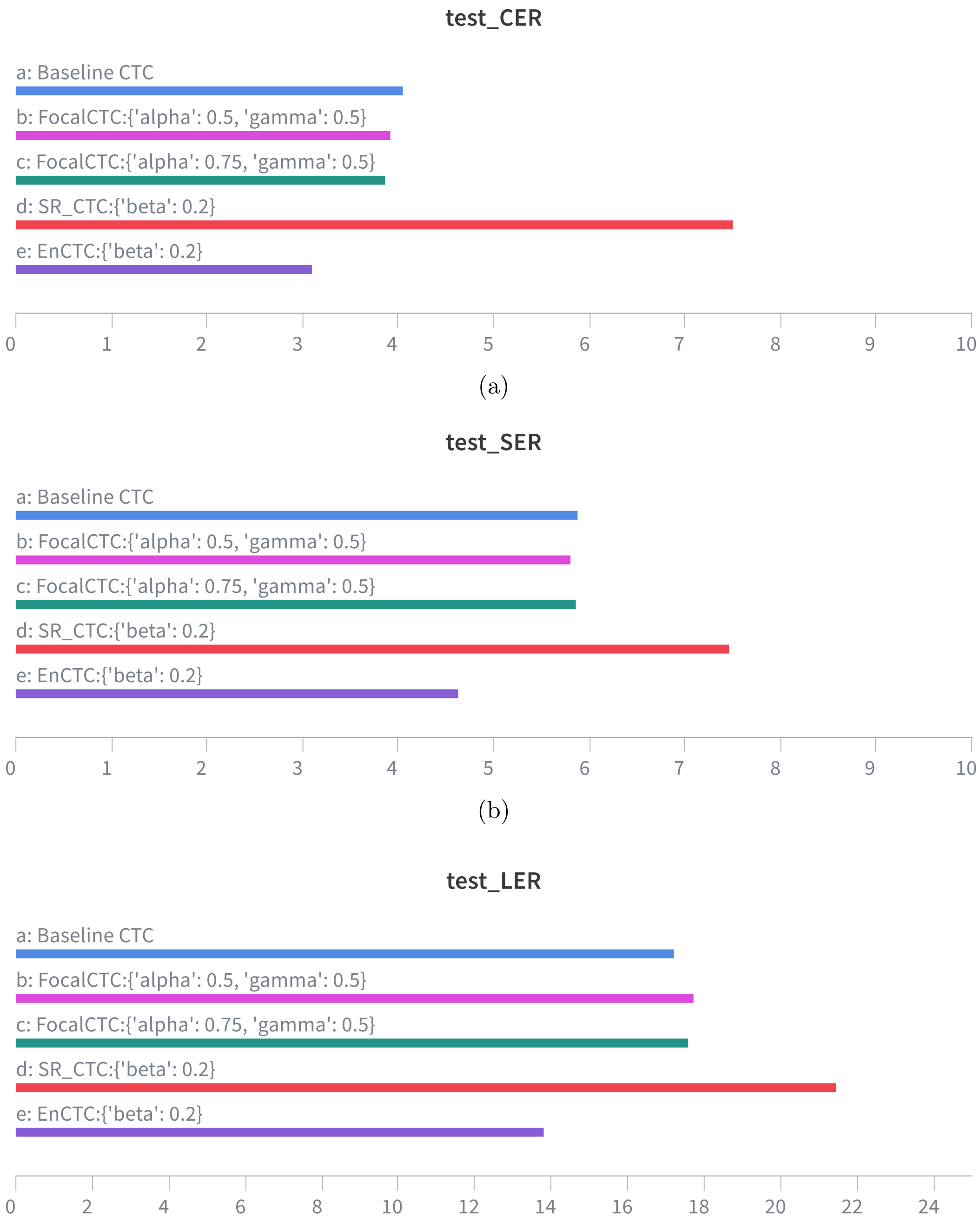
The independent variables for this experiment were the loss functions used, being implementations of SR-CTC [41] and EnCTC [22]. The dependent variables are the metrics listed above. Each experiment was run on four 2080 cards in parallel.

4.5.2 Results

Condition	**CER %**	**SER %**	**LER %**	**No. Epochs**
Baseline	4.05014	5.86764	17.23378	32
FocalCTC $\alpha = 0.50, \gamma = 0.5$	3.91863	5.80615	17.73719	51
FocalCTC $\alpha = 0.75, \gamma = 0.5$	3.86134	5.86597	17.58368	58
SR-CTC	7.49828	7.47131	21.45508	36
EnCTC	**3.08860**	**4.62463**	**13.81206**	53

Table 4.5: Results for both SR-CTC and EnCTC presented alongside the baseline results as well as the conditions from FocalCTC that minimized each error metric. The smallest result for each error metric is bolded.

Results for this experiment compared against the baseline as well as the conditions from FocalCTC that minimized each error metric are presented in Table 4.5. Bar chart visualization containing the CER, SER, and LER for the same runs in Table 4.5 are presented in 4.10. We first observe that SR-CTC underperformed against the other runs presented on all three error metrics. In fact, SR-CTC performed worse on both CER and SER than the lowest scoring condition of FocalCTC in both metrics, being $\alpha = 0.75, \gamma = 2.0$. For LER, although $\alpha = 0.75, \gamma = 2.0$ did achieve a greater LER, there was less than a 0.40% difference in the two. These results are

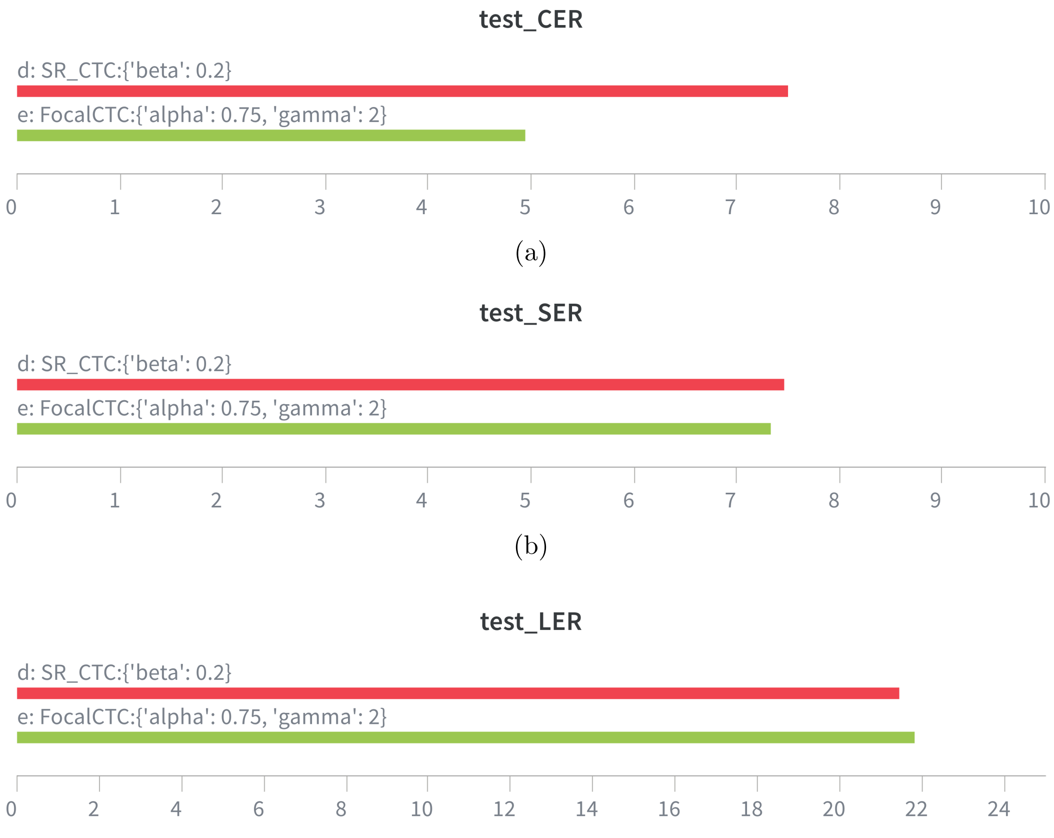


(a)

(b)

(c)

Figure 4.10: The metrics CER (a), SER (b), and LER (c) for the baseline, conditions from FocalCTC that minimized each error metric, SR-CTC, and EnCTC. This mirrors the results found in Table 4.5. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent.



(a)

(b)

(c)

Figure 4.11: Bar graphs showing the CER (a), SER (b), LER (c) of SR-CTC and the FocalCTC results of $\alpha = 0.75, \gamma = 2.0$. Each bar graph is zoomed in instead of being shown from 0 to 100 in order to make the difference more visually apparent.

visualized in Fig. 4.11.

Conversely, we observe that EnCTC produced lower CER, SER, and LER scores than all other losses presented. The most stark improvement can be seen in the LER, where EnCTC had errors in 3.4% fewer lines than that of the baseline CTC, which otherwise had the lowest LER achieved.

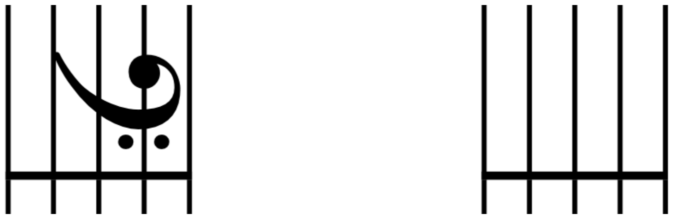
The number of epochs presented for EnCTC represent the number when the snapshot was taken. The convergence time for both SR-CTC and EnCTC were longer than that of the baseline, but SR-CRC was less than the smallest error runs of FocalCTC.

4.5.3 Discussion

We first address the poor performance of SR-CTC. It is suspected that the reason SR-CTC did so poorly is due to the two-dimensional nature of OMR. SR-CTC was designed to process timesteps that naturally occur sequentially and in a single dimension such that as that of ASR [41]. In order to process the set of two-dimensional feature maps in OMR, we must flatten along either the original vertical or horizontal axis, as previously shown in Fig. 3.2. This flattened feature map is our analogous one-dimensional sequence.

However, this presents a problem for SR-CTC – by essentially flattening the feature map in this way, we introduce discontinuities within the canonical output sequence, where timesteps that are placed near each other may not necessarily correspond to cells in the original input image that are near each other. When the smoothing kernel K is then applied, smoothing occurs to non-spatially-adjacent cells. This causes the KL-divergence to optimize for otherwise non-spatially-adjacent timesteps to have more influence on one another than they perhaps should. For example, cells $v_{1,7}$ and $v_{2,1}$ from Fig. 3.2 would appear adjacent in the flattened feature map, and thus have their probabilities for a label k convolved together as a result of smoothing, despite their spatial adjacency not being present in the original two-dimensional feature map.

This, in turn, likely causes the model to assign greater probabilities to tokens that may be present in the previous or next timestep that are radically different to what is present in the current timestep due to the current timestep being on the edge of a discontinuity. Smoothing then can cause the otherwise correct and most likely token to no longer be the most likely, leading to



(a) An example slice of a score, rotated by 90°. It contains a bass clef on the bottom staff.

```
*clefF4 *       *

=       =       =
```

(b) The correct ground truth transcription of (a).

```
*clefF4 *=

=       =       =
```

(c) An incorrect transcription of (a) provided by the SR-CTC model.

Figure 4.12: An example of SR-CTC incorrectly predicting a token due to discontinuous timesteps having influence on one another. Three spines are shown as the original input image had two voices in the top staff. The `*` token represents a spine placeholder, which in this case simply indicates no change from the previous clef and a new clef does not need to be rendered.

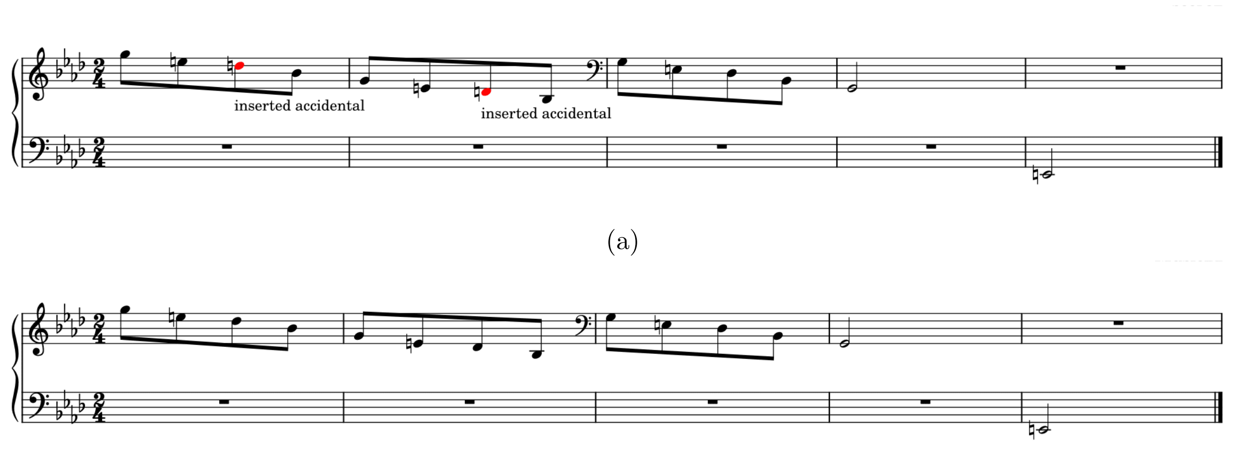
errors. This leads to many of the generated Kern files to be syntactically invalid, meaning they were not able to be rendered into a score that could be viewed. However, this problem can still be clearly seen when we look at the Kern produced by the model with SR-CTC as its loss – as shown in Fig. 4.12. In this figure, we consider why there exists a barline token before the newline token.

Suppose the model got to a timestep where its prediction would be to output a newline, followed by a barline. Due to smoothing, the probability of the newline gets smeared in such a way that it no longer is the most likely token. Instead, the barline from the next timestep was captured by the smoothing and thus a barline is output instead of the newline, causing a barline to appear after the asterisk instead of the newline. This is what is conjectured to have occurred in Fig. 4.12c, though it is worth noting that other smoothing artifacts are likely present as well. If this experiment were to be conducted again, the two dimensional nature of the OMR problem should be taken into consideration either by altering the kernel to be applied in such a way that accounts for the discontinuities (such as not allowing non-adjacent timesteps to interact via convolution with K), or by making the kernel itself two dimensional.

We then address the performance of EnCTC, which achieved the lowest error rates out of all set seed loss functions. We attribute this performance to the much more direct optimization of entropy due to its mathematical definition being incorporated into the loss function itself as was hypothesized. Moreover, the model EnCTC seemed less susceptible to incorrectly not predicting accidentals that are inferred through the key sign, shown in Fig. 4.13. This is likely due to an increased amount of exploration from the model, meaning that alternative paths not dominated by the blank token were assigned greater probabilities of being selected. This allowed it to better learn the relationship between the initial key sign and subsequent accidentals that match what is within the key sign.

As Fig. 4.14 shows, the misclassification of accidentals is still the most common kind of error present for the EnCTC model as well, including those stemming from previous notes within a measure. However, EnCTC does still demonstrate an advantage over the baseline, as the frequency of these kinds of errors, though present, is lower than that of the baseline.

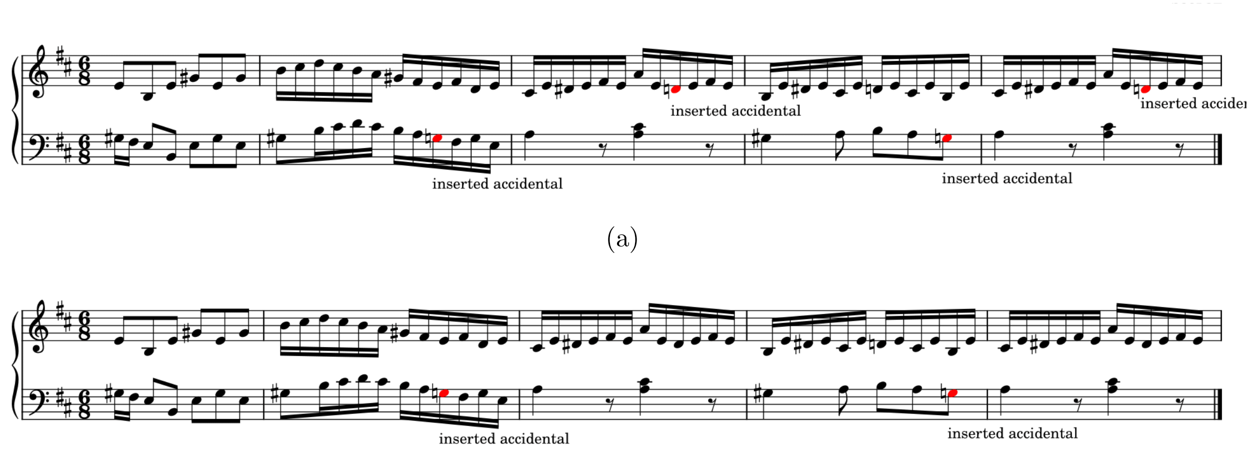
Finally, nothing of note happened with the convergence time of SR-CTC. However, this was not the case for EnCTC. There were several points at which convergence was almost reached, where the model had several epochs in a row of no improvement occurring – the model continued to achieve a lower and lower SER on the validation set. A diagram of this convergence is shown in Fig. 4.15. This suggests that the aforementioned path space of which the entropy is being optimized for is quite large – likely due to the large number of viable paths π that can be used to generate l. This allows the model to continue its exploration of the path space well into training.



(a)

(b)

Figure 4.13: The baseline CTC (a) and EnCTC (b) final model output of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is. Note that the final model output for EnCTC did not produce any errors for this example.



(a)

(b)

Figure 4.14: The baseline CTC (a) and EnCTC (b) final model output of a score from the test set rendered via MuseScore 3. Differences between the ground truth and what each model predicted are annotated via the notehead being red and a text annotation appearing on the score itself detailing what the error is.

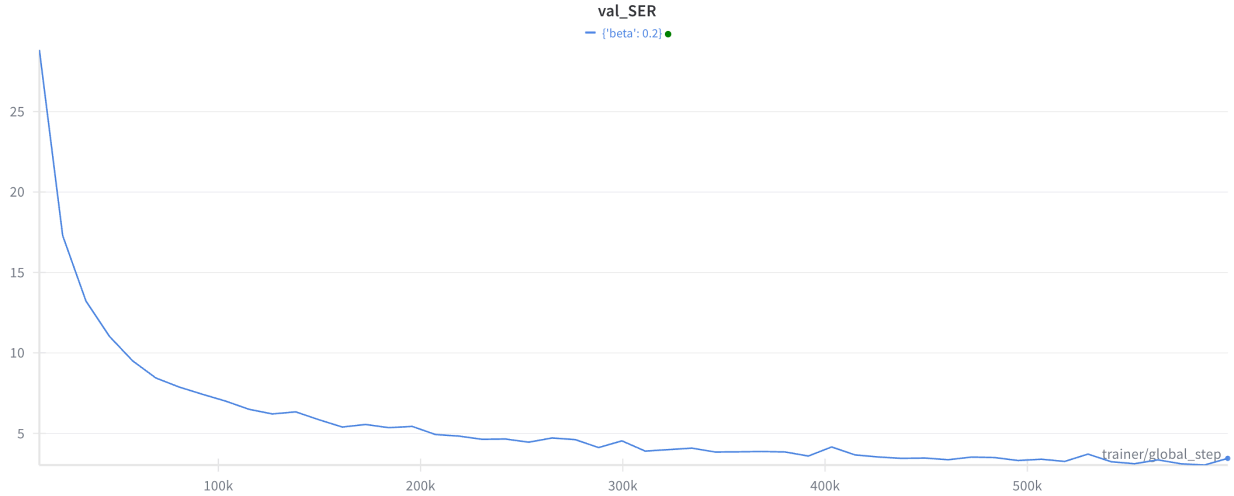


Figure 4.15: The value of the SER metric on the validation set for EnCTC over each training step. Since the model will stop after the SER has not improved in five epochs, this serves as an indication of when the model reached several potential convergence points before continuing to decrease the SER.

Chapter 5

Conclusions

In this work, we examined the effect of high entropy alternatives to the CTC loss function in the field of OMR. These included a hyperparameter grid search of FocalCTC [11] as well as examination of the SR-CTC [41] and EnCTC [22] loss functions. Each of these optimize for high entropy in a different way. For a given value of α and γ, FocalCTC optimizes for increased entropy in each probability distribution $\hat{p}^t$ for each timestep across all tokens by increasing the probability of harder tokens being predicted. SR-CTC optimizes for increased entropy in each prior probability distribution $\hat{p}_k$ for each token across all timesteps by smoothing the estimated prior probability distribution for each token. And EnCTC optimizes for increased entropy in the probability distribution of paths $\pi \in \mathcal{B}^{-1}(l)$ by directly incorporating the entropy of the path space within the loss function itself. Our hypothesis was that loss functions that optimize for high entropy in some way will lead to an alleviation of the spiky distribution problem [8, 15, 20, 22, 41] and thus an decrease in the CER, SER, and LER.

Through our experiments, we found that FocalCTC was able to outperform the baseline CTC loss function on two out of the three error metrics. EnCTC was able to outperform both the baseline CTC and FocalCTC on all three error metrics. SR-CTC, though it has potential, was not able to demonstrate an advantage over using the baseline, though this can be attributed to a flaw in how the loss was attributed given the problem domain. With modification, it is conceivable that a version of SR-CTC could outperform the baseline. Moreover, EnCTC demonstrated that having a large probability space, such as the path space of CTC, and a direct utilization of the mathematical definition of entropy within the loss calculation provides the a much better optimization strategy for OMR compared to baseline CTC. As such, our hypothesis is confirmed as multiple high entropy alternatives to CTC were able to achieve advantageous performance over the baseline CTC implementation.

Regarding future work, there are several experiments that could be run based on the results we achieved. The most pressing would be a re-configuration of SR-CTC such that it works in a two-dimensional setting, which would alleviate the issues of discontinuity between timesteps. Additionally, FocalCTC demonstrated that there may be a local error minimum present around the values $\alpha = 0.75$ to $\alpha = 0.99, \gamma = 1.0$, so a more granular hyperparameter search of this area may reveal more optimal hyperparameters for OMR. A dynamic value of α based on the inverse label frequencies may also be more optimal [11, 21]. Lastly, several loss functions would require a change on the model architecture level in order be to implemented such as Var-CTC [8], but do claim to alleviate the spiky distribution problem. These losses have yet to be tested on OMR and its class imbalance problem, and so experiments doing so would be of great interest.

Bibliography

[1] María Alfaro-Contreras, José M. Iñesta, and Jorge Calvo-Zaragoza. Optical music recognition for homophonic scores with neural networks and synthetic music generation. *International Journal of Multimedia Information Retrieval*, 12(1):12, May 2023.

[2] María Alfaro-Contreras, Antonio Ríos-Vila, Jose J. Valero-Mas, José M. Iñesta, and Jorge Calvo-Zaragoza. Decoupling music notation to improve end-to-end optical music recognition. *Pattern Recognition Letters*, 158:157–163, 2022.

[3] María Alfaro-Contreras and Jose J. Valero-Mas. Exploiting the two-dimensional nature of agnostic music notation for neural optical music recognition. *Applied Sciences*, 11(8):3621, April 2021.

[4] Xingyu Cai, Jiahong Yuan, Yuchen Bian, Guangxu Xun, Jiaji Huang, and Kenneth Church. W-CTC: a connectionist temporal classification loss with wild cards. In *International Conference on Learning Representations*, 2022.

[5] Jorge Calvo-Zaragoza and David Rizo. Camera-primus: Neural end-to-end optical music recognition on realistic monophonic scores. In Emilia Gómez, Xiao Hu, Eric Humphrey, and Emmanouil Benetos, editors, *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, pages 248–255, 2018.

[6] Jorge Calvo-Zaragoza and David Rizo. End-to-end neural optical music recognition of monophonic scores. *Applied Sciences*, 8(4):606, April 2018.

[7] Jorge Calvo-Zaragoza, David Rizo, and José Manuel Iñesta Quereda. Two (note) heads are better than one: Pen-based multimodal interaction with music scores. In *International Society for Music Information Retrieval Conference*, 2016.

[8] Linlin Chao, Jingdong Chen, and Wei Chu. *Variational Connectionist Temporal Classification*, pages 460–476. Springer International Publishing, 2020.

[9] Kwon-Young Choi, Bertrand Coüasnon, Yann Ricquebourg, and Richard Zanibbi. Cnn-based accidental detection in dense printed piano scores. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 473–480, 2019.

[10] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. *Unified language model pre-training for natural language understanding and generation*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[11] Xinjie Feng, Hongxun Yao, and Shengping Zhang. Focal ctc loss for chinese optical character recognition on unbalanced datasets. *Complexity*, 2019(1), January 2019.

[12] Eliseo Fuentes-Martínez, Antonio Ríos-Vila, Juan C. Martinez-Sevilla, David Rizo, and Jorge Calvo-Zaragoza. Aligned music notation and lyrics transcription. *Pattern Recognition*, 170:112094, 2026.

[13] Alejandro Galan-Cuenca, Jose J. Valero-Mas, Juan C. Martinez-Sevilla, Antonio Hidalgo-Centeno, Antonio Pertusa, and Jorge Calvo-Zaragoza. Muscat: A multimodal music collection

for automatic transcription of real recordings and image scores. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, pages 583–591. ACM, October 2024.

[14] Antonio-Javier Gallego and Jorge Calvo-Zaragoza. Staff-line removal with selectional auto-encoders. *Expert Systems with Applications*, 89:138–148, 2017.

[15] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, ICML '06, pages 369–376. ACM Press, 2006.

[16] Jan HajiÄ and Pavel Pecina. The muscima++ dataset for handwritten optical music recognition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 39–46, 2017.

[17] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630, 1957.

[18] E. T. Jaynes. Information theory and statistical mechanics. ii. *Physical Review*, 108(2):171–190, 1957.

[19] Xiaotian Lan, Qianhua He, Haikang Yan, and Yanxiong Li. A novel re-weighted ctc loss for data imbalance in speech keyword spotting. *Chinese Journal of Electronics*, 32(3):465–473, 2023.

[20] Hongzhu Li and Weiqiang Wang. Reinterpreting ctc training as iterative fitting. *Pattern Recognition*, 105:107392, 2020.

[21] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[22] Hu Liu, Sheng Jin, and Changshui Zhang. Connectionist temporal classification with maximum entropy regularization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 839–849, Red Hook, NY, USA, 2018. Curran Associates Inc.

[23] Yipeng Liu, Ruimin Wu, Yifan Wu, Lijie Luo, and Wei Xu. A stave-aware optical music recognition on monophonic scores for camera-based scenarios. *Applied Sciences*, 13(16):9360, August 2023.

[24] Jiří Mayer, Milan Straka, Jan Hajič, and Pavel Pecina. *Practical End-to-End Optical Music Recognition for Pianoform Music*, pages 55–73. Springer Nature Switzerland, 2024.

[25] Emilia Parada-Cabaleiro. *The SEILS Dataset: Symbolically Encoded Scores in Modern-Early Notation for Computational Musicology.; ISMIR.* 2017.

[26] Antonio Ríos-Vila, Jorge Calvo-Zaragoza, and José M. Iñesta. Exploring the two-dimensional nature of music notation for score recognition with end-to-end approaches. In *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 193–198, 2020.

[27] Antonio Ríos-Vila, Jorge Calvo-Zaragoza, and Thierry Paquet. Sheet music transformer: End-to-end optical music recognition beyond monophonic transcription. 02 2024.

[28] Antonio Ríos-Vila, Jorge Calvo-Zaragoza, David Rizo, and Thierry Paquet. End-to-end full-page optical music recognition for pianoform sheet music, 2024.

[29] Antonio Ríos-Vila, Jorge Calvo-Zaragoza, David Rizo, and Thierry Paquet. Sheet music transformer ++: End-to-end full-page optical music recognition for pianoform sheet music. 05 2024.

[30] Antonio Ríos-Vila, Miquel Esplà-Gomis, David Rizo, Pedro J. Ponce de León, and José M. Iñesta. Applying automatic translation for optical music recognition's encoding step. *Applied Sciences*, 11(9):3890, April 2021.

[31] Antonio Ríos-Vila, David Rizo, José M. Iñesta, and Jorge Calvo-Zaragoza. End-to-end optical music recognition for pianoform sheet music. *International Journal on Document Analysis and Recognition (IJDAR)*, 26(3):347–362, May 2023.

[32] Adrián Roselló, Eliseo Fuentes-Martínez, María Alfaro-Contreras, David Rizo, and Jorge Calvo-Zaragoza. *Source-Free Domain Adaptation for Optical Music Recognition*, pages 3–19. Springer Nature Switzerland, 2024.

[33] Elona Shatri and György Fazekas. Doremi: First glance at a universal omr dataset. 07 2021.

[34] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. 07 2015.

[35] Lukas Tuggener, Ismail Elezi, Jürgen Schmidhuber, Marcello Pelillo, and Thilo Stadelmann. Deepscores – a dataset for segmentation, detection and classification of tiny objects. 04 2018.

[36] Lukas Tuggener, Yvan Putra Satyawan, Alexander Pacha, Jürgen Schmidhuber, and Thilo Stadelmann. The deepscoresv2 dataset and benchmark for music object detection. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 9188–9195, 2021.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 06 2017.

[38] Cuihong Wen and Longjiao Zhu. A sequence-to-sequence framework based on transformer with masked language model for optical music recognition. *IEEE Access*, 10:118243–118252, 2022.

[39] Zecheng Xie, Yaoxiong Huang, Yuanzhi Zhu, Lianwen Jin, Yuliang Liu, and Lele Xie. Aggregation cross-entropy for sequence recognition. 04 2019.

[40] Mingru Yang, Qianhua He, Jinxin Huang, and Zunxian Liu. An effective sample learning: Biasing samples with high class uncertainty. 2024.

[41] Zengwei Yao, Wei Kang, Xiaoyu Yang, Fangjun Kuang, Liyong Guo, Han Zhu, Zengrui Jin, Zhaoqing Li, Long Lin, and Daniel Povey. Cr-ctc: Consistency regularization on ctc for improved speech recognition. 10 2024.

[42] Ali Yesilkanat, Yann Soullard, Bertrand Coüasnon, and Nathalie Girard. Full-page music symbols recognition: State-of-the-art deep model comparison for handwritten and printed music scores. In Giorgos Sfikas and George Retsinas, editors, *Document Analysis Systems*, pages 327–343, Cham, 2024. Springer Nature Switzerland.

[43] Xiang-Yi Zhang and Jia-Lien Hsu. Full-scale piano score recognition. *Applied Sciences*, 15(5):2857, March 2025.

Appendix

Appendix 1

Glossary

Adam A machine learning optimizer derived from Adaptive Moment Estimation, used to accelerate the learning of a gradient descent algorithm. It is used to update the weights of a given NN model along with the learning rate. It is considered very efficient.

Aggregation Cross Entropy Loss An alternative to CTC Loss for the sequence-to-sequence problem. The idea is to convert this from that kind of problem into a kind of counting problem. The inputs and outputs of the loss are identical to CTC, so any existing model that has CTC implemented already can have ACE put in instead. Instead of calculating the probability of correct symbol S_l in position l of the sequence given the image and model parameter ω, we instead attempt to calculate the probability that a given symbol indexed by k shows up N_k times in the image. Exists in two flavors, regression based and cross-entropy based [39].

Agnostic Encoding Encoding where every kind of symbol is given a unique token, regardless of any similarities regarding the shape composition tokens or their vertical positioning.all shapes are given their own symbol paired with a height value in the same symbol [6].

Autoregressive Transformer A kind of transformer that utilizes the autoregressive statistical method in order to perform prediction. The statistical assumption being made is that the current state of the system is the result of some function being applied to past values.

Bar See *Measure*.

Batch Normalization The act of normalizing the outputs of a neural layer using the means and variances of those outputs.

Bekern Stands for "Basic Extended Kern". It is a special tokenization scheme for Humdrum Kern that separates based on duration, accidental, pitch, and beaming.

Camera PrIMuS An extension of the PRIMuS dataset, wherein the pngs have been distorted by some noise in order to simulate images of sheet music taken by a real camera.

Camera Printed Music Staves A dataset of monophonic single-staff musical score images, taken via phone cameras, at various angles and lighting conditions [23].

Capitan Collection A set of scores written in Spanish mensural notation from the 16th to 18th centuries [7].

Character Error Rate A kind of edit distance metric. Defined as the number of substitution errors, deletion errors, and insertion errors on the character level summed together, divided by the number of characters in the ground truth. The evaluated metric essentially computes the average percentage of characters that were incorrect. This metric can be normalized by instead dividing by: the value in numerator plus the number of correct characters.

Computer Vision A machine learning task whose high-level goal is to utilize cameras and other visual recording mediums and have a computer analyze the information contained therein to perform some task.

Connectionist Temporal Classification Loss Function Defined as $L_{CTC} = -\ln(p(l|\mathbf{x}))$, where l is the correct output transcription given the current context $\mathbf{x}$. Used often in scenarios where there may not exact alignment been the input and output sequence. The core motivator is that sometimes the number of observations within some data does not have a 1-to-1 correspondence with the output labels. In CRNNs, this correlates to allowing the recurrent layer of the model to optimize the prediction of a symbol sequence with an input sequence from the last convolutional layer, with the output sequence not having to worry about the size of the last convolutional layer [15].

Convolutional Neural Network A very common architecture in used in neural networks that use the convolution operation [1]. Each layer performs the operation on some sequence and the resulting output is passed to the next layer.

Convolutional Recurrent Neural Network A combination of CNNs and RNNs, wherein after some number of convolutional layers, the output of the last layer is fed into a recurrent layer. We can think of this as the convolution portion being used to provide feature detection whilst the recurrent portion takes those features as the context when considering the prediction for the next token [34].

Convolutional Transformer A model based on the CRNN architecture, in which the recurrent portion is replaced with transformer decoder.

Cross-Entropy Loss A standard loss function used in machine learning that uses the concept of entropy from information theory the number of binary bits needed to get from one distribution to another. The

---

[1] though it is referred to as convolution, this is often actually mathematical cross-correlation operation

loss function is defined as measuring the amount of cross entropy from a prediction of a model to the actual answer.

Data Augmentation A technique used in ML where in existing data is modified in some way that allows for that modified data to be used as a part of the training set, in addition to the original data.

Deepscores A dataset for OMR. Initially introduced as a way to have a large dataset for not just OMR but computer vision in general – beforehand they claim there was no other dataset with small objects contained in large images. Some properties include: providing ground truths for multiple kinds of recognition tasks (object classification, semantic segmentation, object detection), having five different kinds of fonts, using an agnostic representation of symbols, etc. It contains articulations, ornamentation, and dynamic markings [35].

DeepscoresV2 An updated version of the Deepscores dataset, which contains more images and identification classes [2].

Dorico A music notation software developed and maintained by Steinberg.

Edit Distance Defined as the amount of work a theoretical user would need to do in order to take a transcribed work and make it correct. Often used as an evaluation metric in OCR research.

Encoder-Decoder Architecture A neural net architecture that can be broken down into two stages. An encoder that takes in some input and produces some set of features or hidden state that describe the

---

[2] https://zenodo.org/records/4012193

input, as well as as a decoder that takes in those features in an attempt to, in some way, reproduce, classify, or label the input to the encoder.

EnCTC A kind of loss function that adds an entropy regularization term H to the normal CTC calculation [22].

EnEsCTC A combination of EnCTC and EsCTC [22].

Entropy A concept in information theory that is used to capture the amount of information/uncertainty that is present within some random variable X. The more predictable the outcome, the lower the entropy.

Entropy Regularization The goal of entropy regularization is to maximize the amount of future entropy, in accordance with the principle of maximum entropy. When the amount of entropy is large, it indicates that the model has not yet sufficiently explored the environment and should continue to do so. In essence, having a high amount of entropy encourages the model to continue exploring the space rather than going down one predefined path. Contrast this with low entropy, which indicates that the model is able to predict the outcome with more ease.

EsCTC A variation on CTC that makes the assumption that, the spacing of two consecutive elements (or the width of the elements) is ”nearly the same” [22].

Evidence Lower Bound A approximation of some probability distribution P that provides a worst-case estimate for the log likelihood of P.

Faster R-CNN Stands for Faster Reigon-based Convolutional Recurrent Network. A neural network based on the CNN architecture where patches of an image are considered in feature detection rather than the entire image. Patches are considered in two stages (an initial set and then a further refined set), leading to faster performance.

Few-shot Learning A kind of learning in machine learning where models are trained on few samples but are still able to make accurate predictions.

FMT Corpus A OMR corpus of sheet music consisting of images of four groups of handwritten scores in modern notation with agnostic encoded ground-truths. The encodings use the same scheme as Camera PrIMuS [30].

Focal Loss Introduced in [21]. A loss function meant to address the issues with standard Cross-Entropy Loss. It introduces new hyperparameters and , which essentially scales the value of the loss result based on the confidence that the identified class is correct. Its defined as such: $FL(p_t) = -\alpha(1-p_t)^\gamma \log(p_t)$. Designed to limit how much high-frequency samples are able to impact the loss calculation.

FocalCTC It attempted to combined the ideas of focal loss into CTC loss by adding in the same hyperparameters α and γ. Defined as: $L_{Focal-CTC} = -\alpha(1-P(l|\mathbf{x}))^\gamma \log(P(l|\mathbf{x}))$ [11].

Full Convolutional Neural Network A model type used in OMR and HTR wherein only the CNN encoder is used rather than an encoder-decoder.

Grand Staff A special kind of staff in music that consists of a staff with a treble clef, and a staff with a bass clef, linked together by middle C.

GrandStaff Dataset A dataset that consists of pianoform scores as images along with their digital encoding. Consists of scores from the Humdrum Dataset as well as scores that have synthetic modifications made.

Ground Truth The "labels" or "answer key" for a set of data.

Humdrum Dataset A collection of files in the Humdrum Kern Format, not a single dataset but a collection. Used quite frequently as training data in OMR research.

Humdrum Kern Format A form of agnostic encoding used in OMR. It consists of multiple lines of symbols and is designed to be non-verbose, unlike MusicXML.

InterRNN A kind of RNN model in which after having multiple (single) recurrent layers be run in parallel, the information is merged, which is then passed into a second recurrent layer [3].

Kern See Humdrum Kern Format.

Kullback-Leibler Divergence A measure of how different one probability distribution is from another using the concepts of bits from information theory. Often used in loss functions for neural networks.

Line Error Rate An Edit Distance metric. Has the same fundamental idea as the Character Error Rate, but evaluated on the semantic level of a symbol instead. This can be a word, such as the case in OCR or a symbol in music for OMR. Defined as the number of Substitution Error, Deletion Error, and Insertion Errors on the line level summed together, divided by the number of lines in the Ground Truth. The evaluated metric essentially computes the average percentage of symbols that were incorrect. This metric can be normalized by instead dividing by: the value in numerator plus the number of correct lines.

Linearized MusicXML Utilizes linearization on the standard MusicXML format to a) remove excess data that is present within MXML and does not provide any learnable information to a model (such as in the starting and ending tags) and b) have an easier output for the model that can be converted in normal MXML later. Originates from [24].

Measure A unit of time within a piece of music. Much like how a word contains letters and those words are in sentences, a measure contains notes and those notes are in musical phrases.

Mensural Notation Music notation that differs from western notation. Used in Europe from the 13th to 17th century, usually in a choral setting.

MIDI A form of music encoding that allows for direct playback in almost all notation software, as well as non-notation software. A MIDI signal can also be sent from an input device to another device with a synthesized sound and be used to control that sound.

Mml-CTC A modification made to the CTC Loss Function. The core idea is to separate how blank tokens are output by first considering the probability distribution of encountering a blank first, and then only iff the token is not determined to be a blank does the model consider outputting a real token [8].

Monophonic Score A score with only once voice and no chords.

MUSICIMA++ A handwritten music dataset with ground truths consisting of a music notation graph describing how all symbols are connected and labelled bounding boxes for all symbols per image [3].

MuseScore A piece of open source music notation software.

MuseScore Database Refers to the website musescore.com, which contains thousands of user submitted and created scores.

Music Composing and Practice The act of producing music, either via engraving music to notation or playing it in a solo/ensemble setting.

Music Encoding Initiative (MEI) The MEI format is a form of XML not dissimilar to MusicXML, wherein it is used to fully describe how a score visually appears when rendered into a page [4].

Music Information Retrieval A high-level machine learning task in which the information is extracted from a piece of music.

MusicDiff A tool that allows users to visual the differences in music files.

MusicXML A text-based transcription of a piece of music, formatted like a traditional XML document. Understood by almost all modern DAW and notation-based software.

Object Classification The act of classifying an image as a single label.

---

[3] https://ufal.mff.cuni.cz/muscima

[4] https://music-encoding.org

Object Detection The act of classifying an image with multiple object labels and indications of where those objects are within the image.

OLiMPiC A subset of OpenScore Lieder, where samples have been turned from MusicXML to Linearized MusicXML [24].

OpenScore Lieder A collection of scores on the MuseScore Database consisting of public domain pieces for piano and vocals.

Optical Character Recognition The task of taking some some form of non-digitized text present within the real world and turning into a format that can be understood by machines (such as ASCII or Unicode).

Optical Music Recognition The task of taking some image of music notation (a score) and converting it into a machine understandable format.

Overfitting A term in ML used to describe when a model aligns too closely to the training data leading it to perform exceedingly well on that training data, but when validated or tested the model performs poorly.

PhotoScore An online tool that is able to, at varying degrees of accuracy, take an image of sheet music and transcribe it into encodings that a computer is able to understand [5].

Positional Encoding A mechanism used in transformer based neural networks, wherein each entity's position in a sequence is represented by some vector or matrix.

---

[5] https://photoscore.me/

Printed Images of Music Staves (PrIMuS) Consists of thousands of images of monophonic single-staff scores in several semantic and agnostic encoding schemes [6].

PyTorch A machine learning library used primarily with the Python programming language. It is open source and is under the Linux Foundation.

Re-weighted CTC Loss Function An alternative to CTC Loss, wherein we introduce a weighting term into the standard definition. Defined as $L_{RE} = -\mathbf{W}\ln(p(l|\mathbf{x})) = WL_{CTC}$, where $\mathbf{W}$ is updated as we go through training samples. The core idea is, for each sample in each epoch we introduce a weight that scales the loss based on the evaluation metrics chosen, such as edit distance [19].

Recurrent Neural Network A kind of network in which outputs of a layer are fed back into itself a set amount of times (typically until the current context ends) as a context window.

Region Based CNN A family of CNNs that ML models used mainly in computer vision and image recognition tasks. Input is an image, output is a set of bounding boxes in which each contains some object with a classification attached.

Residual Network A variation of a CNN layer, wherein layers are grouped by two. Before the 2 convolutions are performed, the input to the first layer is recorded. After the convolutions are applied, that is then added back to (or sometimes subtracted from) the output of the second convolutional layer. The goal is to preserve more of the original features from the input, resulting in less overfit.

Selection Auto-Encoder A modification introduced on top of the traditional auto-encoder model. The idea is to train the model such that the things that are being encoded are biased towards certain features (e.g. noteheads and accidentals, as compared to the staff lines) [14].

Semantic Encoding An encoding where every kind of symbol is given a unique token, regardless of any similarities regarding the shape composition tokens or their vertical positioning.

Sequence-to-Sequence A form of ML used a lot in the translation of natural language. The idea is to convert from a sequence in one domain to a sequence in another domain. Typically, two RNNs are used, one as an encoder and one as a decoder.

Sheet Music Transformer (SMT) A kind of Encoder-Decoder model used for OMR. The input is an image of a full score and the output is a full transcription for the model. At a high level, the encoder performs feature extraction from the image input using a CNN and the decoder is an Autoregressive Transformer. It is a probability model that predicts the most likely token given the input feature as well as all previous tokens generated. Before being flattened and passed to the decoder, the encoder output is applied to a 2D Positional Encoding. Additionally, after embedding but before being passed to the Transformer, a 1D Positional Encoding is applied [28].

Sheet Music Transformer++ An advancement to the SMT architecture that contained three main advancements: "system level pre-training", "incremental-system curriculum learning", "curriculum learning-based fine-tuning". The pre-training stage consists of just being able to do transcription of single systems. Incremental-system curriculum learning is the idea of training by slowing introducing more and more systems, the parameters of which can be fine-tuned by the user. The curriculum learning-based fine-tuning stage essentially uses a probability model that, over time, reduces the number of synthetic samples and performs training on a real sample [29].

Sibelius A music notation software developed and maintained by AVID.

Spiky Distribution Problem A problem that CTC models may encounter wherein the blank token is over predicted due to its prevalence in almost all predicted sequences.

Split-Sequence Encoding A way to encode sheet music in which every symbol has two main aspects: its shape ($s_i \in S$ where S is the set of all visually distinct shapes) and its height ($h_i \in H$ where H is the set of all valid heights aka valid vertical staff positions). Typically the encoded output of a model that uses this encoding looks something like $(s_{i1}, h_{i1}, s_{i2}, h_{i2}, \dots)$ [26].

ST-MLM A seq-2-seq model for OMR. Has five main stages: preprocessing, encoding, decoding, T-MLM, and output. Utilizes stochastic gradient descent and cross-entropy loss [38].

Staff System Essentially a line of music on a page, consisting of multiple measures. When looking at, say a Clarinet part, each line is known as a system. For a Piano, each set of staves that correspond to the same measure on a given line is a system. For an orchestral score, it is usually (though not always) the case that each page only contains one system, since even though all instruments that are present are on different staves, they represent the same set of measures. If we had, say a string quartet, every set of 4 staves we have that represent the same measures going across the page would be a system.

Stochastic Gradient Descent A technique in ML where the amount of adjustment of a model is determined by the gradient of the loss function. The idea is to consider the gradient in an dimensional space (where is the number of parameters of the model) and move in a direction that minimizes the loss function for each parameter.

Symbol Error Rate An Edit Distance metric. Has the same fundamental idea as the Character Error Rate, but evaluated on the semantic level of a symbol instead. This can be a word, such as the case in OCR or a symbol in music for OMR. Defined as the number of Substitution Error, Deletion Error, and Insertion Errors on the symbol level summed together, divided by the number of symbols in the Ground Truth. The evaluated metric essentially computes the average percentage of symbols that were incorrect. This metric can be normalized by instead dividing by: the value in numerator plus the number of correct symbols.

T-MLM An implementation of the Transformer model with the modifications to allow for decoder parallelization. These changes allow for both the encoding and decoding stages to be parallel [10].

Transformer A modification to the traditional seq-2-seq model, where an attention mechanism is used to weight the values in the encoder for each decoder time-step. It allows that allows the model to focus on some subset of the information when considering what output to generate at the point in time [37].

U-Net Architecture A kind of full CNN where pooling is replaced with Upsampling instead of pooling – layers actually end up increasing the resolution as a result. A typical architecture would use pooling (contraction) for a few layers, and then switch to upsampling (expansive). The idea is to reduce the amount of spatial information but increase the amount of feature information.

Underfitting A term in ML used to describe when a model doesnt align very closely to the training data leading it to perform poorly well on that training data. Usually used to describe when a certain feature doesnt appear frequently enough in the training data for it to be a predicted class output.

Upsampling Increases the "resolution" of data by using what is already there to extrapolate more information. Can be thought of as the opposite of downsampling via a convolution.

Variational CTC A modification made to the CTC Loss Function similar to Mml-CTC. Instead of trying to find out the concrete value of predicting a blank as in Mml-CTC, it instead uses the ELBO of $\mathbf{L}_{\mathrm{Mml}}$ and defines that as $\mathbf{L}_{\mathrm{Var}}$ [8].

Voice A concept in music engraving wherein multiple sets of musical rhythms are placed on the same staff. Each one of these sets is called a “voice”.

Wildcard CTC A modification to classic CTC Loss. The core difference is that in addition to the standard ¡blank¿ token, there is also a token that represents a wildcard character, similar to what you would see in the SQL standard. We define the token such that: . In addition, sometimes an additional weighted term is brought in [4].

Vita

Hritik Saynganthone was born in Mayfield Heights, Ohio, United States of America on March 5th, 2002, the son of Tarun Gupta and Manjari Gupta as Hritik Gupta. He is married to Alyssa Saynganthone as of October 26th, 2025. He is currently pursuing his Master of Science degree from Rochester Institute of Technology, United States of America. His research interest includes Machine Learning and Computer Vision as well as their applications to Music. His current research includes application of both of these interests with regard to Optical Music Recognition.

Permanent address: 268 Crittenden Way, Apt. 3
Rochester, New York 14623

This thesis was typeset with LaTeX[†] by the author.

[†] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.