**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**🙥🙦✯✯✯🙤🙧**



**PROJECT REPORT**

**OPTICAL MUSIC RECOGNITION**

Subject: Computer Vision

Instructor: Dr. Dinh Viet Sang

| **Student name** | **Student ID** |
| --- | --- |
| ***Vũ Công Duy*** | ***20176737*** |
| ***Trần Công Minh*** | ***20176825*** |
| ***Đào Hồng Quân*** | ***20176850*** |
| ***Trần Thị Hằng*** | ***20176748*** |
| ***Dương Thị Huê*** | ***20176772*** |

Hanoi, January 2022

[**INTRODUCTION**](#_heading=h.30j0zll) **3**

[Project Overview](#_heading=h.1fob9te) 3

[Original Work](#_heading=h.3znysh7) 3

[Variables](#_heading=h.2et92p0) 3

[**THEORETICAL BACKGROUND**](#_heading=h.tyjcwt) **4**

[Neural network](#_heading=h.o0n4fl9j8ua6) 4

[Convolutional Neural network](#_heading=h.5r3ivlx94cn7) 5

[Recurrent Neural network](#_heading=h.fnwcvbxbutim) 6

[Connectionist temporal classification](#_heading=h.rsuz155t7v6b) 7

[**IMPLEMENTATION**](#_heading=h.3dy6vkm) **8**

[Dataset](#_heading=h.stjl7z7eoohu) 8

[Framework](#_heading=h.tcwuqe3i6t2p) 8

[Data preprocessing](#_heading=h.m80id0chj6z4) 8

[Convolutional recurrent neural network](#_heading=h.ba7d00h3xhsq) 8

[Multitask learning](#_heading=h.6br5igawm8am) 8

[**RESULT**](#_heading=h.1t3h5sf) **8**

[Evaluation metrics](#_heading=h.nkyvb4hvhv2d) 8

[Performance](#_heading=h.y5y2qjt6gwlv) 8

[**OVERVIEW**](#_heading=h.wkvcdqjuhu27) **9**

[Data](#_heading=h.phwdonhjoyey) 9

[Models](#_heading=h.gg1esv9cgko6) 9

[**REFERENCES**](#_heading=h.sq1s2b2qrtz) **9**

[Libraries](#_heading=h.39m4w9s4x3ad) 9

[Related works](#_heading=h.6jqjqbkjbwys) 9

# INTRODUCTION

## Project Overview

Reading and understanding music sheets is one of the most straightforward examples of optical tasks that can be improved with the help of computers. Yet there is not enough research on this subject. The lack of sufficient datasets (mostly because of labels and copyright) was only resolved in the recent years. It took 50 years for the optical music recognition (OMR) field to gain attraction.

While it is possible to tackle music reading without the use of advancements like neural networks, those methods rely greatly on the property of the dataset used. The use of deep learning guarantees that the computer vision project can be applied on any type of data.

In this project, we will implement and improve an existing OMR neural network project to read and translate images of single-staff scores. The main model uses convolutional recurrent neural network with connectionist temporal classification loss.

## Original Work

This project is based on Jorge Calvo-Zaragoza and David Rizo’s research paper “End-to-End Optical Music Recognition of Monophonic Scores”. This work is the baseline for modern approaches to the OMR task.

We chose this research due to the massive collection of data presented by the authors and the simple model structure.

Since the source code was written in the deprecated Tensorflow 1 framework, we rewrote the code using Pytorch. We also implemented multitask learning to boost the performance of the project

## Variables

There are two types of image input for each sample. The first one is a basic png file of the score. The other is the jpg file containing the distorted version. These inputs are trained separately to test how generic the model is.

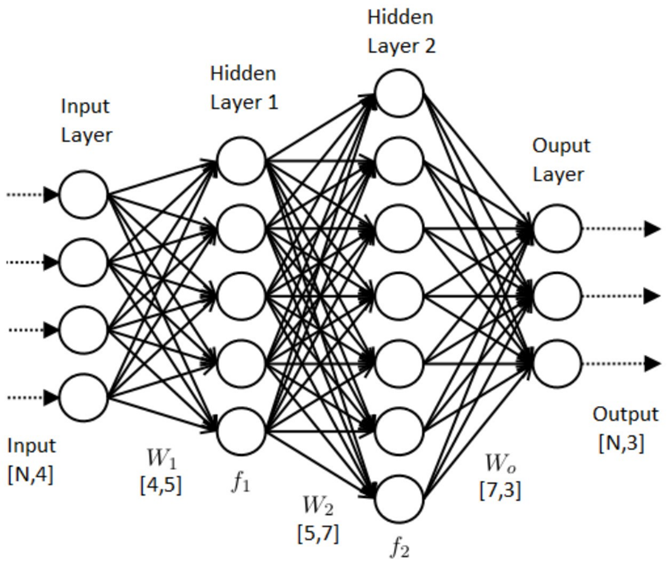
There are also two types of output, both are a sequence of symbols representing the image. The semantic label refers to the musical meaning of symbols while the agnostic label ignores meaning and describes the shape and position of symbols on the staff. The separation helps reduce the alphabet size and make the prediction more accurate.

# THEORETICAL BACKGROUND

## Neural Network

The architecture of the neural network was inspired by and partly mimics the biological construction of the human brain. The network consists of computational units called neurons, or nodes.

Each node, after receiving input information from either other nodes or an external source, will perform data calculation and produce an output. Associated with every input is a weight (w), which signifies its relative importance in comparison to other inputs. Multiple nodes are divided into different collective groups called layers. After receiving the weighted sum of input, the node will apply some activation function on the sum to produce a result.



A traditional Artificial Neural Network (ANN) usually includes the following components:

* Input Nodes (input layer): The layer that receives information from external sources and passes them to other layers.
* Hidden nodes (hidden layer): The layers where intermediate computation is processed.
* Output Nodes (output layer): The layer that conforms to a desired output format for the neural network.
* Connections and weights: Each node transfers information to other nodes through a connection, which is assigned with a particular weight.
* Activation function: the activation function to be applied by a node on the weighted sum of input to produce a result.

When the network receives the input, it will feed forward the information through its layers and apply the activation function to create the desired output. During the training process, the loss between the output and the ground truth is calculated using a certain function. With the help of an optimizer, the network uses the loss to modify its weights accordingly through backpropagation.

Complex tasks require more advanced architectures such as Generative Adversarial Network (GAN), Convolutional Neural Network (CNN), or Recurrent Neural Network (RNN). Different architectures can also be combined to build a complete deep learning model.

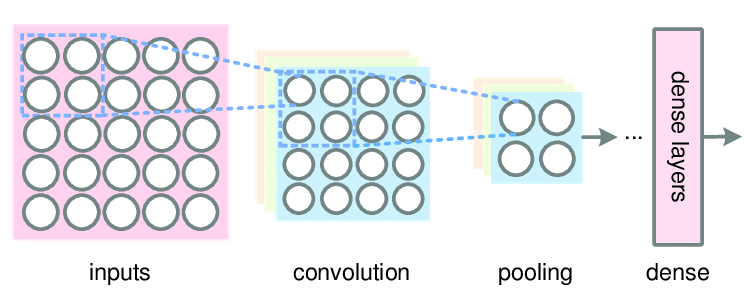
## Convolutional Neural Network

ANN does not perform well in image processing. If each pixel were to be used as an input value for such a network, the input size would be equal to the area of the image multiplied by the number of channels. For instance, a 128x128 black and white image when flattened will require 16,384 input nodes, and the model’s number of nodes may grow exponentially with the number of layers. This may lead to numerous problems, including severe computational cost and overfitting. Therefore, working with images requires an advanced neural network type, called Convolutional Neural Network.

The main difference between CNN and ANN is the convolutional layer. Before feeding information to the typical neural network layers, CNN uses the convolutional layers to extract only relevant information from the input, thus reducing the size and computation cost.

The convolutional layer applies filters to the original image by moving a kernel from top left to bottom right and employing convolution operation. Since the kernel functions by performing many-to-one mapping, proximity will be considered; in other words, related or shared information between nearby pixels will be taken into calculation. Each filter produces a different channel, revealing more features as the number of filters increases.

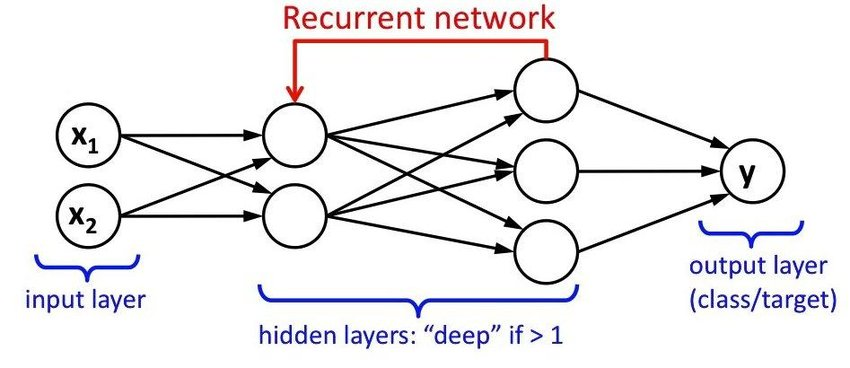
Each convolutional layer is usually accompanied by a pooling layer, which performs operations such as averaging or finding max value in the filter region. Through these layers, the information will be extracted for meaning and also reduced in size.



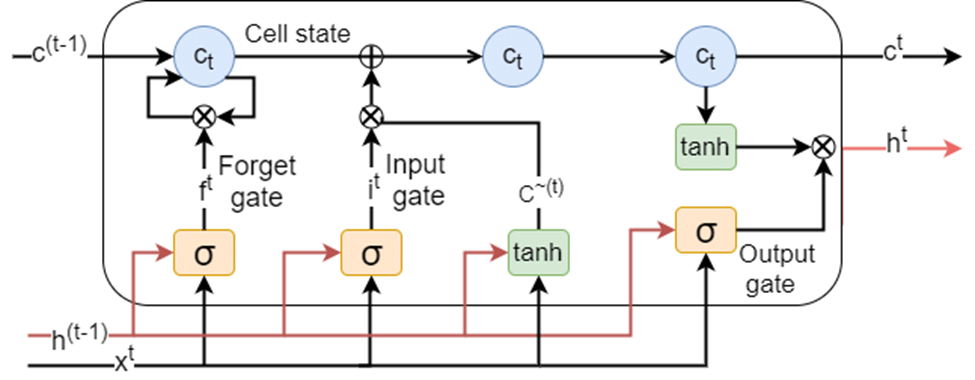
## Recurrent Neural Network

Like CNN, the recurrent neural network was constructed to solve certain problems that the traditional neural network falls short of, specifically ones involving processing sequential data, such as time series prediction, speed recognition or natural language processing.

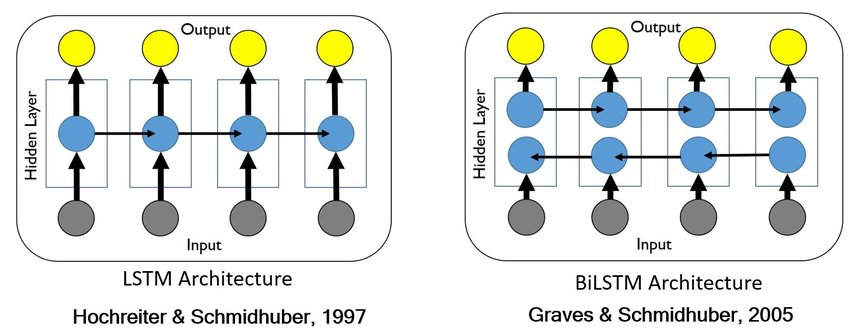
RNN introduces the concept of memory. In RNN, the output of a node will be used as input for previous layers or nodes of the same layer, allowing information to persist.



One of the most popular RNN architectures is the Long Short-Term Memory (LSTM) architecture. A common formation of LSTM units is composed of a cell and three regulators. The three regulators, which include an input gate, an output gate and a forget gate, control the flow of information in and out of the cell. The cell acts as the memory, saving different arbitrary values through time while the forget gate decides what to drop.



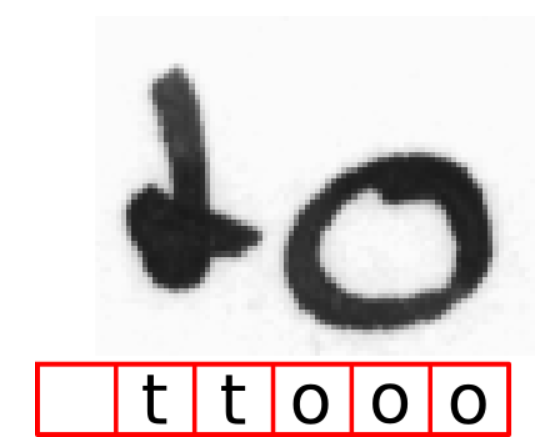
An extension of LSTM is the bidirectional LSTM, or biLSTM, a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. BiLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm.



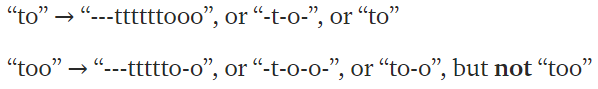
## Connectionist Temporal Classification

In optical recognition tasks such as music and text reading, even with size reduction from the convolutional layers, the output size is still significantly larger than the length of the label. The space of each symbol in the images also varies, making it difficult to divide the image evenly and predict each part. The solution for these tasks is Connectionist Temporal Classification (CTC).

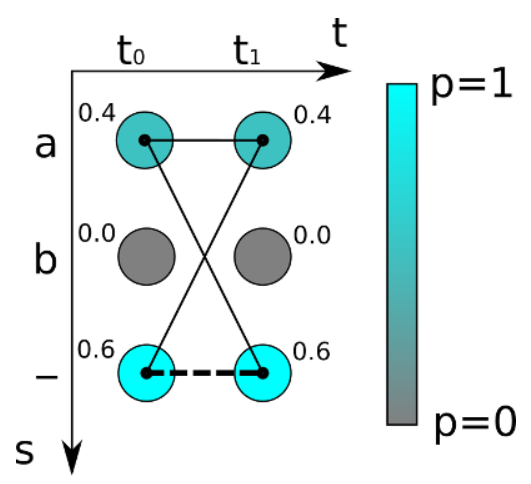
In CTC, the task is treated as a multi-class prediction problem, with target cardinality equal to the alphabet plus one. The output classes include the original alphabet and a “blank” token.

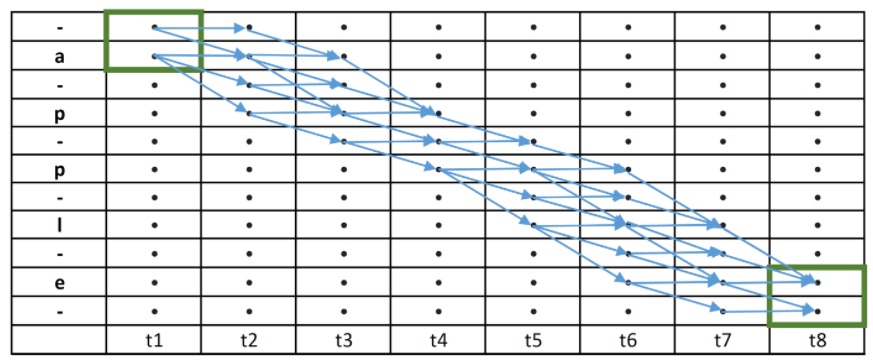


The prediction is treated as a time series with each step corresponding to the state with the highest probability. If a subsequence has the same state in all time steps, it is considered a single character. The blank character is used to pad the output or separate characters, especially for separating duplicate characters.



Since the output is a matrix of probability between each element and each of the classes, the ground truth score is calculated by summing the probabilities of all combination sequences that return the ground truth. The ideal score should be as close to 1 as possible. CTC loss is defined by the negative logarithm of this score.





For the alphabet {a,e,l,p}, ground truth “apple”, and sequence of length 8, each dot is a value of the output and the score is calculated by the sum of probability of all blue paths.

# IMPLEMENTATION

## Dataset

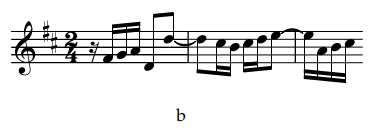
The dataset of this project, PrIMuS, is retrieved from the original work, which consists of only single-staff, monophonic scores. This collection contains enough combinations of high quality data for deep learning. We chose the updated dataset, which includes a distorted version for each of the images.



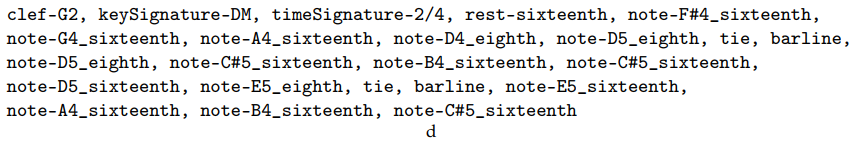


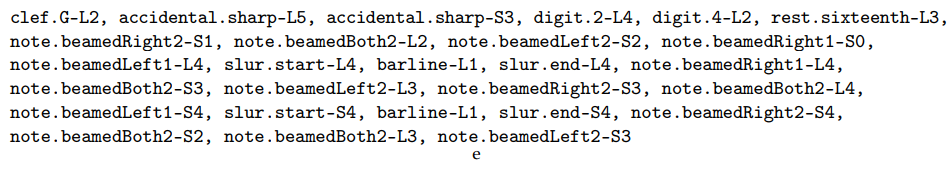
PrIMuS contains 87,678 real sequences of notes, each represented in a folder of six files: the two images, normal and distorted; the representation of the scores in pae and mei formats, which can be used for real procedures like converting to sound file or rendering scores; these code files are simplified into semantic encoding for meaning and agnostic encoding for graphical description. The last two files are used as the ground truths for this project.

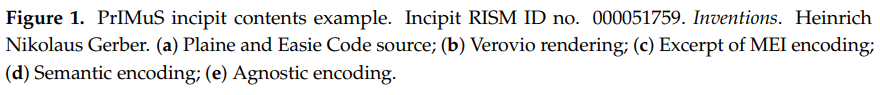




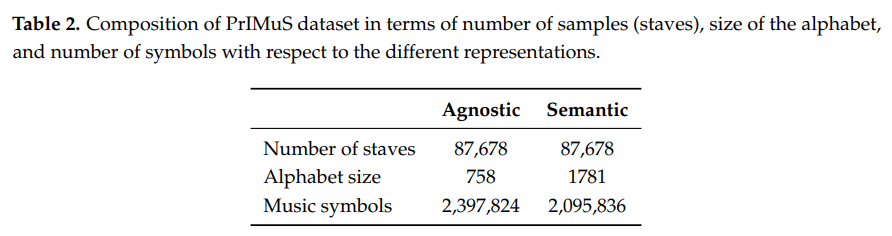








While it is obvious that the main goal of the OMR task is to translate the meaning of the image, which puts the semantic representation in higher priority, the agnostic representation still holds some advantages. It focuses on the shape and position of the symbols, helping the model adapt to music scores with different display styles. Therefore, the two encoding types have the same importance in this project.



## Framework

The original work was written in Tensorflow v1, which has been deprecated. We decided to rewrite this model using Pytorch 1.6.0.

## Data preprocessing

Since the samples are grouped as independent folders and the sample size is massive, a new data loader class should be created for the sole purpose of loading and preprocessing data by batch.

The library OpenCV 4.0.1 is used to open and scale images. As the images do not require color, they are treated as one-channel black and white images. Normalization is also necessary during this process.

The input data are grouped as batches. After resizing to the same height, the images still vary in length and need to be padded to the length of the largest one in the batch.

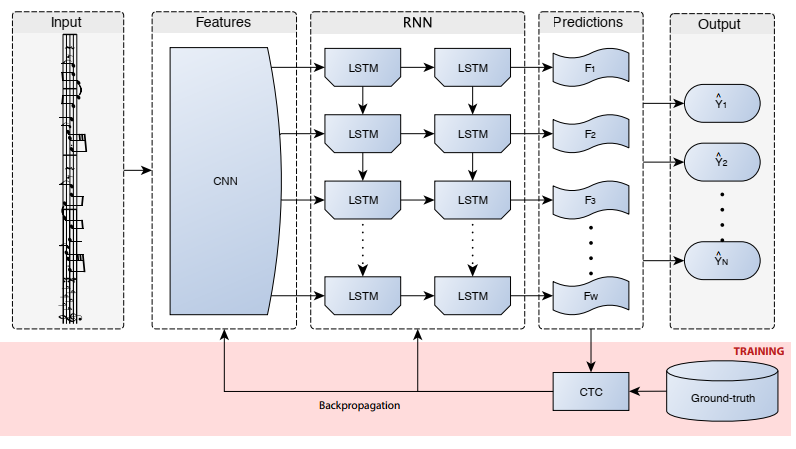
The label of each image needs to be converted to a sequence of numerical values (classes). For the CTC loss function, an array containing the length of each label in the batch is also required.

During each training iteration, the data loader processes and returns the necessary data in batch. During the validation phase, only a batch of the same size is used for evaluation instead of the whole set, due to hardware limitation.

## Convolutional Recurrent Neural Network

For this project, CNN and RNN architectures are combined to make the Convolutional Recurrent Neural Network (CRNN). The convolutional layers will extract features from the input image and flatten them to feed the recurrent layers. The final result is performed log softmax operation to create a matrix of probability.

The output, its label, and arrays containing their respective length are then fed to the CTC loss function for backpropagation and validation.



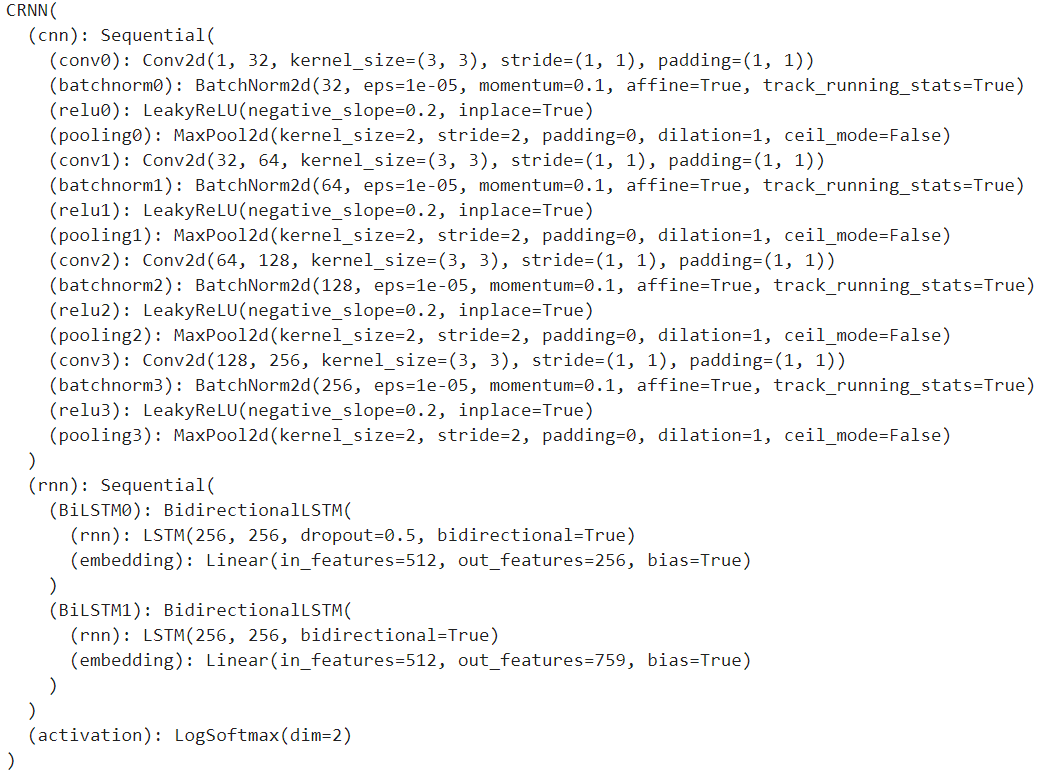
Each convolutional layer, kernel size 3, padding and stride of 1, is accompanied by a batch normalization layer, the leakyReLU function, and a max pooling layer of kernel size and stride 2. There are four layers in total, with the respective numbers of filters 32, 64, 128, 256

Each recurrent layer is a self made BiLSTM layer consisting of a bidirectional LSTM layer and a dense layer. Two layers are used, with 256 hidden states in each layer.

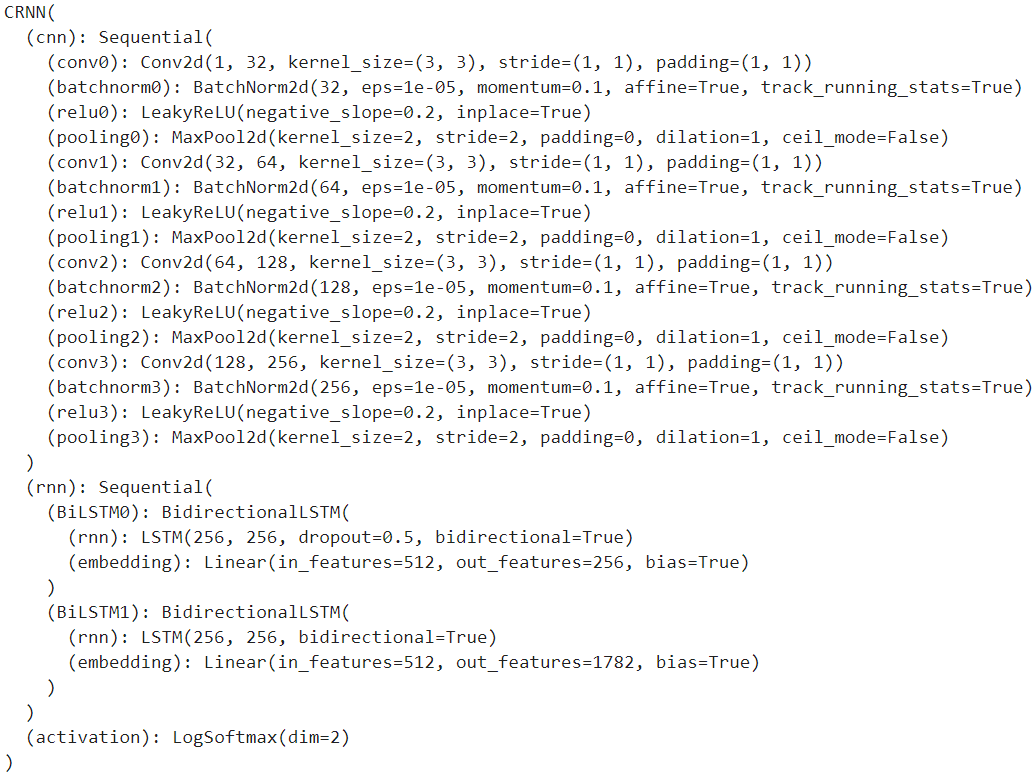
In this project, the CTC loss function is used. For optimization, we used the Adam algorithm. Due to hardware limitation, each iteration has the batch size of 4 and the number of epochs is 5. 70% of the dataset is for training, 20% for validation, and the remaining 10% is for testing. With 15,344 iterations in each epoch, the whole training phase will go through 76,720 iterations.

There are two distinct models: semantic and agnostic recognition. The difference between these models is the number of output classes.

These two models will be trained on two separate dataset: normal images and distorted images. There will be four models with two architectures in total.



Agnostic model



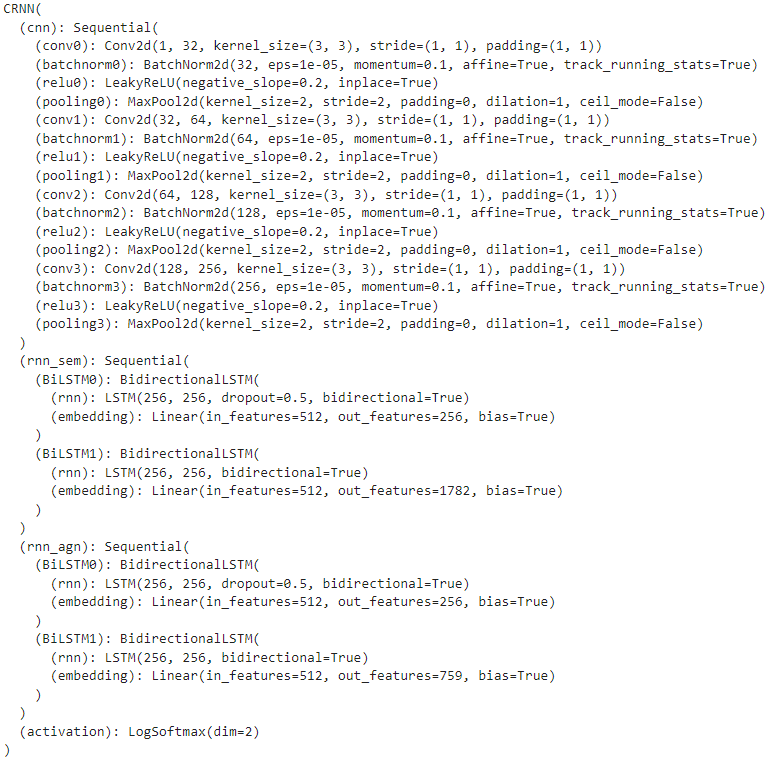
Semantic model

## Multitask learning

In an attempt to improve the performance of the model, we hypothesized that both recognition models will benefit from the same feature extraction layers. Therefore, the two models are combined into a multitask model, where the input is extracted and fed to two separate recurrent layers, returning two outputs. The loss function is then calculated by the sum of semantic and agnostic CTC loss.

The parameters of this model do not differ from those of the previous ones.

However, the model is prone to overfitting. To tackle this problem, we used weight decay with a gamma rate of 0.99 every 1000 iterations. Smaller rates may hinder the model from learning further.



Multitask model

# RESULT

## Evaluation metrics

The original work relied on two metrics: sequence error rate and symbol error rate.

Sequence error rate refers to the percentage of sequences with at least 1 wrong prediction.

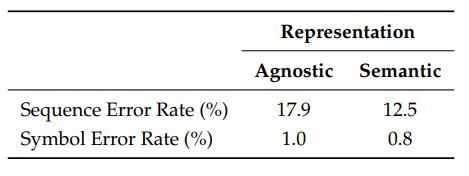
Symbol error rate refers to the amount of modifications required to produce the ground truth from the predicted sequence. Levenshtein distance is the perfect metric for this.

During testing, we realized that some predicted symbols, while wrong, are not entirely different from the label. These errors tended to be small even in real life, such as the notes “do” and “re” may be confused due to being close to each other. We proposed symbol confusion rate, where the difference rate between the wrong symbols is returned instead of the number of modifications.

We also measured accuracy instead of sequence error rate.

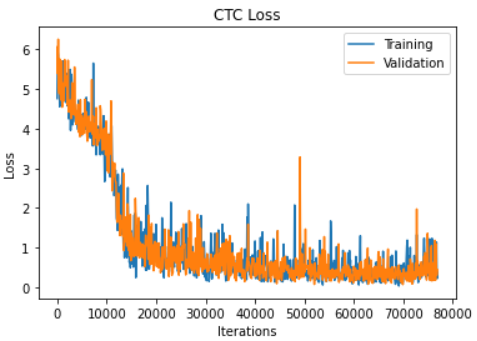
## Performance

Normal set:

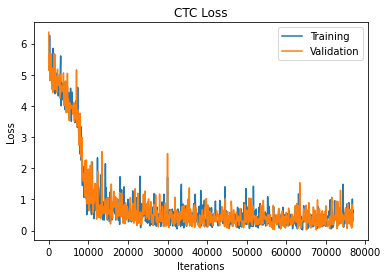


We could not produce the results as good as the baseline model, due to the difference in batch size (4 vs 16) and epochs (5 vs 16).

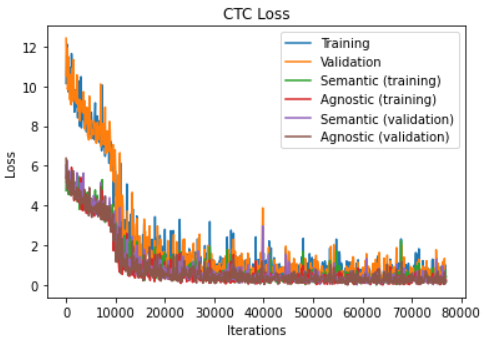
The semantic model has accuracy of 24.24%, with symbol error rate of 10.14% and confusion rate of 4.17%.



The agnostic model, just like in the baseline, has worse results, at 18.98% accuracy, 14.13% symbol error and 8.41% confusion rate.



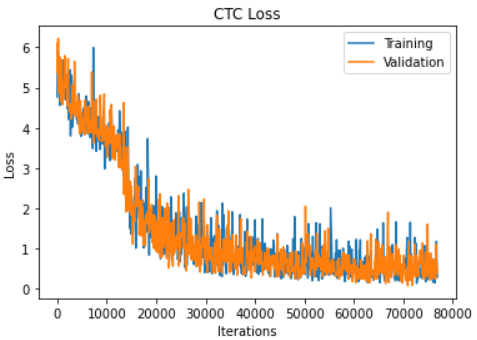
Multitask learning actually improved the results of both predictions, with the complete accuracy of both outputs at 20.29%. Semantic prediction increased from 24.24% to 27.67%; its symbol error and confusion rates also dropped to 9.40% and 3.91% respectively. The agnostic prediction benefited greatly from this model, as it surpassed semantic accuracy, rising to 32.15% with 9.13% symbol error and 4.89% confusion. The model also fitted faster.



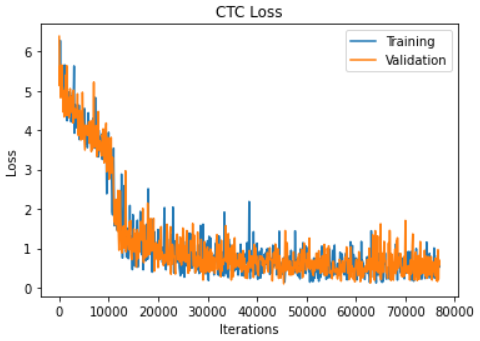
Distorted set:

There is no result for the baseline model in this project. The results were not as encouraging as the normal dataset, but showed some potential for the multitask model.

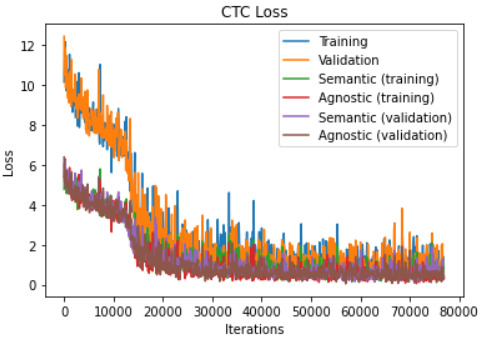
The semantic model dropped to 17.96% accuracy with 14.20% error and 7.50% confusion.



The agnostic model only had 8.51% accuracy, 18.20% symbol error rate, and 12.88% confusion rate.



The multitask model, while no longer improving the result of both models, balanced the difference. It only had 6.17% complete accuracy. Semantic and agnostic accuracy are at 14.01% and 14.29%; symbol error rates 15.15% and 14.94%; symbol confusion rates 6.99% and 9.74%.



# OVERVIEW

## Data

The data of this project was relatively simple, as real life music sheets may have multiple notes at the same time or have multiple staffs (which can be dealt with using staff recognition). However, the dataset is sufficient for small research on CTC tasks in general.

While the distorted data had lower results, the small difference meant that the model to some extent worked with noise in data.

## Models

Multitask learning can improve the results and balance the difference between models. However, the model is vulnerable to overfitting.

As 80,000 iterations required 5 hours and there was only enough memory for batch size 4, we could not run this project exactly like the baseline model.

This project also required optimization in coding to prevent exhausting CPU and GPU

# REFERENCES

## Libraries

pytorch=1.6.0

opencv=4.0.1

numpy=1.19.2

matplotlib=3.4.2

## Related works

OMR: <https://github.com/OMR-Research/tf-end-to-end>

CRNN: <https://github.com/meijieru/crnn.pytorch>

## 