Project Assignment - Image Processing

Pouya Mohseni - 610398164

Handwritten music symbol recognition

Abstract

In this assignment the problem of optical music recognition is investigated, the relevant datasets are examined, and models regarding solving this problem are introduced. Additionally, we aim to replicate and evaluate the novel model proposed in the research paper titled "An ensemble of deep transfer learning models for handwritten music symbol recognition" by simulating it and conducting tests. In the end, we introduce several other models in order to out-compete the proposed model in the paper. These models are leveraged by the ensemble or hierarchy ensemble methods. Three of our proposed models exceed the model introduced in the paper in accuracy. Two of the presented models are based on a 1-layer ensemble and one of them is based on a hierarchy of ensemble models. To demonstrate our approach, we illustrated the successful models in this report. The code of this project can be found in the three attached files.

Table of Contents

1. Introduction

Optical music recognition (OMR) is a field of research that focuses on developing systems and algorithms capable of automatically interpreting and analyzing handwritten or printed music scores. OMR technology plays a vital role in digitizing and processing musical compositions, enabling a wide range of applications such as music transcription, score editing, and music information retrieval.

However, recognizing and deciphering the handwritten music symbols poses a significant challenge due to the inherent variations observed across different sheets. These variations can arise from factors such as individual handwriting styles, varying musical notations, and the quality of the scanned images. As a result, developing effective models capable of accurately identifying and classifying these diverse symbols is crucial for advancing the field of OMR and enabling efficient automated music analysis.

Symbols employed in musical notations bear resemblance to the letters of an alphabet, rendering music symbol recognition a critical component of any optical music recognition (OMR) system. However, recognizing handwritten or pen-based music symbols poses a challenging endeavor due to subtle variations exhibited across music sheets created by different individuals. In the past, machine learning techniques have been extensively applied in the realm of music symbol recognition, both in online and offline scenarios. Various studies [1][2] have employed diverse feature extraction methods, occasionally coupled with feature selection techniques, and utilized models such as support vector machines (SVM) and k-nearest neighbors (k-NN) for effective classification purposes.

2. A Glance at OMR Datasets

Optical Music Recognition (OMR) consists of a wide range of tasks, including symbol classification, staff line removal, and various end-to-end recognition problems. A valuable collection of datasets relevant to these tasks can be found in reference [3]. In addition, Shatiri has also introduced a list of significant datasets in the field of OMR, contributing to the existing literature [4]. However, in this assignment, we will focus on symbol classification which plays a vital role in the literature.

Dataset	Engraving	Symbols	Images	Classes	Format	Usage
DoReMi	Typeset	911771	6432	94	✓ML metadata, images, MIDI, MEI, MusicXML	Object Detection, Reconstruction and Encoding, End-to-end
Handwritten Online Musical Symbols (HOMUS) [6]	Handwritten	15200	-	32	Text-File	Symbol Classification (online + offline)
Universal Music Symbol Collection [7]	Typeset + Handwritten	90000	-	79	Images	Symbol Classification (offline)
MUSCIMA ++ [4]	Handwritten	91255	140	110	Images, Measure Annotations, MuNG	Symbol Classification, Object Detection and Measure Recognition
DeepScores [9], [19]	Typeset	100m	255,386	135	Images, XML	Symbol Classification, Object Detection, Semantic Segmentation
PrIMuS [11]	Typeset	87678	-	-	Images, MEI, simplified encoding, agnostic encoding	End-to-End Recognition
Capitan collection [20]	Handwritten	-	10230	30	Images, Text-File	Symbol Classification
Bounding Box Annotations of Musical Measures [21]	Typeset	940	24,329	-	Images	CSV, plain JSON and COCO

Table 1) Comparison of major OMR datasets [4]

The symbol classification problem, also, consists of two different problems: 1) Hand-written symbol classification without staff line, and 2) Hand-written symbol classification with staffing. However, the consistency of these data sets gives the flexibility to use them on these two problems.

a. Working with Handwritten data

Handwritten music symbol data does not typically consist of images in a commonly used format. Instead, it primarily takes the form of a text file that includes stroke information representing the process of writing the symbol where each stroke is a set of 2D points separated by a semicolon. In turn, each dimension is separated by a comma. An example of a Quarter Note is as follows:

```
Quarter-Note
114,155;112,155;104,157;100,160;98,162;98,163;99,164;103,163;108,160;112,158;114,
156;116,154;116,151;114,151;112,152;108,154;102,159;100,161;100,162;103,163;107,1
62;112,159;115,158;117,156;117,154;116,154;113,155;110,158;108,160;106,163;106,16
4;106,163;108,163;110,161;111,160;112,157;112,155;111,155;109,155;106,160;104,162
;104,162;106,162;106,162;128,85;128,85;127,89;126,101;124,112;123,124;122,136;121
,145;120,154;120,158;121,159;121,159;
```

Which can be translated into an image as below:

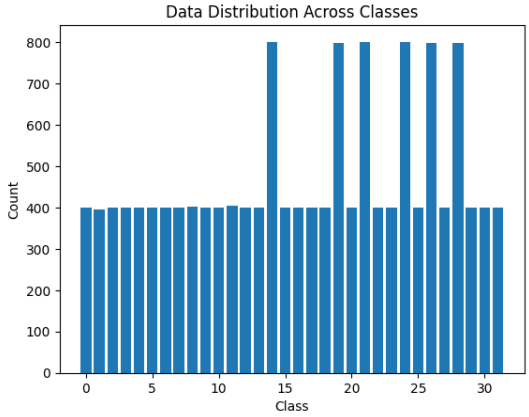
There are many ways in order to generate an image of these text files. One of which is using a module in Python named `omrdatasettools`. Below is a line of a code that generates an image from the text file:

```
HomusImageGenerator.create_images(raw_data_directory="HOMUS",
                                  destination_directory="homus_data",
                                  stroke_thicknesses=[3],
                                  canvas_width=96,
                                  canvas_height=192,
                                  staff_line_spacing=14,
                                  staff_line_vertical_offsets=[24])
```

Using these modules enables the image to be generated in any size with respect to the aforementioned problems. However, we just focus on generated images without staff lines:

To clearly understand the HOMUS dataset, we calculate the total number of classes and images. Moreover, we plot the distribution of instances in each class:

```
Total number of instances: 15200
Number of classes: 32
```

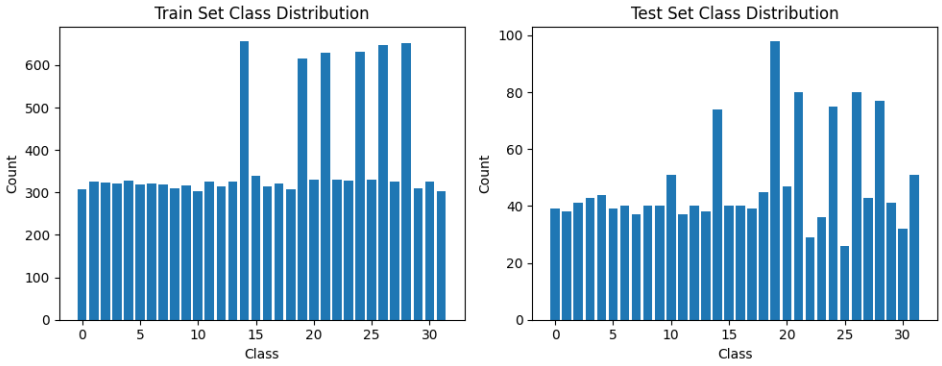


3. Applying a CNN

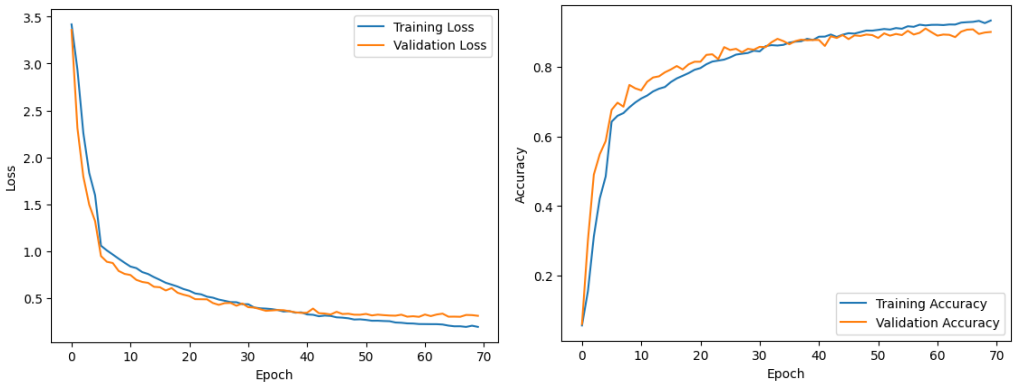
Firstly, we apply a Convolutional Neural Network (CNN) to analyze the dataset. Nevertheless, previous studies have indicated that training a CNN on this particular dataset can be a time-intensive process. It is important to note that our primary objective is not to achieve exceptionally high accuracy using this model.

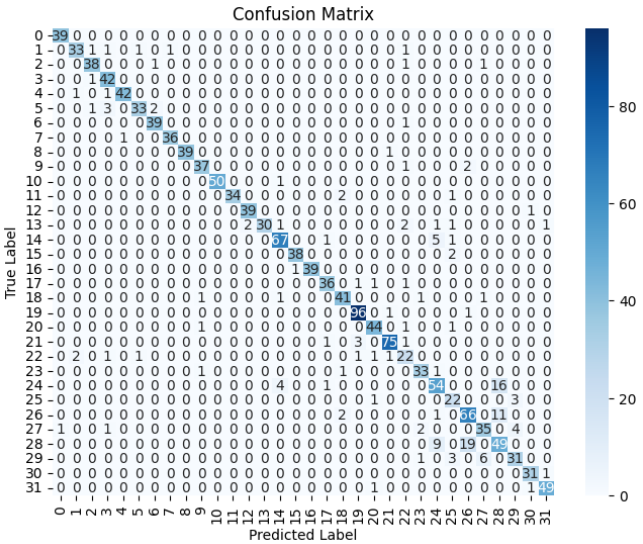
To train the model, we resize the input image to `(96, 96)` and normalize the pixel values. Subsequently, we split the input image with an 8/2 ratio to train and test data sets. In the following train and test distribution over classes are illustrated:

```
Number of instances for training: 12160
Number of instances for testing: 1520
```



After training the CNN model on this multiclass problem, we obtained an accuracy of 89.41% on the test set with 70 epochs which took around 1 hour. While this accuracy is commendable, it falls short compared to the recent achievement of 98% accuracy in this task. Here is a graph illustrating the training improvement in loss and accuracy in both validation and training sets which is followed by the confusion matrix of the performance of the model on the test set:

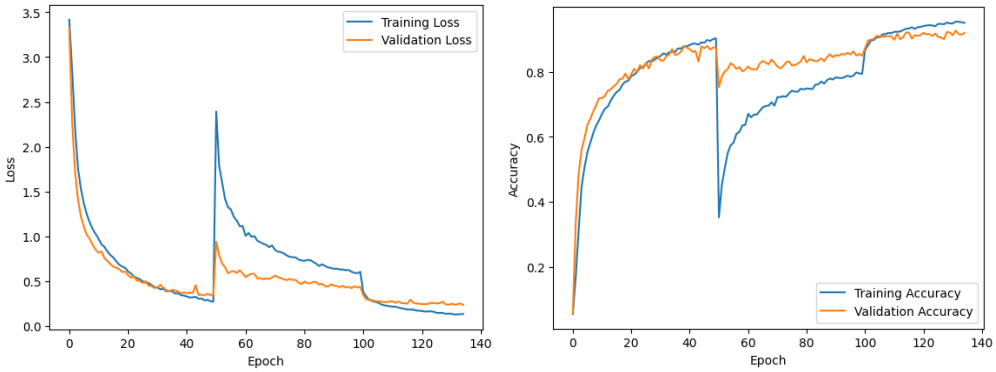


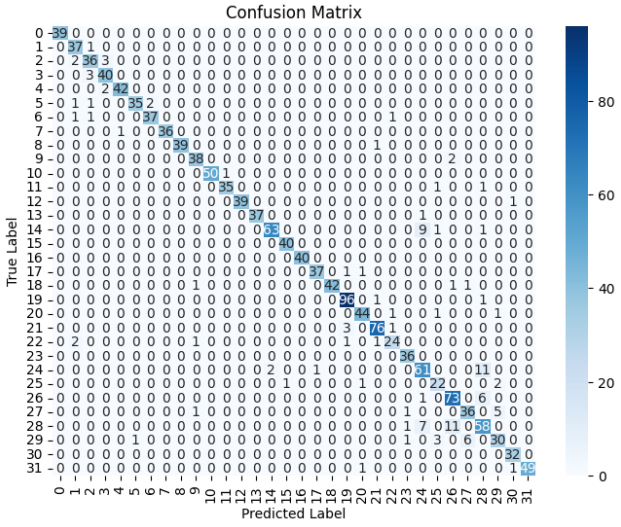


In order to further improve the accuracy of our model, we decided to implement an augmentation layer. This layer introduces additional variations in the training data by applying random transformations such as rotation, scaling, and flipping to the images. By augmenting the dataset, we aim to provide the model with a more diverse and robust set of training examples. The code below shows the augmentation methods that are performed randomly on the training set:

```
def data_augmentation(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    degrees = tf.random.uniform(shape=[], minval=-5, maxval=5)
    radians = degrees * tf.constant(3 / 180.0)  # Convert degrees to radians
    k = tf.cast(radians // (tf.constant(3 / 2.0) + 1e-7), tf.int32)
    image = tf.image.rot90(image, k=k)
    return image, label
```

We then retrained the neural network using the augmented dataset. For 50 epochs the model is fed with the non-augment data. However, after that, for 50 more iterations the augmented data is introduced to the model. Finally, in the last 40 epochs the model is fed with the non-augmented training data. The augmentation method helped to enhance the model's ability to generalize and capture important features, resulting in an improved accuracy of 92% on the test set. This demonstrates the effectiveness of data augmentation in enhancing the performance of the CNN model on this dataset.





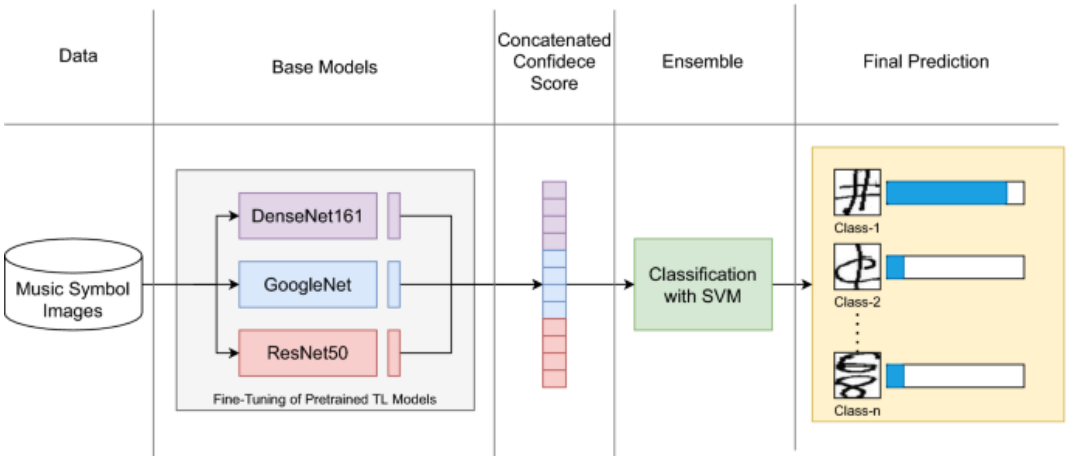
4. Transfer Learning

As we saw, the training of the model takes a considerable amount of time. Therefore, we decided to explore the potential of transfer learning to leverage pre-trained models and diminish the training time. Transfer learning involves utilizing the knowledge and learned representations from a pre-trained model on a different but related task and applying it to our specific problem.

In this section, we explore the paper titled "An Ensemble of Deep Transfer Learning Models for Handwritten Music Symbol Recognition" [5], which implements ensemble learning utilizing transfer learning of three renowned models augmented with an SVM model. Transfer learning involves training these models with all layers locked except for the last one, which is fine-tuned during the training phase.

Inspired by the methodology presented in the paper, we applied transfer learning to fine-tune three specific models: DenseNet161, GoogleNet, and ResNet50. After the fine-tuning process, we evaluated the accuracy of each model on the test set.

In Section 5, we aggregate the results of these three models using an SVM model. This design is the one that is used in the paper [5] which can be seen below:



However, based on the manuscript, the initialization and model setting are not clear; the training set, test set, and validation set are not defined clearly; moreover, the number of epochs as well as the size of the input image is not determined in the paper. Therefore, we guess these parameters based on our knowledge as well as our computation power. Therefore, we are not expecting the same results as the paper because of the potential of different initialization and setting.

However, the manuscript lacks clarity regarding the initialization and model settings. Additionally, the definitions of the training set, test set, and validation set are not explicit. Moreover, crucial details such as the number of epochs and the size of the input image are missing from the paper. As a result, we had to make informed guesses considering our computational resources to determine these parameters. Consequently, we do not anticipate achieving identical results to those reported in the paper due to potential initialization and setting variations.

We considered 20% of the dataset as the test set, 2.5% as the validation set, and 77.5% as the training set. We, also, considered 20 epochs for fine-tuning the model. Moreover, 96*96 images are chosen for the input image size with a stroke thickness of 3 pixels. Moreover, the authors do not point to the fact that they freeze all of the layers except for the last one; therefore, we adhere to this procedure, although it increases our computational overhead.

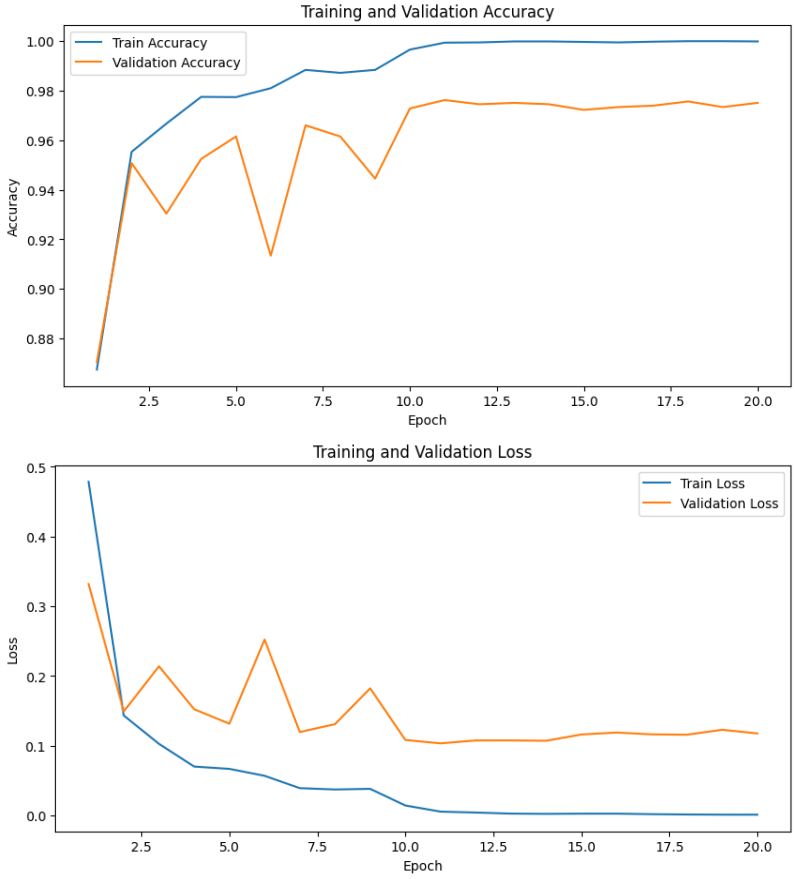
```
train_files = class_files[:num_train_samples]
test_files = class_files[num_train_samples:num_train_samples + num_test_samples]
validation_files = class_files[num_train_samples + num_test_samples:]


def get_models():
    googlenet = torchvision.models.googlenet(pretrained=True)
    resnet = torchvision.models.resnet50(pretrained=True)
    densenet = torchvision.models.densenet161(pretrained=True)

    densenet.classifier = nn.Linear(2208,num_classes)
    resnet.fc = nn.Linear(2048,num_classes)
    googlenet.fc = nn.Linear(1024,num_classes)
    densenet = densenet.to(device)
    resnet = resnet.to(device)
    googlenet = googlenet.to(device)

    return [densenet,googlenet,resnet]
```

Here is the learning process of the GoogleNet model. This model achieved an accuracy of 97.30% during the fine-tuning. Although the paper has mentioned 96.29% as the achieved accuracy, as mentioned before, this variance might be a result of different initialization and settings.



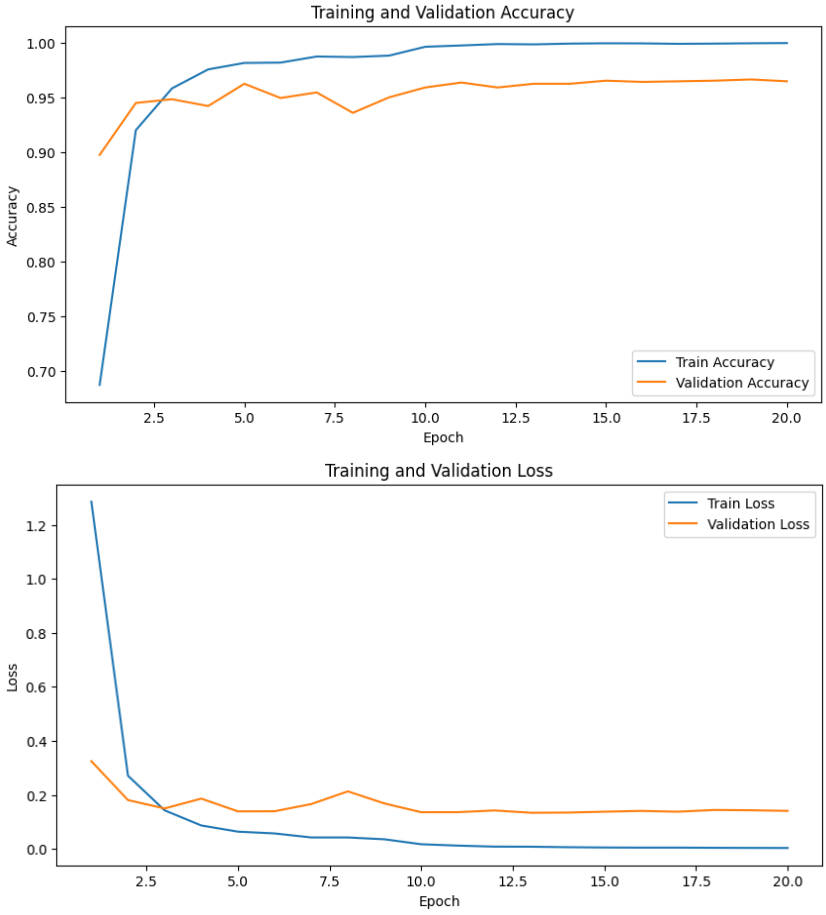
The training process is completed within 19m 43s using the GPU processor of the Google Colab.

```
Epoch 1/20
----------
train Loss: 0.4785 Acc: 0.8674
val Loss: 0.3318 Acc: 0.8704
…
Epoch 19/20
----------
train Loss: 0.0011 Acc: 1.0000
val Loss: 0.1228 Acc: 0.9734

Epoch 20/20
----------
train Loss: 0.0010 Acc: 0.9999
val Loss: 0.1176 Acc: 0.9751

Best val Acc: 0.976231
Final Accuracy:  tensor(0.9730, device='cuda:0')
```

The DenseNet161 model's learning process is presented here. The model attained a 96.21% accuracy during fine-tuning, while the reported accuracy in the paper was 96.10%. As previously stated, this difference could be attributed to distinct initialization and settings.



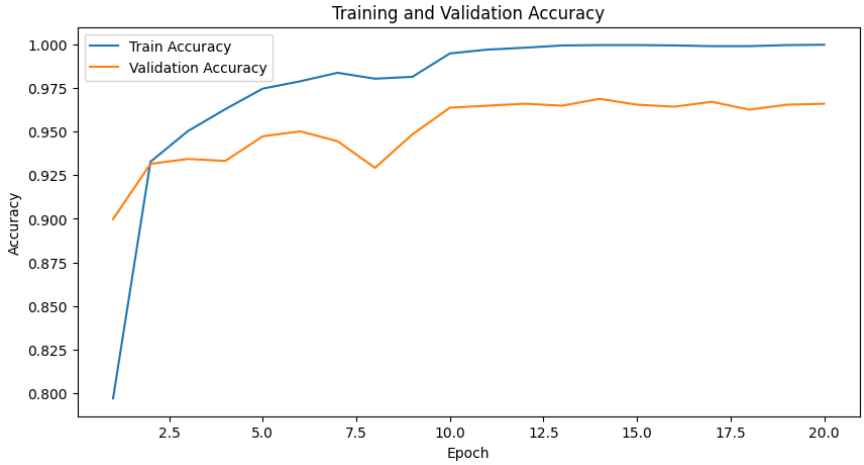
The training process is completed within 6m 31s using the GPU processor of the Google Colab.

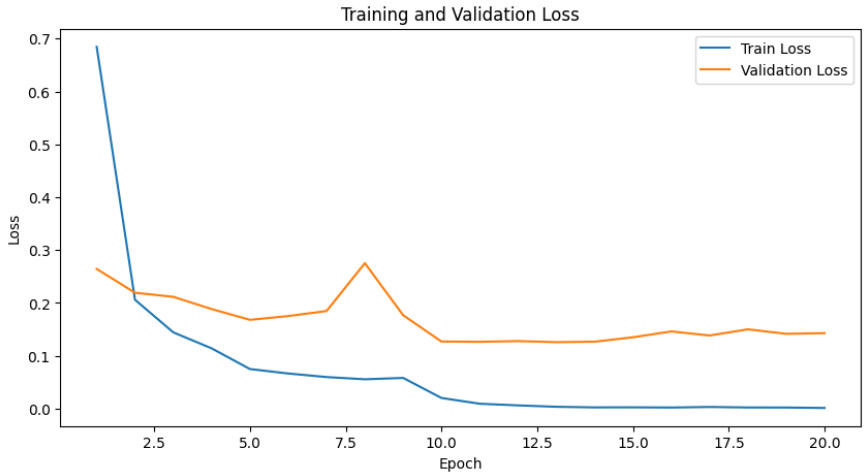
```
Epoch 1/20
----------
train Loss: 1.2853 Acc: 0.6871
val Loss: 0.3245 Acc: 0.8976
…
Epoch 19/20
----------
train Loss: 0.0035 Acc: 0.9997
val Loss: 0.1429 Acc: 0.9666

Epoch 20/20
----------
train Loss: 0.0033 Acc: 0.9999
val Loss: 0.1406 Acc: 0.9649

Best val Acc: 0.966610
Final Accuracy:  tensor(0.9621, device='cuda:0')
```

The learning process of the ResNet50 model is outlined in this section. During fine-tuning, the model reached an accuracy of 96.88%, whereas the accuracy reported in the paper was 96.00%. As previously noted, this variation might be due to varying initialization and settings.





The training process is completed within 11m 10s using the GPU processor of the Google Colab.

```
Epoch 1/20
----------
train Loss: 0.6845 Acc: 0.7969
val Loss: 0.2644 Acc: 0.8998
…
Epoch 19/20
----------
train Loss: 0.0023 Acc: 0.9997
val Loss: 0.1416 Acc: 0.9655

Epoch 20/20
----------
train Loss: 0.0015 Acc: 0.9999
val Loss: 0.1430 Acc: 0.9660

Best val Acc: 0.968874
Final Accuracy:  tensor(0.9674, device='cuda:0')
```

5. Simulating the paper titled "An ensemble of deep transfer learning models for handwritten music symbol recognition"

In this section, we continue our procedure of simulating the paper [5]. Here, we use the pre-trained models in Section 5 and aggregate their results by applying a Support Vector Machine (SVM) to their outputs; rbf is chosen to be the kernel of the SVM. This ensemble technique aims to enhance the overall performance and robustness of the recognition system. Through this simulation, we aim to evaluate the effectiveness of the ensemble method and compare our findings with the reported results in the original paper.

```
Ensemble Accuracy:  0.9772652388797364
```

As can be seen, the achieved accuracy is about 97.72% which is slightly higher than the 97.32% achieved by the authors.

In order to enhance the results, we continue our investigation around the acquired ensemble model. In the paper, three other ensemble methods namely sum rule, product rude, and NN ensemble are surveyed; however, the accuracy in these models is lower than the proposed ensemble. The achieved accuracies are 96.93%, 96.45%, and 96.96% respectively.

In the next section, we survey other possible methods for aggregating the results hoping to achieve better accuracy than is mentioned in the paper.

6. Enhancing the Presented Method

In this section, we examine other techniques for aggregating the outputs of these three models. Subsequently, we introduce a two-level hierarchy ensemble method.

The decision tree is one of the approaches to aggregating the outputs of the models. It is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. The achieved accuracy through this method is 96.24%.

```
Decision Tree-based Ensemble Accuracy:  0.9624382207578254
```

Another approach is random forest. It is a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. The achieved accuracy through this method is 97.85% which is slightly better than the SVM approach.

```
Random Forest-based Ensemble Accuracy:  0.9785831960461285
```

As the third method, we tested the XGboost method. XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an

ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. The achieved accuracy through this method is 96.37%.

```
XGboost-based Ensemble Accuracy:  0.9637561779242174
```

Another approach is naïve Bayes. It is a probabilistic algorithm based on Bayes' theorem. It assumes that the features are conditionally independent given the class label. The accuracy is achieved at 97.42%.

```
Naive Bayes-based Ensemble Accuracy:  0.9742998352553542
```

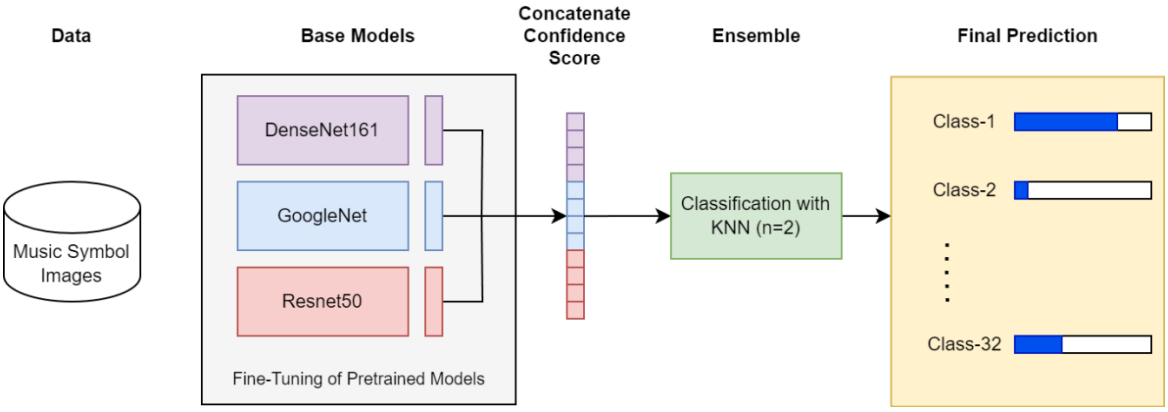
Now, we apply the Adaboost technique. Also called Adaptive Boosting, it is a technique in Machine Learning used as an Ensemble Method. The most common estimator used with AdaBoost is decision trees with one level which means Decision trees with only 1 split. The achieved accuracy through this method is 96.14%.

```
Adaboost-based Ensemble Accuracy:  0.9614497528830313
```

KNN is a simple and intuitive algorithm where the class of a data point is determined by the majority class among its K-nearest neighbors in the feature space. The achieved accuracy through this method is 97.69%.

```
KNN(n=2)-based Ensemble Accuracy:  0.9769357495881383
```

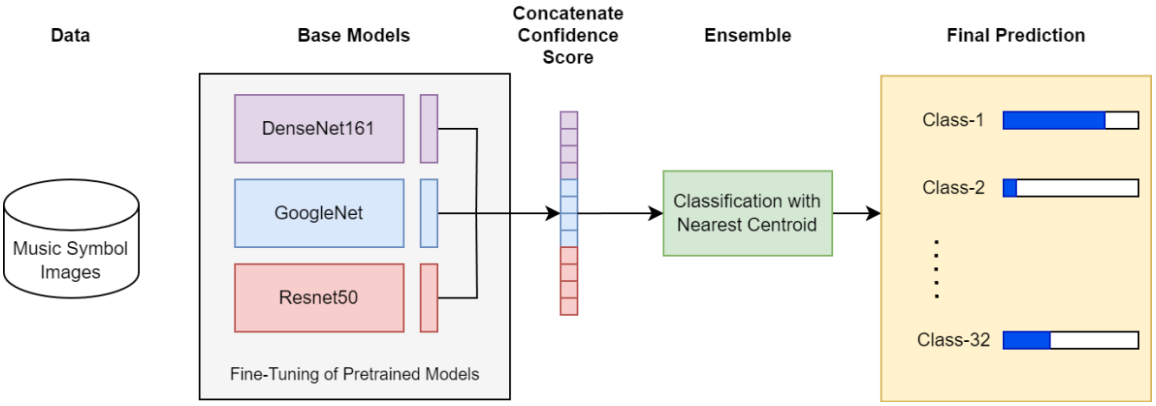
The diagram of this model is illustrated below:



Finally, we apply the Nearest Centroid Classifier. This is a simple and lightweight algorithm that uses the centroid (mean) of each class to make predictions. The achieved accuracy through this algorithm is 97.72%.

```
Nearest Centroid-based Ensemble Accuracy:  0.9772652388797364
```

The diagram of the Nearest Centroid ensemble model can be found below:



In the table below, we summarize the achieved accuracies by using these methods:

```
Ensemble Accuracy:  0.9772652388797364
Decision Tree-based Ensemble Accuracy:  0.9624382207578254
Random Forest-based Ensemble Accuracy:  0.9785831960461285
XGboost-based Ensemble Accuracy:  0.9637561779242174
Naive Bayes-based Ensemble Accuracy:  0.9742998352553542
Adaboost-based Ensemble Accuracy:  0.9614497528830313
KNN(n=2)-based Ensemble Accuracy:  0.9769357495881383
Nearest Centroid-based Ensemble Accuracy:  0.9772652388797364
```

However, while the free Google Colab has restrictions, our runtime exceeded the limits and thus, our results were lost. From this point, we consider our findings in the second run which can be summarized as follows:

```
Ensemble Accuracy:  0.9789126853377266
Decision Tree-based Ensemble Accuracy:  0.9525535420098846
Random Forest-based Ensemble Accuracy:  0.9769357495881383
XGboost-based Ensemble Accuracy:  0.9683690280065897
Naive Bayes-based Ensemble Accuracy:  0.9739703459637562
Adaboost-based Ensemble Accuracy:  0.9542009884678748
KNN(n=2)-based Ensemble Accuracy:  0.9831960461285009
Nearest Centroid-based Ensemble Accuracy:  0.9795716639209225
```

Now, instead of applying the SVM to the three base models, we apply the SVM to these eight second-level models in order to initial a hierarchy of ensemble models. However, the SVM module does not result in an increase in accuracy. The achieved accuracy through this method is 97.79% which is almost equal to the achieved one by the authors.

```
Ensemble Accuracy:  0.9779242174629325
```

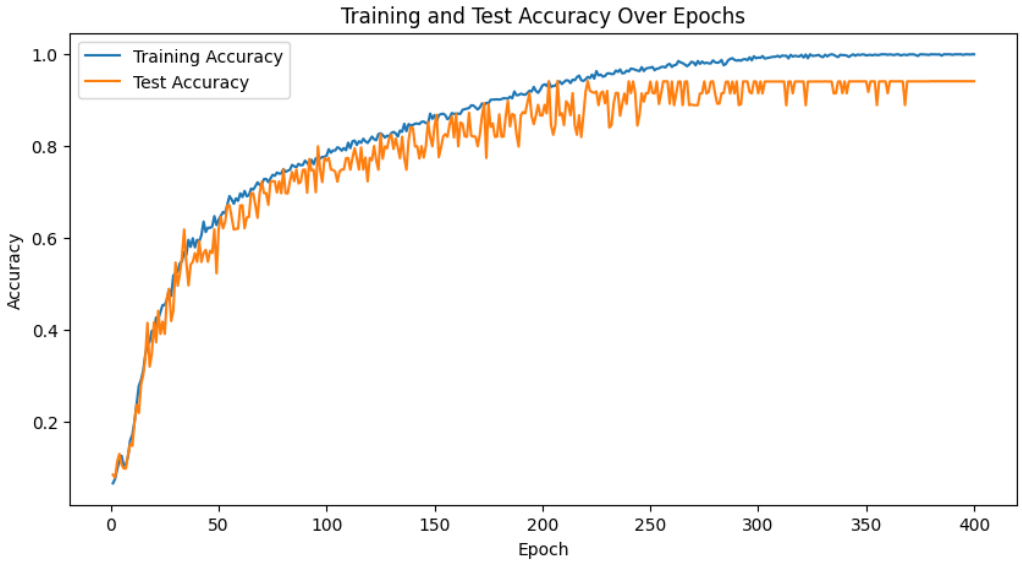
We also, try other ensemble methods in order to come up with an architecture to exceed the paper in accuracy. We, now, try a simple 2-layer neural network with the structure below:

```
self.fc1 = nn.Linear(input_dim, 128)
self.fc2 = nn.Linear(128, 64)
self.fc3 = nn.Linear(64, output_dim)
```

And with the below optimizer and scheduler:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=100, gamma=0.75)
```

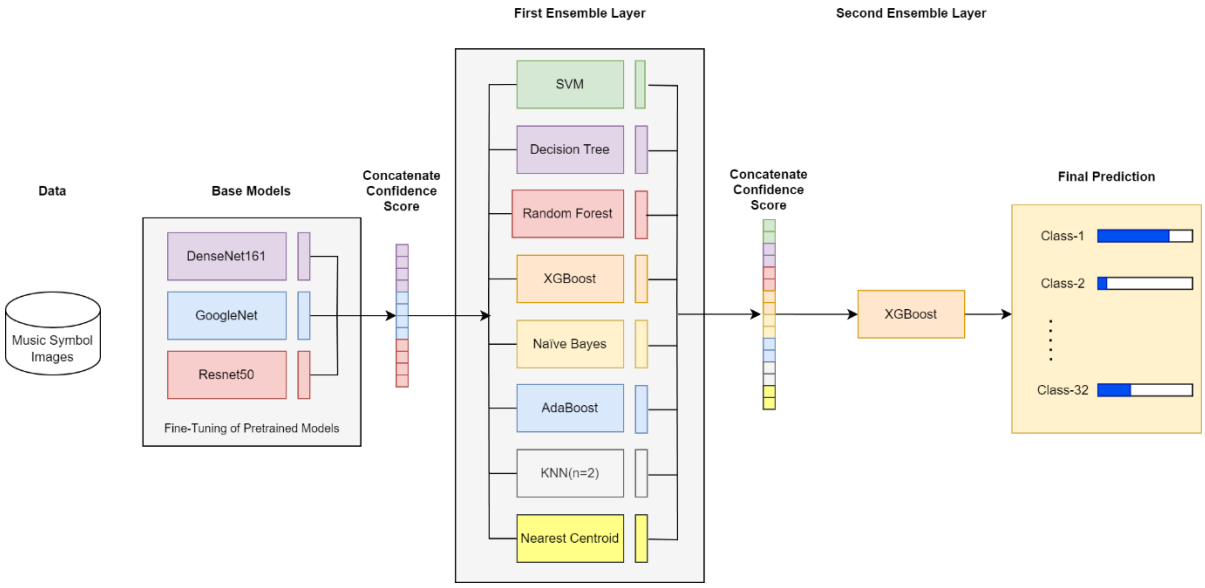
This method, however, is not successful. The achieved accuracy in the test set is 94.14%. The training process is illustrated below:



Moreover, we tested all of the aforementioned ensemble methods to establish a two-level hierarchy ensemble method. However, achieved accuracies fall short in comparison to the previously surveyed one. Here, we list the achieved accuracies:

```
Ensemble Accuracy:  0.9779242174629325
2-layer Neural Network Accuracy: 0.941414141414
Decision Tree-based Ensemble Accuracy:  0.9693574958813839
Random Forest-based Ensemble Accuracy:  0.9779242174629325
XGboost-based Ensemble Accuracy:  0.9789126853377266
Naive Bayes-based Ensemble Accuracy:  0.9476112026359144
Adaboost-based Ensemble Accuracy:  0.9584113450463545
KNN(n=2)-based Ensemble Accuracy:  0.942667846128509
Nearest Centroid-based Ensemble Accuracy:  0.9436579110378381
```

As can be seen, one of the methods can capture a better accuracy than the presented model. This method, as said, is based on a hierarchy of ensemble methods. The figure below illustrates the hierarchical ensemble method:



7. Discussion

In this table, we summarized all of the models with their achieved accuracies.

Model Name	Category	Accuracy
CNN	Deep Learning	89.41%
CNN + Augmentation	Deep Learning	92.92%
DenseNet	Transfer Learning	97.30%
GoogleNet	Transfer Learning	96.21%
Resnet	Transfer Learning	97.74%
SVM	Ensemble	97.89%
Decision Tree	Ensemble	95.25%
Random Forest	Ensemble	97.69%
XGboost	Ensemble	96.83%
Naïve Bayes	Ensemble	97.39%
AdaBoost	Ensemble	95.42%
KNN(n=2)	Ensemble	98.31%
Nearest Centroid	Ensemble	97.95%
SVM	2-Layer Ensemble	94.14%
Neural Network	2-Layer Ensemble	97.79%
Decision Tree	2-Layer Ensemble	96.93%
Random Forest	2-Layer Ensemble	97.79%
XGboost	2-Layer Ensemble	97.89%
Naïve Bayes	2-Layer Ensemble	94.76%
AdaBoost	2-Layer Ensemble	96.84%
KNN(n=2)	2-Layer Ensemble	94.26%
Nearest Centroid	2-Layer Ensemble	94.36%

The Proposed method in the paper is highlighted in yellow, while the methods with higher accuracy proposed by us are highlighted in green. As can be seen, three of our methods excel the proposed architecture slightly. Moreover, the models that achieved better accuracy are illustrated in the report using https://app.diagrams.net/.

Note that there are 3 Python files accompanying this report. One of them is related to the dataset initialization. The other one is related to the CNN and the augmentation method. And the last one is related to transfer learning and ensemble methods.

References

[1] Nawade, Savitri Apparao, et al. "Old handwritten music symbol recognition using directional multi-resolution spatial features." 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE). IEEE, 2018.

[2] Fornés, Alicia, Josep Lladós, and Gemma Sánchez. "Old handwritten musical symbol classification by a dynamic time warping based method." International Workshop on Graphics Recognition. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[3] "OMR-Datasets." OMR-Datasets, apacha.github.io/OMR-Datasets/.

[4] Shatri, Elona, and György Fazekas. "DoReMi: First glance at a universal OMR dataset." *arXiv preprint arXiv:2107.07786* (2021).

[5] Paul, Ashis, et al. "An ensemble of deep transfer learning models for handwritten music symbol recognition." *Neural Computing and Applications* 34.13 (2022): 10409-10427.