Department of Computer Engineering

CS464 Term Project

Section 2 - Group 19

Fall / 2019

Listen to What Poster Says: Music Generation Based on Movie Posters

Final Report

Team

Talha Şen 21702020
Hakan Sivuk 21601899
Gonca Yılmaz 21702727
Cevat Aykan Sevinç 21703201
Mustafa Çağrı Güngör 21602251

Instructor: Shervin Rahimzadeh Arashloo

Teaching Assistant: Oğuzhan Karakahya

Table of Contents

1. Introduction

The aim of our project is to, first, predict the genre of a movie poster, then, produce a suitable soundtrack for the predicted genre. Because of this, our project has a machine learning pipeline. Our project will have two machine learning models. The first model will take a movie poster image as an input and the output of the model would be its predicted genre. This output will be the input of our second model which is for producing a soundtrack. Consequently, the predicted soundtrack depicts a prediction of movie poster's soundtrack according to its genre.

Deep learning is used to perform operations mentioned above. By using deep learning models, we are able to form complex models that can make accurate predictions.

CNN is used for predicting movie poster genre in our project. For this, the team has found the movie poster dataset from kaggle [1]. There are 5 genres for our movie poster dataset: action, animation, comedy, drama, horror. In the same order, we have 4412, 1545, 9136, 9297, 1859 images for each genre. The images are 182 by 268 pixels jpeg files and in RGB format. We used customly created model as our CNN architecture and we used several optimizers but we got the best result from the Adam optimizer. We split 20% of the data as test set and rest for training and validation sets. We train our model and then using the training set we predict the category of the test film. To prevent overfitting of action, comedy and drama posters, as they have much higher data than other categories, we normalize the categories by cutting some of the action, drama and comedy. This way we provided variation in different kind of film categories.

LSTM is used for producing soundtrack in our project. The dataset consists of 5 genre files introduced in CNN model. The team has found 112 action, 52 animation, 124 comedy, 88 drama, 27 horror soundtracks respectively for each genre in order. The soundtrack files are in MIDI format and include only a single instrument. Our LSTM model has been trained with these 5 categories to have 5 models for each genre. Depending on the input genre, the regarding model is used to produce the soundtrack. As a result of these processes, 5 LSTM models for 5 different categories were obtained. Best output results were obtained from models that fit the data perfectly, therefore models were trained until error function takes low values. At the same time, in order to prevent overfitting to particular melodies (In the demo, our action LSTM model overfitted to the soundtrack of Pirates of Caribbean), datasets were adjusted carefully to provide variation of different kinds of melodies.

2. Problem Description

Lots of movies are remembered with their soundtrack music such as Titanic, Lord of the Rings and so on. They contain important components from the movie like emotions, theme, tempo. On the other hand, posters are important presentations of movies. One can see important clues about the movie from its poster. As movie posters and soundtracks are taken into consideration, it can be seen that both include important information in terms of genre of the movie. Therefore, we wanted to investigate whether a connection can be formed between poster and soundtrack of a movie. So, our main question is whether soundtrack music can be generated by just looking poster of a movie.

To address this question, first, we need to infer a meaning from a poster. Even for a human, trying to analyze the category of the film from its poster may be difficult, especially nowadays where film posters contain so many characters, details, objects and information about the film but each category still contains some unique labels that may be used for differentiating it from other categories. This is why we use CNN and try to identify these unique parts of the poster which may be used to determine categories of other films. To predict a film’s category, we definitely need to identify the color palettes of the poster, distribution of the poster elements throughout the image, text fonts used etc. This is what we are doing with our CNN model to predict the category of a movie. We analyze the image, distribute its elements and try to catch a pattern from these distributions. When we catch a unique and universal pattern, we used this for prediction of a movie category from its poster.

After determining the genre of a movie from a poster, we should generate a soundtrack for this genre. Therefore, in this part of the project, we tried to address how we can generate soundtracks for a particular genre that are original but cannot be distinguishable from music composed by a human. Producing music in real life is somehow selecting notes by evaluating previous notes. Also, crucial component of a piece of music is harmony of its elements. In soundtrack generation part, our main problem is to come up with a model that can produce a soundtrack whose notes are in harmony with each other. Apart from that, a music may contain chords which is union of several different notes. Therefore, to produce a music, our model should also decide between single notes and chords as well as it decides between notes for each prediction. Those are the fundamental needs of the model to produce soundtrack music as if it is written by a composer. As it is mentioned above, beside generating music with our models, generated soundtrack should also belong to particular genre.

3. Methods

3.1 Poster Recognition

3.1.1 Dataset

Dataset is created using IMDB website and dataset includes 4412 action, 1545 animation, 9136 comedy, 9297 drama, 1859 horror images. Due to avoiding overfitting, number of images for each genre were kept close to each other. Final dataset includes 1445 action, 1688 animation, 1700 comedy, 1335 drama, 1243 horror with a total of 7254 images. The dataset is split as train (5928), validation (1536) and test (1483). The images are resized as 48x48 pixel, then considering power of GPU and getting probability of high accuracy, preprocessing is repeated while resizing images as 96x96 pixels.

3.1.2 Data Manipulation Techniques

3.1.2.1 Data Augmentation

One of the most important techniques of data manipulation is data augmentation in the CNN projects. It reproduces your train set to get higher accuracy and it helps to avoid overfitting. The script below is used to execute data augmentation.

```
from keras.preprocessing.image import ImageDataGenerator
data_generator = ImageDataGenerator(
                    width_shift_range=0.1,
                    height_shift_range=0.1,
                    zoom_range=0.2,
                    shear_range=0.1,
                    rotation_range=10)
data_generator.fit(train_features)
```

3.1.2.1 Applying PCA on Training Set

The movie poster dataset is hard to train for getting high accuracy. Because naturally, classifying genres of movies according to their posters is not an easy task even for humans. The dataset contains lots of exceptions such as there can be gun that recalls action movies in the poster of animation movies or the poster of action movie may contain colorful poster which recalls animation movies. Briefly, the features in the dataset are not obvious and due to the fact that it may contain noises. It is thought that applying PCA on the training set may reduce the noise and using important features could increase accuracy and other metrics. The python script below is used to apply PCA with 50 Cumulative PCA on training set.

```
from numpy.linalg import svd
def pca_manual(X):
  N = len(X)
  X = X - X.mean(axis=0) # Mean normalization
  # Covariance Matrix Process
  X = X.T
  Y = X.dot(X.T)/N
  U,S,V = svd(Y) # svd turns u,s,v values of array.
  print("Completed")
  return U,S,V
def reconstruct(u1,u2,u3,r,g,b):
  # The loop reconstruct and display images.
  i = 50
  reconstructed_r_flatten = (r.dot(u1[:,0:i+1])).dot(u1[:,0:i+1].T)
  reconstructed_g_flatten = (g.dot(u2[:,0:i+1])).dot(u2[:,0:i+1].T)
  reconstructed_b_flatten = (b.dot(u3[:,0:i+1])).dot(u3[:,0:i+1].T)
  # Combined to r,g,b values
  new_image = np.ones((d2*d3,d4),'uint8')
  new_image[:,0], new_image[:,1], new_image[:,2] =
reconstructed_r_flatten, reconstructed_g_flatten,
reconstructed_b_flatten
  new_image = new_image.reshape((d2,d3,d4))
  return new_image


d1, d2, d3, d4 = train_features.shape
X = train_features.reshape((d1, d2*d3,d4))
r, g, b    = X[:, :, 0], X[:, :, 1], X[:, :, 2]

# U,S,V values are taken for all color channels.
u1,s1,v1 = pca_manual(r)
u2,s2,v2 = pca_manual(g)
u3,s3,v3 = pca_manual(b)
```

```
# The methods reconstructs given image by using 1,50,250,500,4096
components and displays them.
train_features = reconstruct(u1,u2,u3,r,g,b)
```

3.1.2 Model

The dataset is trained using different types of models such as LeNet, the models inspired from other CNN projects which was optimized after many attempts(Custom Model), Rasnet20.

The model plot of LeNet:

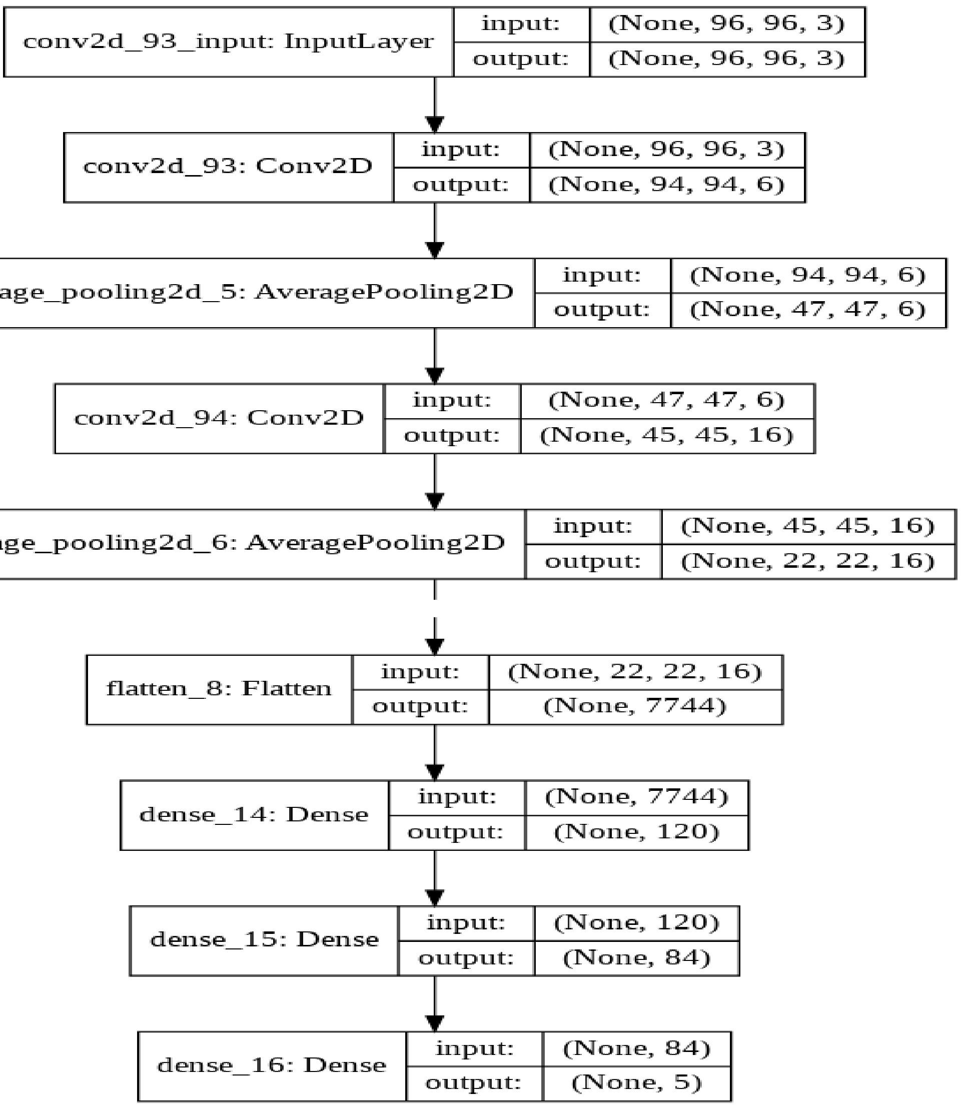


Figure 1: The model plot of LeNet.

Due to the sizes of diagrams of Custom model and Rasnet20, their png file links are shared in above.

The model plot of Custom Model can be reached in this link:
model_plot_custom1.png

The model plot of RasNet20 can be reached in this link:
model_plot_rasnet20.png

3.2 Soundtrack Generation

3.2.1 Dataset

Our dataset is for generating a soundtrack for a particular movie genre. It consists of different genre folders that have movie soundtracks belonging to that genre. Dataset has 5 distinct genres which are the same with our first dataset. These categories have 120,000 action, 6600 comedy, 10,000 horror, 6000 drama, 22,203 animation notes respectively. Soundtracks are in midi format and include only one instrument which holds the main theme of the music. To form this dataset, we have collected soundtrack in midi format for all these genres from different websites such as ‘Midi Movie Theme Songs’ [2]. After that, we have edited them to include only a single instrument with the main melody of the soundtrack for each midi file. Having completed this process, we have formed our dataset for the second part of the project.

3.2.2 Tools

- **Keras**

Keras is a Tensorflow library. It allows easy construction and training of RNN layers [3]. Since Keras library allows customization on the model, the team will try to optimize the model for our MIDI file dataset using this library.

- **Music21**

Music21 library is included in Python library for processing the music structure. MIDI files consist of chords and note denotations [4]. Music21 library can create and manipulate MIDI files using note and chord objects. The note objects contain three kinds of information about a note as octave, offset, and pitch. The chord object holds information about set of notes played simultaneously. Firstly, Music21 library is used to open all MIDI files in the dataset and create a sequence of musical notations consisting of notes and chords. Then, this dataset is given as the training input of the model. Secondly, Music21 library is used again to produce a MIDI file. When the model predicts a music sequence for the given genre, the sequence output consists of chords and notes. Music21 library processes the sequence and creates a MIDI file for the objects in the order of the output sequence.

3.2.3 Training The Model

Recurrent Neural Networks (RNN)

Recurrent Neural Networks are used to process sequences of data to predict time series or natural language. It keeps an internal record of timesteps of experienced sequences while processing the timesteps of a sequence [3]. This way, RNN can predict future occurrences of timesteps in a data sequence. Long Short-Term Memory (LSTM) is used for the model layer as it can efficiently perform sequence modelling with gradient descent. LSTM can remember long sequences of data and since our project processes music, LSTM can keep track of the music sequences.

When the previous works and projects about music production by using deep learning are investigated thoroughly, two important projects are found which we use as a guidance. Both of the projects use LSTMs to produce music due to the meaning of the LSTM as it is discussed above. Although these projects help us to determine the path we follow, we try to put a unique work forth to get better results. Therefore the project we developed differ from the other two projects by some parts.

We have implemented different models with different layer combinations by using RNN. We have bidirectional LSTM layers, dropout layers, dense layers, softmax activation layers. Also, different normalizations and optimizers are used to obtain the best result.

Layers we have used:

- Dense layers are regular fully connected layers.
- Unidirectional LSTM layers can only use the input from the past however bidirectional LSTM layers can use the input from the future as well as the past. It allows bidirectional LSTM layer to understand the context better, which is very beneficial in the context of producing music [5]. We are currently using 512 BLSTM layers. These layers get a sequence as an input and returns either a sequence or a matrix. We have used sequence outputs in our project.
- Dropout layers drop some fraction of the neurons in the layer randomly during the training time. This regularization process can be seen as teaching data to each of the activation unit equally. It helps to prevent overfitting [6]. The fraction parameter we have used in the project is 0.3.

- Softmax activation layers make sum of outputs of each class in the model to be 1 which makes the output numbers interpretable in terms of probability [7].

Normalization techniques and optimizers we used:

- Batch normalization layer adjusts and scales neurons in its input layer. Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift) which allows model to generalize its evaluation of the data. It gives better learning rates by ensuring that there is no activation that is too high or too low and also it reduces the overfitting [8].
- Rmsprop is a technique for scaling the learning rate during the training time. It is an optimization technique for the gradient descent algorithm which makes the algorithm much faster [9].

Loss function we have used:

- Categorical cross entropy loss function is used as a loss function. Categorical cross entropy (or Softmax loss) decreases as the predicted probability of the sample converges to the actual value [10]. It is used for comparing the distribution of predictions with the actual distributions.

As the RNN team, we have decided to first test our action MIDI dataset on a LSTM network. Consequently, we have used an LSTM model (will be referred as model A) with the following layer configuration and checkpoint:

```
model = Sequential()
model.add(LSTM(
        512,
        input_shape=(network_input.shape[1],
network_input.shape[2]),
        recurrent_dropout=0.3,
        return_sequences=True
    ))
    model.add(LSTM(512, return_sequences=True,
recurrent_dropout=0.3,))
    model.add(LSTM(512))
    model.add(BatchNorm())
    model.add(Dropout(0.3))
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(BatchNorm())
    model.add(Dropout(0.3))
    model.add(Dense(n_vocab))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy',
optimizer='rmsprop')

    checkpoint = ModelCheckpoint(
        filepath,
        monitor='loss',
        verbose=0,
        save_best_only=True,
        mode='min') [11]
```

The Model checkpoint allowed us to save the models with best loss. Consequently, we were able to stop and reload models at any instant during training. Since an epoch took around 30 minutes to complete, this was an important factor for continuously training our model.

However, when we had inserted our action genre MIDI files as the input, the monitored loss was around 11.10. When we produced a MIDI file with this loss, the outcome was meaningless. Only a single note was played during the whole sequence. As a team, we interpreted this loss and output as a mean of bad input. Since we had prepared our action MIDI data by hand, our data may not be the most ideal MIDI configuration. Because of this, we had decided to use a MIDI dataset of an existing project to test the output of the data. When we had trained the same model with a professionally crafted MIDI dataset, the monitored loss was around 4.7 and after 14 epochs, it did not improve. After this, we thought if there was a problem within the model itself. For this, we analyzed the weights of the prediction layer and the note file consisting of note and chard objects. Weights were not 0 and note file was successfully parsed with Music21. After this, we experimented with different batch sizes of 128, 512, 2048 to observe the loss but the results were the same. Consequently, we decided to compare our model with a model used in another project that was inspired from our initial model.

After we analyzed the other project, we changed our layers according to this structure without changing the checkpoint configuration (will be referred as model B):

```
model = Sequential()
    model.add( Bidirectional( LSTM( 512,
return_sequences=True),
                 input_shape=(network_input.shape[1],
network_input.shape[2]),
           ) )
```

```
        model.add( Dropout( 0.3))
        model.add( Bidirectional( LSTM( 512)))
        model.add( Dense( n_vocab))
        model.add( Activation( "softmax"))
        model.compile(loss='categorical_crossentropy',
    optimizer='rmsprop')
```
[12]

Previously, we could not manage to produce a meaningful soundtrack. Using this structure, after 30 epochs with a batch size of 64, we monitored a loss of 0.36. Finally, using the model with this loss, we managed to produce a meaningful MIDI sequence as the output. Nevertheless, the output was not from our MIDI dataset. On the other hand, we observed that the model can be successfully trained with this layer configuration.

After generating meaningful output with other project's dataset, we moved again on our dataset. Due to the fact that the number of the action notes is larger compared to others and it is easy to distinguish action soundtracks from other, we started with the action dataset. This dataset included 12.000 notes. We trained our model with this dataset until seeing 0.0143 as loss value. When we generated soundtrack with this trained model, we get quite meaningful output file. Although this output file gives the impression of action genre, all generated soundtracks include the same melody (Pirates of Caribbean) somewhere in them. We concluded this situation as overfitting of the model to Pirates of Caribbean. When we examined the dataset and saw the number of notes in Pirates of Caribbean is very much larger than others. It confirmed our assumption that it dominates others.

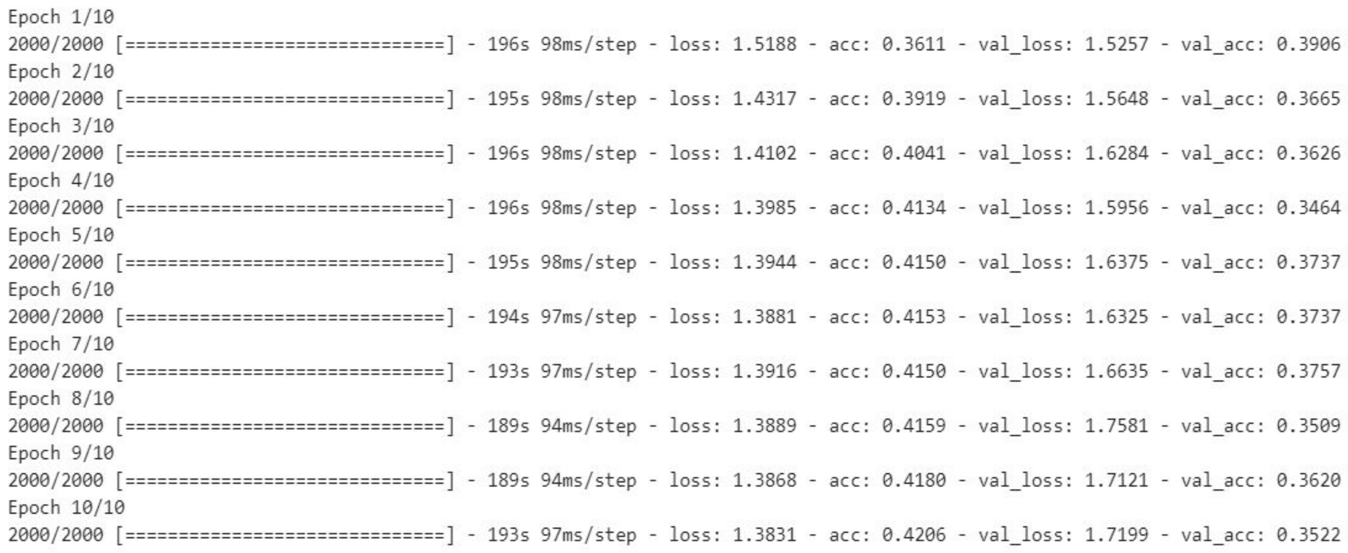
After realizing this situation, we decided to prepare the dataset more carefully. We tried to add balanced number of notes from each different melody. Changed version of dataset includes 10.000 notes. Although it is smaller than the previous number of total notes, when we trained and generate outputs, we got unique and original outputs which cannot be distinguished from music composed by a composer and also can be differentiated from each other and music from the dataset.

When we saw this successful outputs, we decided to follow a similar process for other genres. We prepared the dataset for each of the genres in a similar way and we trained our model with these datasets and the same layer configuration. As it is expected, we got successful results. Furthermore, for action dataset, using the same model, we increased the total number of notes to 120,000 to check how number of notes perform against overfitting. Next, we changed the configuration of our model A to BLSTM to create a model C and tested this model on the comedy dataset.

4. Results

4.1 Poster Recognition

4.1.1 LeNet Model Results



```
Epoch 1/10
2000/2000 [==============================] - 196s 98ms/step - loss: 1.5188 - acc: 0.3611 - val_loss: 1.5257 - val_acc: 0.3906
Epoch 2/10
2000/2000 [==============================] - 195s 98ms/step - loss: 1.4317 - acc: 0.3919 - val_loss: 1.5648 - val_acc: 0.3665
Epoch 3/10
2000/2000 [==============================] - 196s 98ms/step - loss: 1.4102 - acc: 0.4041 - val_loss: 1.6284 - val_acc: 0.3626
Epoch 4/10
2000/2000 [==============================] - 196s 98ms/step - loss: 1.3985 - acc: 0.4134 - val_loss: 1.5956 - val_acc: 0.3464
Epoch 5/10
2000/2000 [==============================] - 195s 98ms/step - loss: 1.3944 - acc: 0.4150 - val_loss: 1.6375 - val_acc: 0.3737
Epoch 6/10
2000/2000 [==============================] - 194s 97ms/step - loss: 1.3881 - acc: 0.4153 - val_loss: 1.6325 - val_acc: 0.3737
Epoch 7/10
2000/2000 [==============================] - 193s 97ms/step - loss: 1.3916 - acc: 0.4150 - val_loss: 1.6635 - val_acc: 0.3757
Epoch 8/10
2000/2000 [==============================] - 189s 94ms/step - loss: 1.3889 - acc: 0.4159 - val_loss: 1.7581 - val_acc: 0.3509
Epoch 9/10
2000/2000 [==============================] - 189s 94ms/step - loss: 1.3868 - acc: 0.4180 - val_loss: 1.7121 - val_acc: 0.3620
Epoch 10/10
2000/2000 [==============================] - 193s 97ms/step - loss: 1.3831 - acc: 0.4206 - val_loss: 1.7199 - val_acc: 0.3522
```

Figure 2: The training steps of dataset with LeNet model.

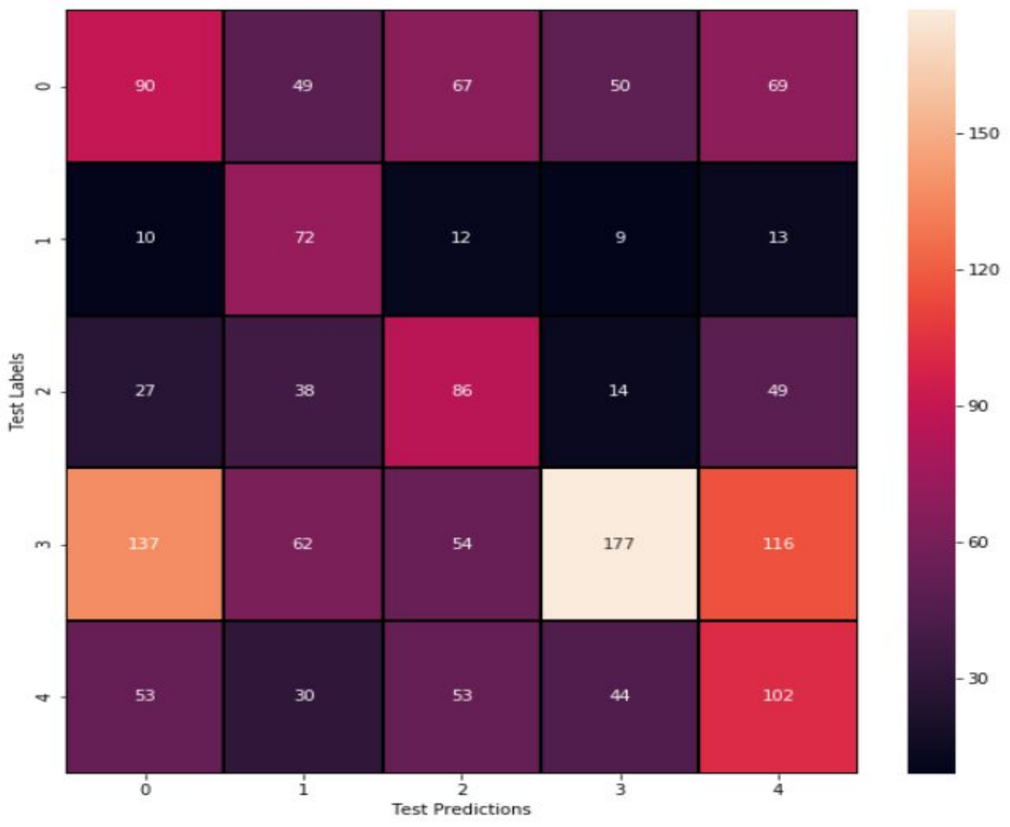
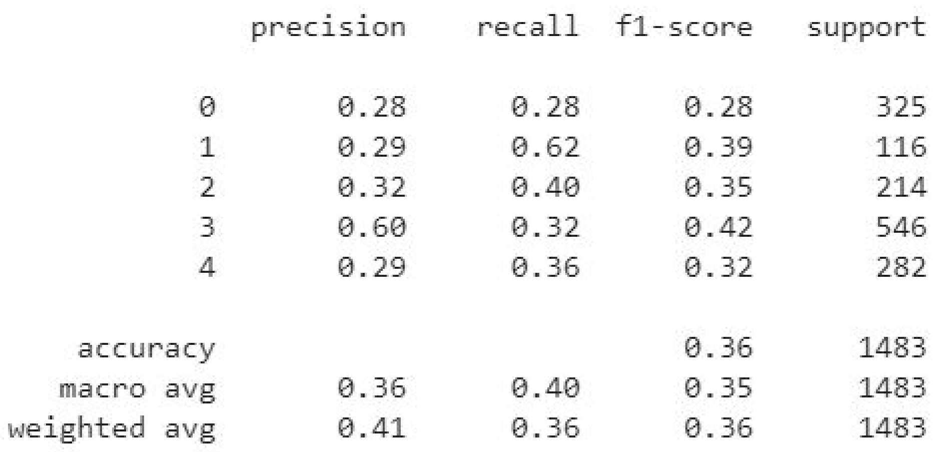


Figure 3: The confusion matrix of the labels and predictions with LeNet model.



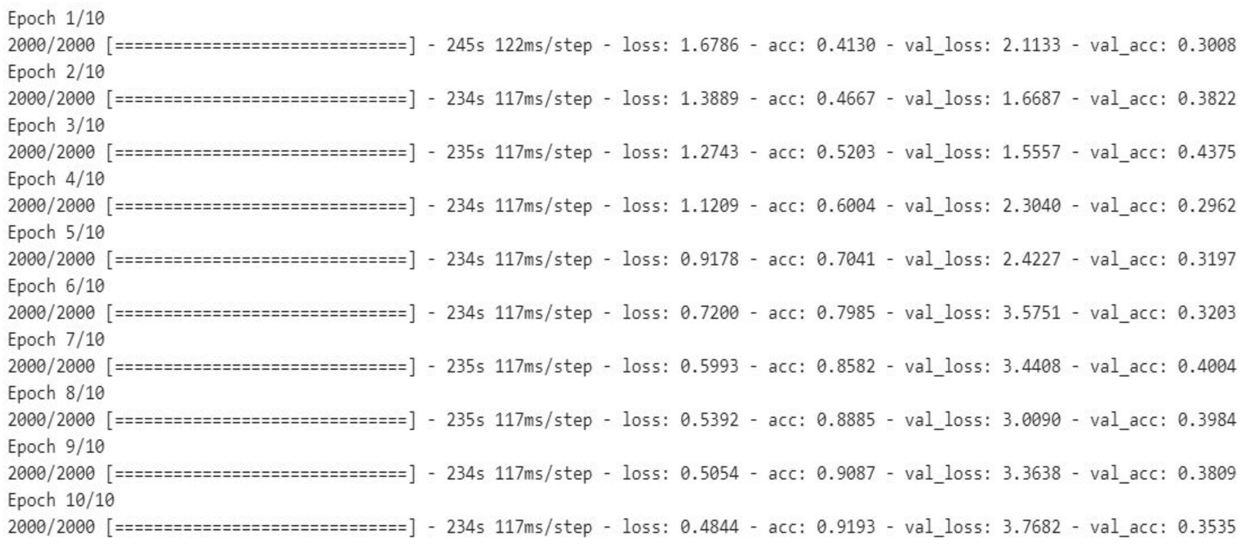
```
              precision    recall  f1-score   support

           0       0.28      0.28      0.28       325
           1       0.29      0.62      0.39       116
           2       0.32      0.40      0.35       214
           3       0.60      0.32      0.42       546
           4       0.29      0.36      0.32       282

    accuracy                           0.36      1483
   macro avg       0.36      0.40      0.35      1483
weighted avg       0.41      0.36      0.36      1483
```

Figure 4: Different metrics in addition to accuracy with LeNet model.

4.1.2 RasNet20 Model Results



```
Epoch 1/10
2000/2000 [==============================] - 245s 122ms/step - loss: 1.6786 - acc: 0.4130 - val_loss: 2.1133 - val_acc: 0.3008
Epoch 2/10
2000/2000 [==============================] - 234s 117ms/step - loss: 1.3889 - acc: 0.4667 - val_loss: 1.6687 - val_acc: 0.3822
Epoch 3/10
2000/2000 [==============================] - 235s 117ms/step - loss: 1.2743 - acc: 0.5203 - val_loss: 1.5557 - val_acc: 0.4375
Epoch 4/10
2000/2000 [==============================] - 234s 117ms/step - loss: 1.1209 - acc: 0.6004 - val_loss: 2.3040 - val_acc: 0.2962
Epoch 5/10
2000/2000 [==============================] - 234s 117ms/step - loss: 0.9178 - acc: 0.7041 - val_loss: 2.4227 - val_acc: 0.3197
Epoch 6/10
2000/2000 [==============================] - 234s 117ms/step - loss: 0.7200 - acc: 0.7985 - val_loss: 3.5751 - val_acc: 0.3203
Epoch 7/10
2000/2000 [==============================] - 235s 117ms/step - loss: 0.5993 - acc: 0.8582 - val_loss: 3.4408 - val_acc: 0.4004
Epoch 8/10
2000/2000 [==============================] - 235s 117ms/step - loss: 0.5392 - acc: 0.8885 - val_loss: 3.0090 - val_acc: 0.3984
Epoch 9/10
2000/2000 [==============================] - 234s 117ms/step - loss: 0.5054 - acc: 0.9087 - val_loss: 3.3638 - val_acc: 0.3809
Epoch 10/10
2000/2000 [==============================] - 234s 117ms/step - loss: 0.4844 - acc: 0.9193 - val_loss: 3.7682 - val_acc: 0.3535
```

Figure 5: The training steps of dataset with RasNet20 model.

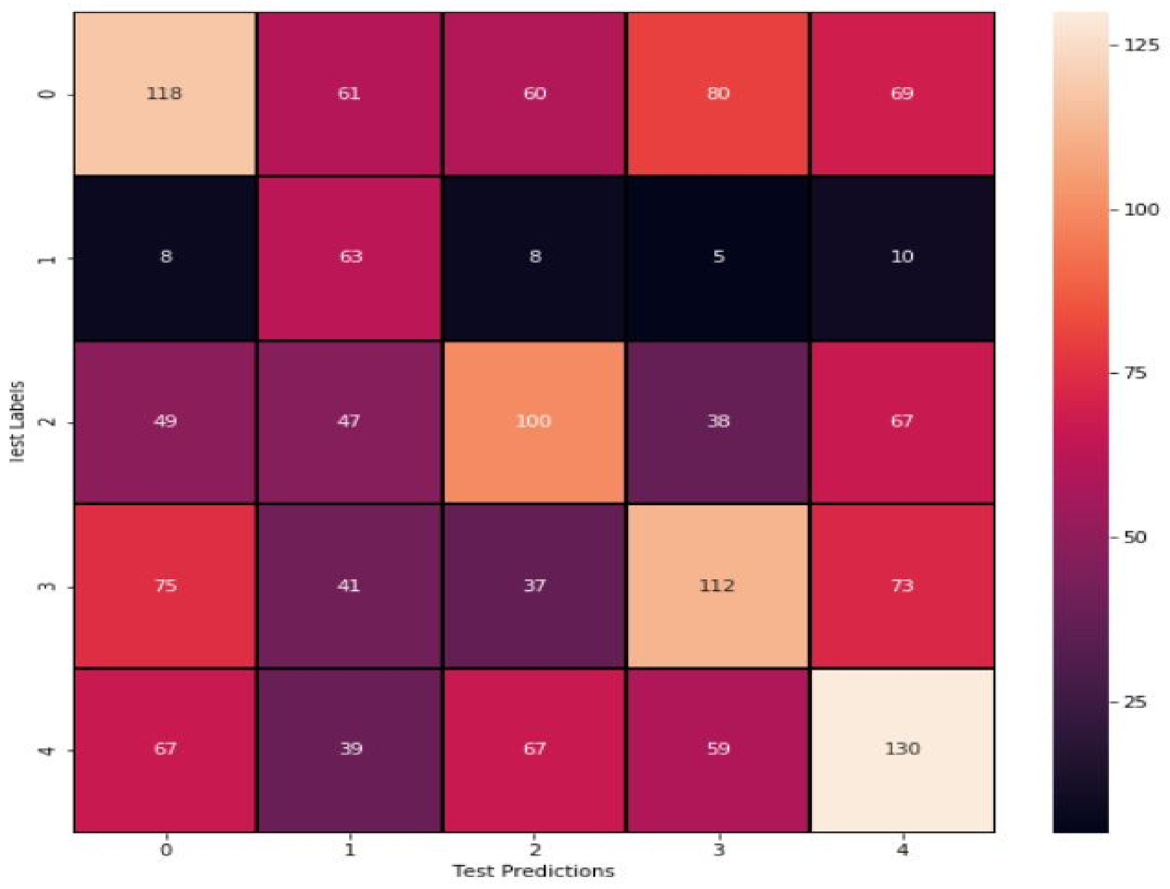
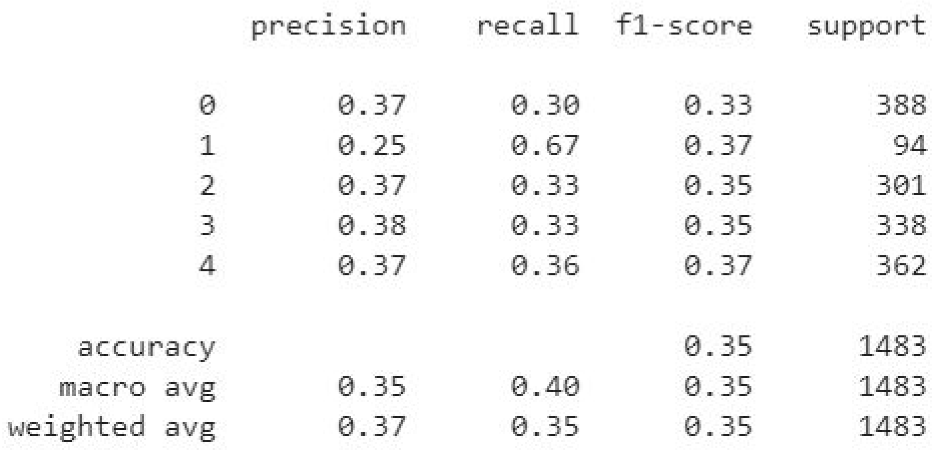


Figure 6: The confusion matrix of the labels and predictions with RasNet20 model.



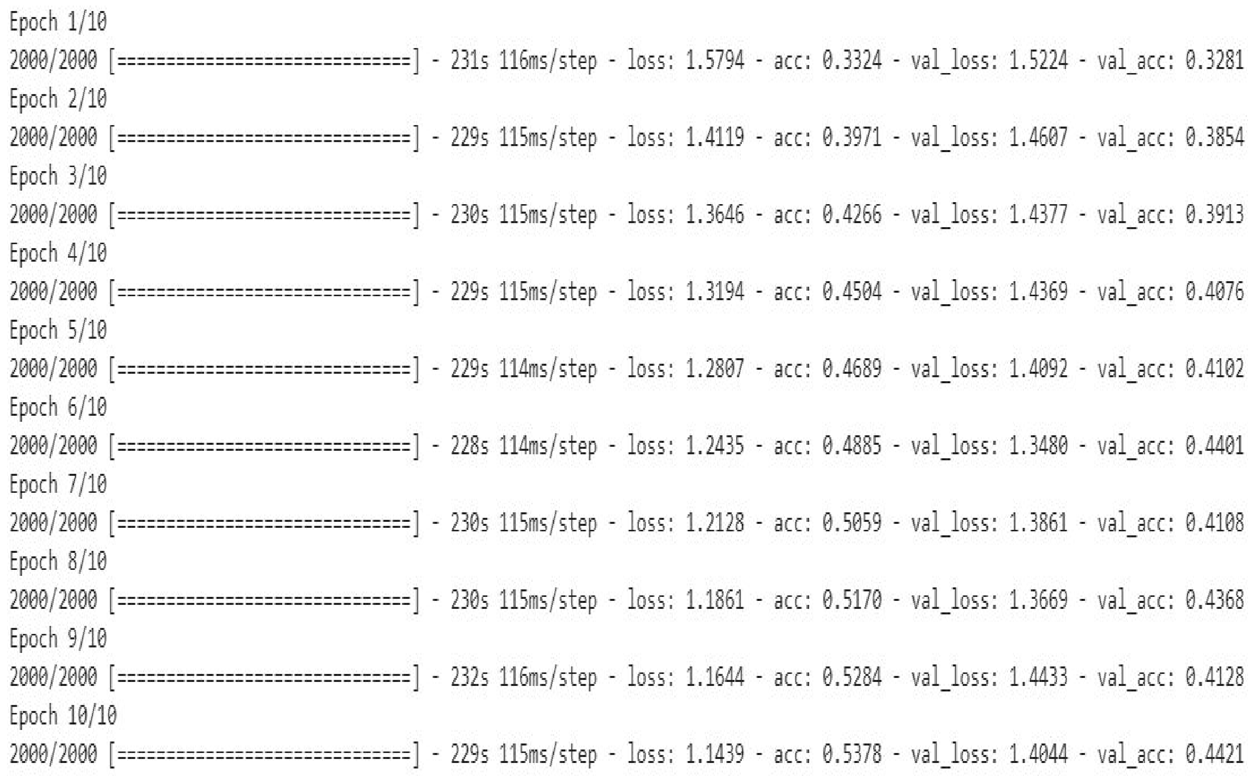
```
              precision    recall  f1-score   support

           0       0.37      0.30      0.33       388
           1       0.25      0.67      0.37        94
           2       0.37      0.33      0.35       301
           3       0.38      0.33      0.35       338
           4       0.37      0.36      0.37       362

    accuracy                           0.35      1483
   macro avg       0.35      0.40      0.35      1483
weighted avg       0.37      0.35      0.35      1483
```

Figure 7: Different metrics in addition to accuracy with RasNet20 model.

4.1.3 Custom Model Results



```
Epoch 1/10
2000/2000 [==============================] - 231s 116ms/step - loss: 1.5794 - acc: 0.3324 - val_loss: 1.5224 - val_acc: 0.3281
Epoch 2/10
2000/2000 [==============================] - 229s 115ms/step - loss: 1.4119 - acc: 0.3971 - val_loss: 1.4607 - val_acc: 0.3854
Epoch 3/10
2000/2000 [==============================] - 230s 115ms/step - loss: 1.3646 - acc: 0.4266 - val_loss: 1.4377 - val_acc: 0.3913
Epoch 4/10
2000/2000 [==============================] - 229s 115ms/step - loss: 1.3194 - acc: 0.4504 - val_loss: 1.4369 - val_acc: 0.4076
Epoch 5/10
2000/2000 [==============================] - 229s 114ms/step - loss: 1.2807 - acc: 0.4689 - val_loss: 1.4092 - val_acc: 0.4102
Epoch 6/10
2000/2000 [==============================] - 228s 114ms/step - loss: 1.2435 - acc: 0.4885 - val_loss: 1.3480 - val_acc: 0.4401
Epoch 7/10
2000/2000 [==============================] - 230s 115ms/step - loss: 1.2128 - acc: 0.5059 - val_loss: 1.3861 - val_acc: 0.4108
Epoch 8/10
2000/2000 [==============================] - 230s 115ms/step - loss: 1.1861 - acc: 0.5170 - val_loss: 1.3669 - val_acc: 0.4368
Epoch 9/10
2000/2000 [==============================] - 232s 116ms/step - loss: 1.1644 - acc: 0.5284 - val_loss: 1.4433 - val_acc: 0.4128
Epoch 10/10
2000/2000 [==============================] - 229s 115ms/step - loss: 1.1439 - acc: 0.5378 - val_loss: 1.4044 - val_acc: 0.4421
```

Figure 8: The training steps of dataset with Custom model.

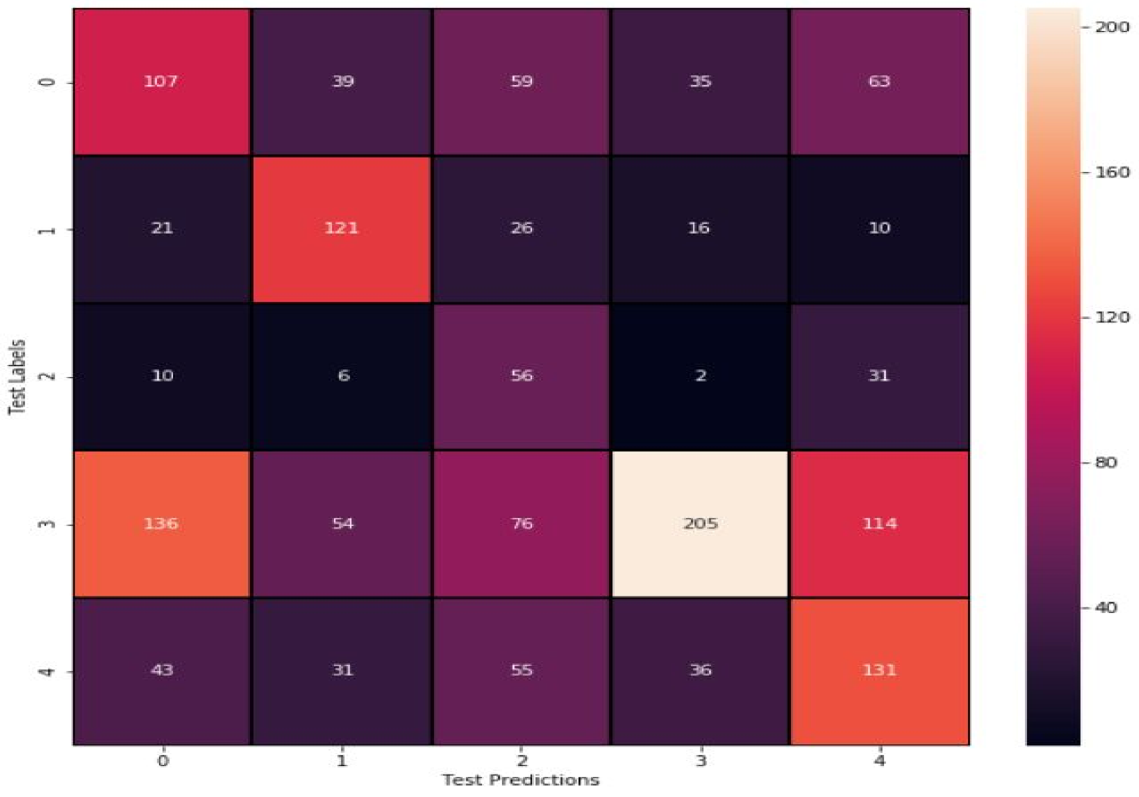
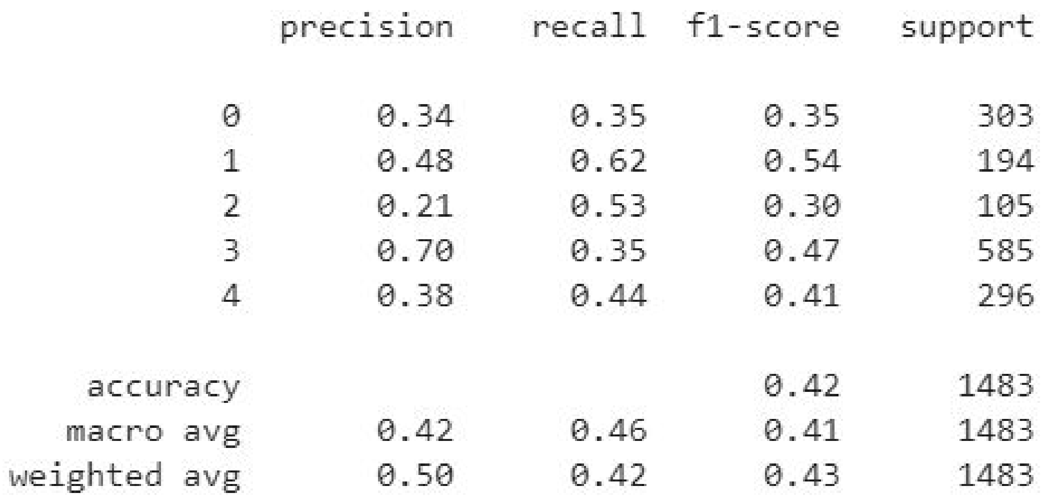


Figure 9: The confusion matrix of the labels and predictions with Custom model.



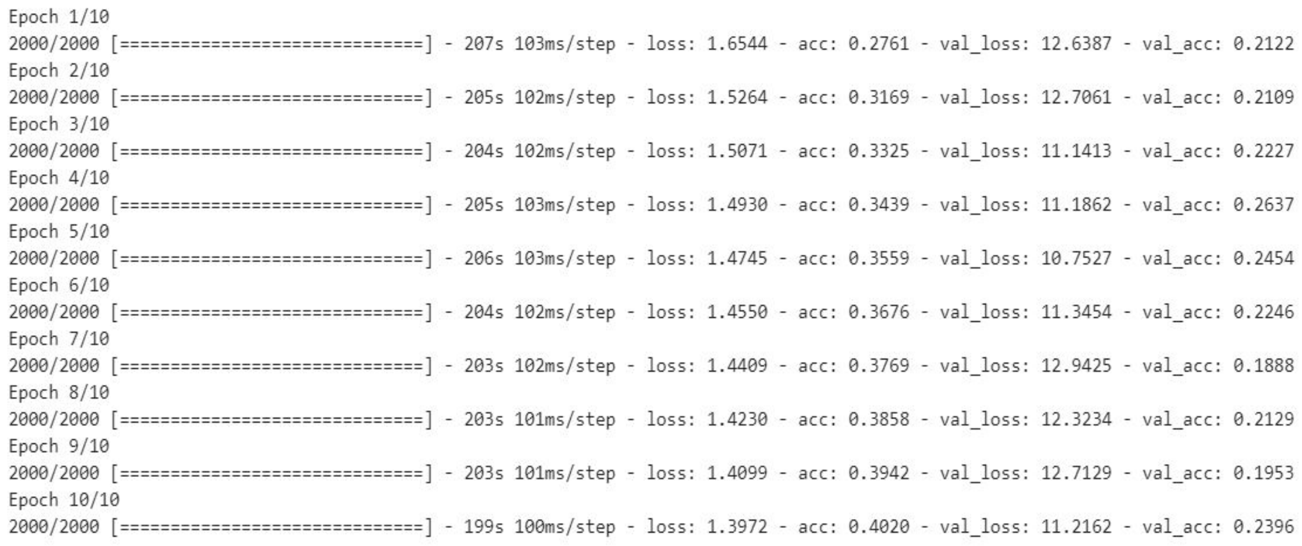
```
              precision    recall  f1-score   support

           0       0.34      0.35      0.35       303
           1       0.48      0.62      0.54       194
           2       0.21      0.53      0.30       105
           3       0.70      0.35      0.47       585
           4       0.38      0.44      0.41       296

    accuracy                           0.42      1483
   macro avg       0.42      0.46      0.41      1483
weighted avg       0.50      0.42      0.43      1483
```

Figure 10: Different metrics in addition to accuracy with Custom model.

4.1.4 Custom Model with PCA(50 Principal Component) Results



```
Epoch 1/10
2000/2000 [==============================] - 207s 103ms/step - loss: 1.6544 - acc: 0.2761 - val_loss: 12.6387 - val_acc: 0.2122
Epoch 2/10
2000/2000 [==============================] - 205s 102ms/step - loss: 1.5264 - acc: 0.3169 - val_loss: 12.7061 - val_acc: 0.2109
Epoch 3/10
2000/2000 [==============================] - 204s 102ms/step - loss: 1.5071 - acc: 0.3325 - val_loss: 11.1413 - val_acc: 0.2227
Epoch 4/10
2000/2000 [==============================] - 205s 103ms/step - loss: 1.4930 - acc: 0.3439 - val_loss: 11.1862 - val_acc: 0.2637
Epoch 5/10
2000/2000 [==============================] - 206s 103ms/step - loss: 1.4745 - acc: 0.3559 - val_loss: 10.7527 - val_acc: 0.2454
Epoch 6/10
2000/2000 [==============================] - 204s 102ms/step - loss: 1.4550 - acc: 0.3676 - val_loss: 11.3454 - val_acc: 0.2246
Epoch 7/10
2000/2000 [==============================] - 203s 102ms/step - loss: 1.4409 - acc: 0.3769 - val_loss: 12.9425 - val_acc: 0.1888
Epoch 8/10
2000/2000 [==============================] - 203s 101ms/step - loss: 1.4230 - acc: 0.3858 - val_loss: 12.3234 - val_acc: 0.2129
Epoch 9/10
2000/2000 [==============================] - 203s 101ms/step - loss: 1.4099 - acc: 0.3942 - val_loss: 12.7129 - val_acc: 0.1953
Epoch 10/10
2000/2000 [==============================] - 199s 100ms/step - loss: 1.3972 - acc: 0.4020 - val_loss: 11.2162 - val_acc: 0.2396
```

Figure 11: The training steps of dataset with custom model with PCA.

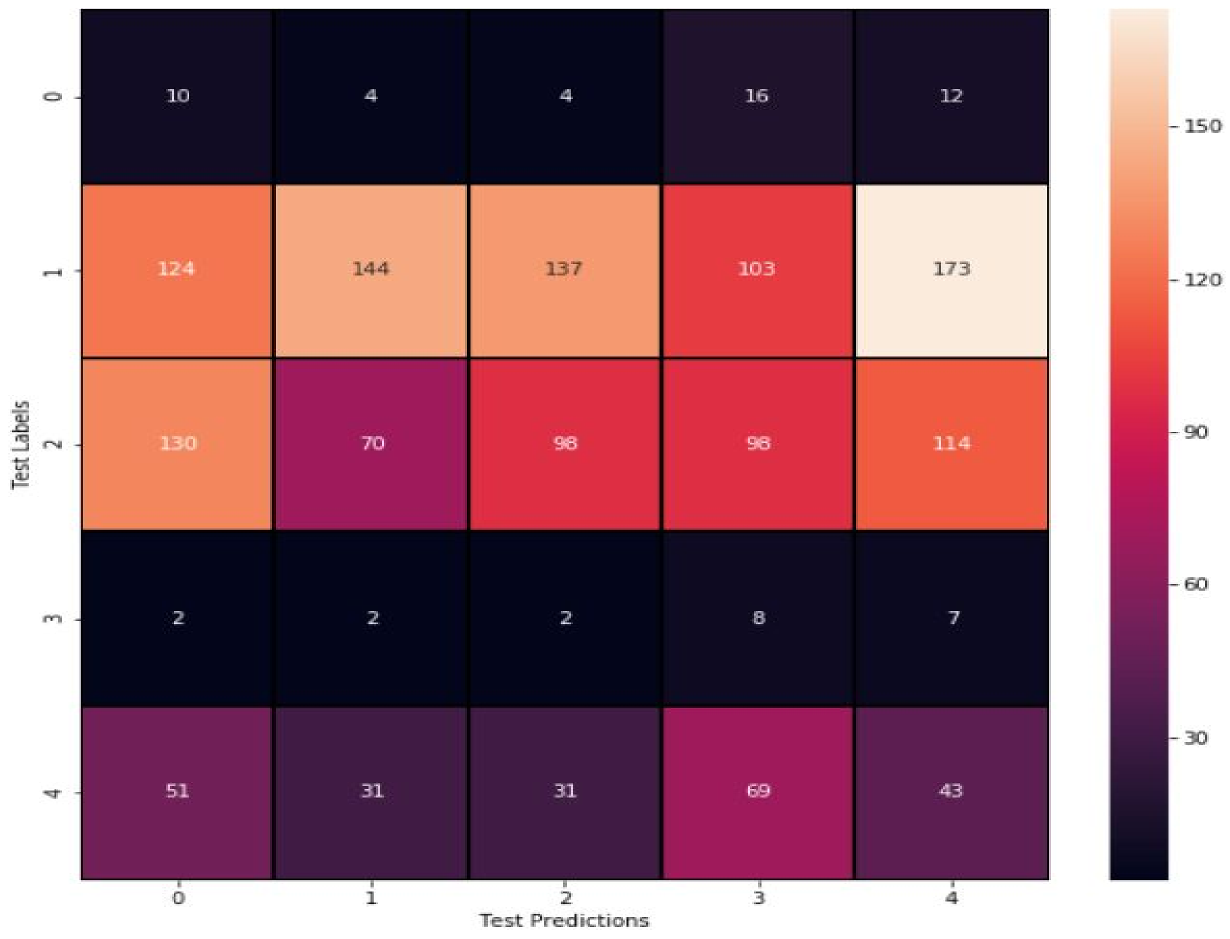
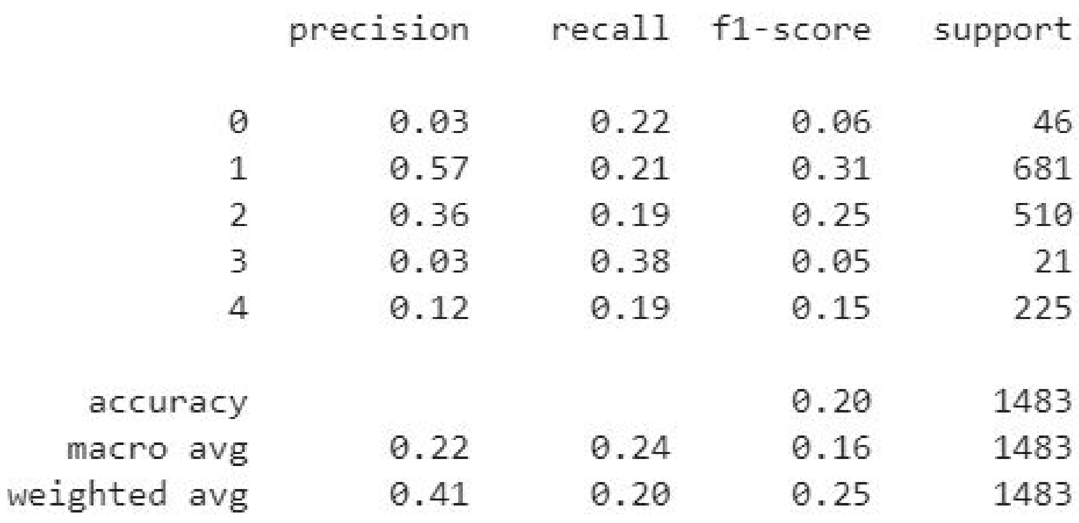


Figure 12: The confusion matrix of the labels and predictions with custom model with PCA.



```
              precision    recall  f1-score   support

           0       0.03      0.22      0.06        46
           1       0.57      0.21      0.31       681
           2       0.36      0.19      0.25       510
           3       0.03      0.38      0.05        21
           4       0.12      0.19      0.15       225

    accuracy                           0.20      1483
   macro avg       0.22      0.24      0.16      1483
weighted avg       0.41      0.20      0.25      1483
```

Figure 13: Different metrics in addition to accuracy with custom model with PCA.

4.2 Soundtrack Generation

As it is mentioned in the method section, categorical cross-entropy is used as loss function for our model. Loss values we got for the action model after trainings are as follows:

- Training with the first version of the action midi dataset on model A mentioned in the method section, we got 11.10 loss. The output was meaningless as can be seen in Appendix 8.2.1.

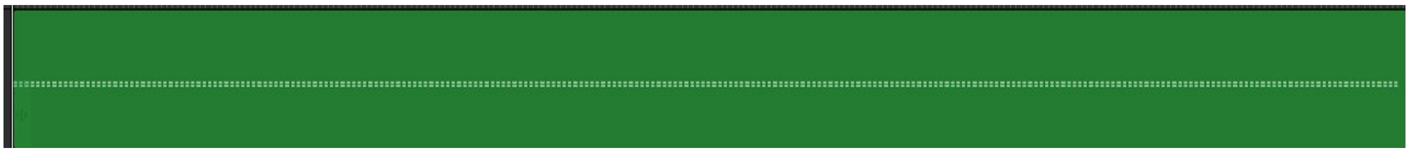


Figure 14: Meaningless output notes.

- Training with the revised version of the action midi dataset on model A, we got 4.7 loss, however, output was the same as Appendix 8.2.1. At this point, we were worried if our model was not performing well.
- Training with the same action midi dataset on model B, we got 0.0143 loss. When we predicted an output, the result was surprising. The model predicted a meaningful sequence. However, the model also overfitted to the Pirates of the Caribbean (Appendix 8.2.2). This was because the soundtrack of this movie was longer than the other sequences and this movie soundtrack dominated others. When we improved the dataset, we got a more balanced outcome (Appendix 8.2.3).

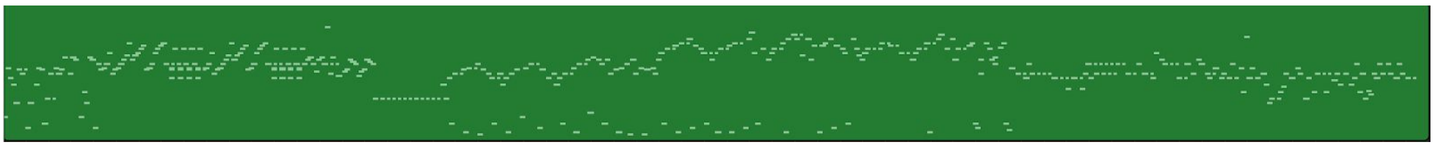


Figure 15: Reforged Pirates of the Caribbean notes.

After observing this, we used the same model and dataset configurations for other genres and got loss values for them as follows:

- Training with the comedy midi dataset on model B, we got 10^-4 loss (Appendix 8.2.4).
- Training with the horror midi dataset on model B, we got 0.0482 loss (Appendix 8.2.5).
- Training with the animation midi dataset on model B, we got 0.0114 loss (Appendix 8.2.6).

- Training with the drama midi dataset on model B, we got 0.019 loss (Appendix 8.2.7).

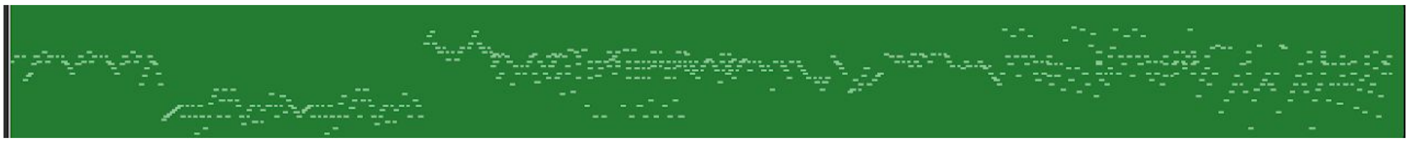


Figure 16: Comedy output notes.

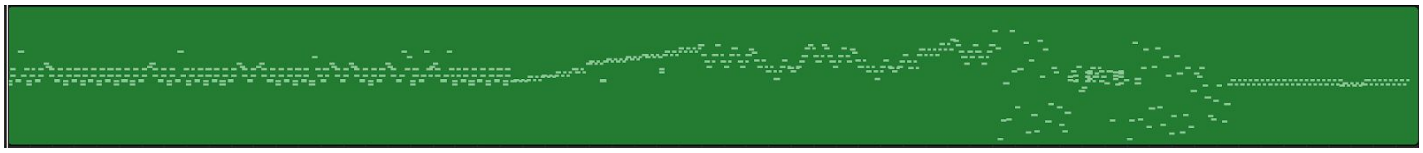


Figure 17: Horror output notes.

After training the network, we have created a new neural network to load the weights that we have saved in training process. Then, we choose a random index to start with and retrieve the predictions, as integers, according to that start index. Then, we convert integers to notes and generate an output MIDI file. In this way an output midi file containing notes of the generated soundtrack is obtained. For the trained models, generated soundtracks for genres were meaningful. At this stage, we wanted to do some experiments with number of notes. We increased the notes on action dataset to 120,000. We have trained a model B on this dataset and got loss 0.04 (Appendix 8.2.8).

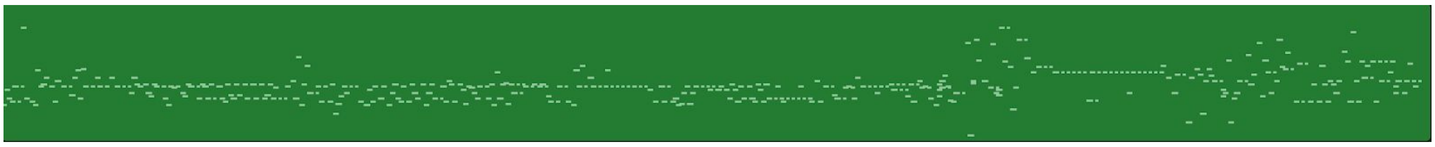


Figure 18: Action with 120,000 midi dataset output notes.

When we predicted an outcome, it seemed that this model B performed better than the previous action model B that was trained with 12,000 notes. As the number of notes increased, possibilities of new sequences from a note also increased. This way, we were able to break overfitting. As a second experiment, we changed the LSTM layer configuration of model A to BLSTM and trained the comedy dataset on this new model C. A loss of 10^-4 was acquired similar to the comedy model B. The results were both surprising and amazing (Appendix 8.2.9). While comedy dataset had 6600 notes, which could have caused overfitting due to this low number of notes, the output was original. It showed no sign of overfitting and was smooth while the model B output had repetitions and resemblance of input soundtracks.

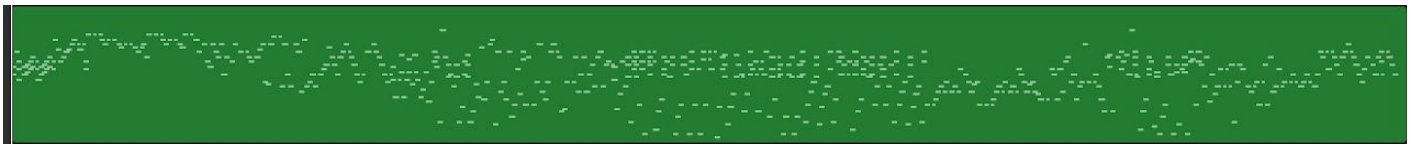


Figure 19: Comedy new output notes.

Ultimately, we have constructed 7 models, first one (model A) failed at the beginning, and the second model B successfully learnt from note sequences. However, model C proved that a more complex layer configuration can help deter the overfitting while predicting unique soundtracks even in lower number of notes.

5. Discussion

5.1 Poster Recognition

5.1.1 Model Configuration

We had 4 different models: A LeNet model, a RasNet20 model, and our customly structured model that used PCA for noise reduction. Our customly structured model used LeNet-5 as its CNN architecture, Adam as its optimizer, Categorical Cross-entropy as its cross-entropy and ReLU as its activation function. This helped overcoming LeNet's and RasNet20's problems to a degree. LeNet model was too simple for this problem and it overfitted and RasNet required very large dataset which we did not have. Our custom model did not face these problems.

5.1.2 Performance Metrics

As you can see from the results, our custom model had the best accuracy with 42% accuracy. The reason LeNet and RasNet20 models did not perform well is that as I mentioned in the previous subsection, LeNet is too simple for this problem that results in overfitting the data. RasNet20 requires a lot of data (millions) which we did not have and could not construct from scratch. Due to RasNet20 is a complicated and powerful model, it reached 91% accuracy in training dataset but unfortunately overfitted to validation dataset. This resulted in badly trained model for RasNet20 and it had low accuracies. Our model still does not go above 50% accuracy, but this also ties with our dataset having two main problems that affects our models immensely. One of them is that our data is not big enough to catch varying patterns of the film categories and other one is that from a small pool of our dataset, we realized that it had wrongly labeled films. Other than that, it performs relatively well for posters that had distinct and universal elements for a category. Related to PCA, it is thought that if PCA was used to the dataset and the dataset was trained with our custom model, accuracy would increase. However, it did not work. we assumed that after applying PCA, dataset lost many important features, it did not overfit the data and accuracy of training set did not increase much.

5.1.3 Dataset Problems

As I mentioned above, our dataset had issues that affected all of our models. We used Kaggle's movie genre from its poster:
https://www.kaggle.com/neha1703/movie-genre-from-its-poster
This dataset included 10,000 comedy and drama posters while having 1000 to 2000 horror and animation posters. It had a huge gap that no doubtedly would result in overfitting so we cut the data to approximately 1500 posters for each category in a total of approximately 7500 total dataset. This is low for training such a complex model, but this was necessary to not overfit our model. Our model did not overfit, and it still had 42% accuracy but if we had millions of data that was distributed equally among the categories, both RasNet20 and our model would perform better.

The other issue of the dataset was it included posters that was wrongly labeled. We realized this while observing our results. The prediction would say horror to a movie, but the actual label was reported as action. When we checked the film's category we would see that it was indeed horror. We realized this issue for 5-7 films for a pool of maybe 50-55 films. If 10% of this pool was wrongly labeled, we did not know how many among all data were wrongly labeled and how many of these affected our prediction results. Our theory is that our model indeed has better accuracy than it is reported but because we cannot manually search for every one of the wrongly labeled films we accept our accuracy as 42%.

5.1.4 Possible Improvements in Future

Improvements can be mostly made from the dataset. If the two main problems I described in the previous subsections would be solved for our model, we would get better results. Like any model, our model also heavily depends on the dataset and as long as small and wrongly labeled dataset is presented for CNN, predictions will be flawed.

Another improvement may be done by multi-labeling. Normally, films do not have only one category, there are usually several categories labeled for a film. We used only the main labels of a film from the dataset (the kaggle dataset did have all of the listed categories for a film on IMDB). If we used all the labels for a film for training, we could lower the problem of posters having varying elements of different film categories and this affecting our model's training. In this case, we also need to consider labeling the posters with multiple categories while predicting. This is a harder task than what we used in our poster recognition, but it would no doubt improve the performance of any model we used for our project.

5.2 Soundtrack Generation

5.2.1 Loss Value

First of all, as we observed from the final loss values of and generated soundtracks from the models, lower loss values are important for generating soundtracks that cannot be differed from music composed by a musician. As it is seen from the result appendix 8.2.1 and 8.2.2, difference between soundtracks generated with larger and lower loss values is huge. Also as we get lower loss values for a dataset, the quality of the output soundtrack increases. Therefore, we always got the best result from model with the lowest loss value.

5.2.2 Originality of the Output

As it is mentioned above, first meaningful output we got resembles the soundtrack of Pirates of Caribbean. It starts with an original melody, however, after some point it starts to be very similar to soundtrack of Pirates of Caribbean and continues almost the same with it. After we examined the dataset, we observed that number of notes in soundtrack of Pirates of Caribbean is very much larger compared to other midis and it repeats the same melody couple of times. Since we prefer lower loss values, the model learns the dataset as much as it can. If there is data in the dataset that has the most number of the notes in the dataset individually, the model focuses on learning it more than other midis to minimize the loss value. In other words, it overfits to this data and it results in similarity in terms of melody in our case. Since lower loss values are still preferable for the task, we should prevent this overfitting by revising our dataset carefully to remove or make smaller some data that cause this issue as it is mentioned in the previous sentences. After this review, our result is more original as it can be seen from the result appendix 8.2.3.

5.2.3 Model Configuration

As it is mentioned above, in terms of model configuration the most considerable difference observed when unidirectional LSTM layer was replaced by bidirectional LSTM layer. The difference between unidirectional and bidirectional LSTM layers is that the data sequence is fed into unidirectional layer only starting from the beginning of the sequence through the end, whereas the data sequence is fed into bidirectional layer both starting from the beginning and the end through the other side of the sequence. First of all, we got lower loss values with larger number of epochs by using bidirectional layers. It helps model to learn dataset better and it reflects to the loss value. Also it allows model to capture the harmony of a music composed by a musician and maintain the same feeling through the generated soundtrack.

5.2.4 Possible Improvements in Future

There are some deficiencies of the soundtrack generation part of the project. First of all, we create single instrument soundtracks, although most of the movie soundtrack is produced with large number of instruments. Since we try to make listeners feel the emotions related to this genre by only one instrument, generated soundtracks with our models are weaker than real movie soundtracks in terms of feelings they contain. Solving this problem may result in better products in terms of this issue in the future.

Besides, our models generate soundtracks by looking only one genre. It also causes deficiency in emotions since movies may include several genres with several weights. Therefore, to improve it, instead of one genre input to the model, several genre inputs with their weights can be used and it makes generated soundtracks more suitable for particular movie.

6. Conclusions

6.1 Poster Recognition

As an outcome of our experiments, results, observations and discussions, we have concluded that from our 3 different models(LeNet, Rasnet20, Custom Model), we could get 44% accuracy at best because recognizing posters is a very complex task. We need to consider many varying factors in a poster and there are many posters that may not obey to its category’s general poster patterns which can increase the noise. One more problem we faced while training the data is that the data we have used has wrongly labeled films, which decreased the prediction accuracy for our model but in the end, even with a small data set, too much noise and wrongly labeled posters, we concluded that it is possible to create a model that can predict a movie’s poster. It is also possible to create a better model if we had better and bigger dataset but our model still worked fine. We used customly created

CNN architecture with Adam optimizer and Categorical Cross-entropy and we got the best results from this model.

To sum up, our model can predict a movie poster that has very unique and universal poster patterns that obeys to its film category, but faces some difficulties with posters that do not obey universal patterns and has complex elements in it. It is improvable, but that mostly depends on the dataset that is provided to us. We have managed to predict a category that can be passed to the soundtrack generation for generating a soundtrack for its film category.

6.2 Soundtrack Generation

As an outcome of our experiments, results, observation and discussion, we concluded that it is possible to create a model that can capture the harmony of the music, decide between notes and chords and reflect these crucial components of a music to its generated soundtracks belong to a particular genre. Final trained models can generate original soundtracks different from songs in the dataset and each other. Also outputs are not distinguishable from a music produced by a human which also addresses some of our questions at the beginning of the project. Using LSTM layers and RNN method, especially bidirectional LSTM layers, allows model to capture general structure of a song without breaking the harmony. Besides, proper preparation of the dataset as it is mentioned in previous sections allows model to produce more original outputs which is one of the issues mentioned in the problem description section. To sum up, our project has addressed the main problem that are described at the beginning of the project which is how to generate soundtracks that are original, belong to a particular genre and not distinguishable from the music produced by a human.

7. References

[1] “Movie Genre from its Poster,” *Kaggle*, 2018. [Online]. Available: https://www.kaggle.com/neha1703/movie-genre-from-its-poster. [Accessed: 30-Oct-2019].

[2] “Movie Themes Main Page” *moviethemes,* [Online]. Available: https://moviethemes.net. [Accessed: 30-Oct-2019].

[3] “Recurrent Neural Networks (RNN) with Keras,” *tensorflow,* [Online]. Available: https://www.tensorflow.org/guide/keras/rnn. [Accessed: 23-Nov-2019].

[4] “What is music21?,” [Online]. Available: http://web.mit.edu/music21/. [Accessed: 23-Nov-2019].

[5] “Tf.Keras.Layers.Bidirectional,” *tensorflow,* [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional. [Accessed: 23-Nov-2019].

[6] “Tf.Keras.Layers.Dropout,” *tensorflow,* [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout. [Accessed: 23-Nov-2019].

[7] “Tf.Keras.Layers.Softmax,” *tensorflow,* [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Softmax. [Accessed: 23-Nov-2019].

[8] F D, “Batch Normalization in Neural Networks”, towardsdatascience, [Online]. Available: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c. [Accessed: 23-Nov-2019].

[9] “Tf.Keras.Optimizers.RMSprop,” *tensorflow,* [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop. [Accessed: 23-Nov-2019].

[10] “Tf.Keras.Backend.Categorical_Crossentropy,” *tensorflow,* [Online]. Available:https://www.tensorflow.org/api_docs/python/tf/keras/backend/categorical_crossentropy. [Accessed: 23-Nov-2019].

[11] S. Skúli, “How to Generate Music using a LSTM Neural Network in Keras,” *towardsdatascience,* [Online]. Available: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5. [Accessed: 23-Nov-2019].

[12] T. Doll, “Making Music with Machine Learning,” *towardsdatascience,* [Online]. Available: https://towardsdatascience.com/making-music-with-machine-learning-908ff1b57636. [Accessed: 23-Nov-2019].

8. Appendix

8.1 Member Contributions

8.1.1 Gonca Yılmaz

I have mainly played a role in music generation part. I have prepared datasets by reducing MIDI files to single instrument with Cevat and Hakan. Then, I have worked on two projects that we had found for music generation to use it as our guideline and found research which concern music generation with RNN in a deeper level, which might help us in the future of our project. I was in a collaboration with Cevat and Hakan and doing some experiments to get better results in music generation process. For RNN part, I have played roles in music generation for drama and action genres. After, obtaining results with our model, I have reviewed datasets to obtain better results.

8.1.2 Mustafa Çağrı Güngör

I helped creating midi dataset for RNN and was responsible for CNN part of project. Read lots of articles and analyzed many source codes of CNN projects. Regulated dataset, divided it to close number of images for each class and convert them from jpg file to npy file to reduce load time and store dataset. Data augmentation and preprocessing part are applied. Different CNN models, optimizers, loss functions are tried and found best accuracy. I evaluated reliability of dataset and discussed future plans for reaching better accuracy with Talha.

8.1.3 Talha Sen

For the music generation part I helped creating midi dataset for RNN and I was a part of the genre prediction from posters, which used CNN. My main part in the team was researching how to solve the problem of predicting genre from the posters, how to apply CNN for this problem, how could we increase the accuracy of our predictions and find articles related to these problems. Çağrı did the main coding for CNN and I analyzed and discussed with him how we could improve our results.

8.1.4 Hakan Sivuk

I have tried to manage the project by setting group meetings, following the deadlines and making the members aware of the dates. I have found some midi files for each genre and processed the midi files to make them single instrument midi files. I have trained one of the projects we found as a background work and changed some parts for our project and observed performance increase while using bidirectional LSTM layer instead of unidirectional LSTM layer. I prepared the action midi dataset with Cevat and Gonca and change it when it was needed after some observations mentioned in above sections. After we trained a model for action successfully I prepared comedy and action dataset by cutting needed parts and making them single instrument. Then, I trained a model with the same configurations used for action by using animation dataset. After trained the model I observed loss value and evaluated the result soundtrack.

8.1.5 Cevat Aykan Sevinç

I have developed a python3 script to compose the movie poster dataset. Furthermore, I have found two previously implemented projects for music generation using RNN. Moreover, I have experimented with RNN to produce soundtracks while communicating with Hakan and Gonca for training our model. I created the midi dataset for action and horror. Furthermore, I have trained the model for action, comedy and horror genre. Next, I have changed the configurations of our method to get a different output in comedy genre.

8.2 Soundtrack Outputs

All:
https://drive.google.com/drive/folders/1jZKT7FsUW2mTdFrXwNpikg-zBddgrzQd?usp=sharing.

- **8.2.1 Meaningless:**

  https://drive.google.com/file/d/152OT3Ko_cNldI9jcW0Ecq_uOPFYT6FxB/view?usp=sharing.

- **8.2.2 Fake Pirates of the Caribbean:**
  https://drive.google.com/file/d/1XjTmf1daCb1JLsw3pZv4hYoDUaKjyWis/view?usp=sharing.

- **8.2.3 Action with Better Dataset**:

  https://drive.google.com/file/d/1WKkEN7menrKWnTbp-AjyC-bHcqXtQokS/view?usp=sharing.

- **8.2.4 Comedy Example**:

  https://drive.google.com/file/d/1l48Y4LzXFneHvktSBcRKovbOdeiQYfsV/view?usp=sharing.

- **8.2.5 Horror Example**:

  https://drive.google.com/file/d/1JFYrT3J1bTxGkt7njCp6sZEsCpMkhP_k/view?usp=sharing.

- **8.2.6 Animation Example**:

  https://drive.google.com/file/d/1fgfaG6IqehIpDFnDx4q_nLxg7Jbbe1H7/view?usp=sharing.

- **8.2.7 Drama Example**:
  https://drive.google.com/file/d/1yClcF9BebBpHJyTY5iUcY_-QHIBw75cQ/view?usp=sharing

- **8.2.8 Action with 120,000 Notes**:

  https://drive.google.com/file/d/1sSGdeC_SYvSwx9C4It845sEnbNvboYbb/view?usp=sharing.

- **8.2.9 Comedy Model C**:

  https://drive.google.com/file/d/1p60R99gRXLWf9For36lObNqv7sakdon8/view?usp=sharing.