ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

BUSINESS ANALYTICS Master of Science

Machine Learning and Content Analysis

Vogiatzis Giorgos P2821827

Prasinos Michalis P2821823

Zourou Myrsini Eirini P2821828

Judge A Movie by Its Cover

1. Description

The scope of this project is to predict the movie genre by its cover image, short description or both. This is interesting because, if the models we develop are accurate, we can classify movies to more than one genre. Business-wise this is useful for internet platforms (Netflix, Amazon or similar), where they consume large volumes of movies to share with their users. These movies are classified by these platforms by their respective categories. Also for the movie industry, it is crucial that the poster of the movie is in accordance with its genre. So before these models go publicly available, we have to make sure that the poster-short description send the right message genre-wise to its audience. These are the problems that we attempt to solve and for this project we need to acquire a large dataset of movie posters and short descriptions through web-scrapping from Imdb, clean and transform our dataset, extract useful features, train our models, test them with a validation dataset and evaluate them. A successful model could be used in production by similar platforms or graphic design and content teams in film studios.

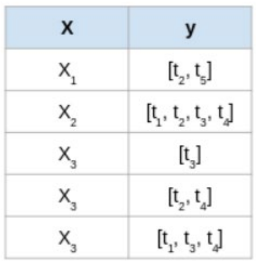
2. Mission

Our mission is to find if there are certain elements of a poster that allow a model to predict the movie's genre. The input to our algorithm is a color image of a movie poster and our model outputs is a list of genres that classify the movie. We apply various deep learning models and strategies to extract the features of the poster to make our predictions.

Therefore, our problem is also a multi-label classification problem. In machine learning, multi-label classification and the strongly related problem of multi-output classification are variants of the classification problem where multiple labels may be assigned to each instance.

Multi-label classification is a generalization of multiclass classification, which is the single-label problem of categorizing instances into precisely one of more than two classes. In the multi-label problem, there is no constraint on how many of the classes the instance can be assigned to.

Formally, multi-label classification is the problem of finding a model that maps inputs x to binary vectors y (assigning a value of 0 or 1 for each element (label) in y). Below we represent how should be the input (x) and the output (y) of a multi label problem. We observe that we have multiple tags not only across the table but for individual inputs as well.



X	y
X_1	$[t_2, t_5]$
X_2	$[t_1, t_2, t_3, t_4]$
X_3	$[t_3]$
X_3	$[t_2, t_4]$
X_3	$[t_1, t_3, t_4]$

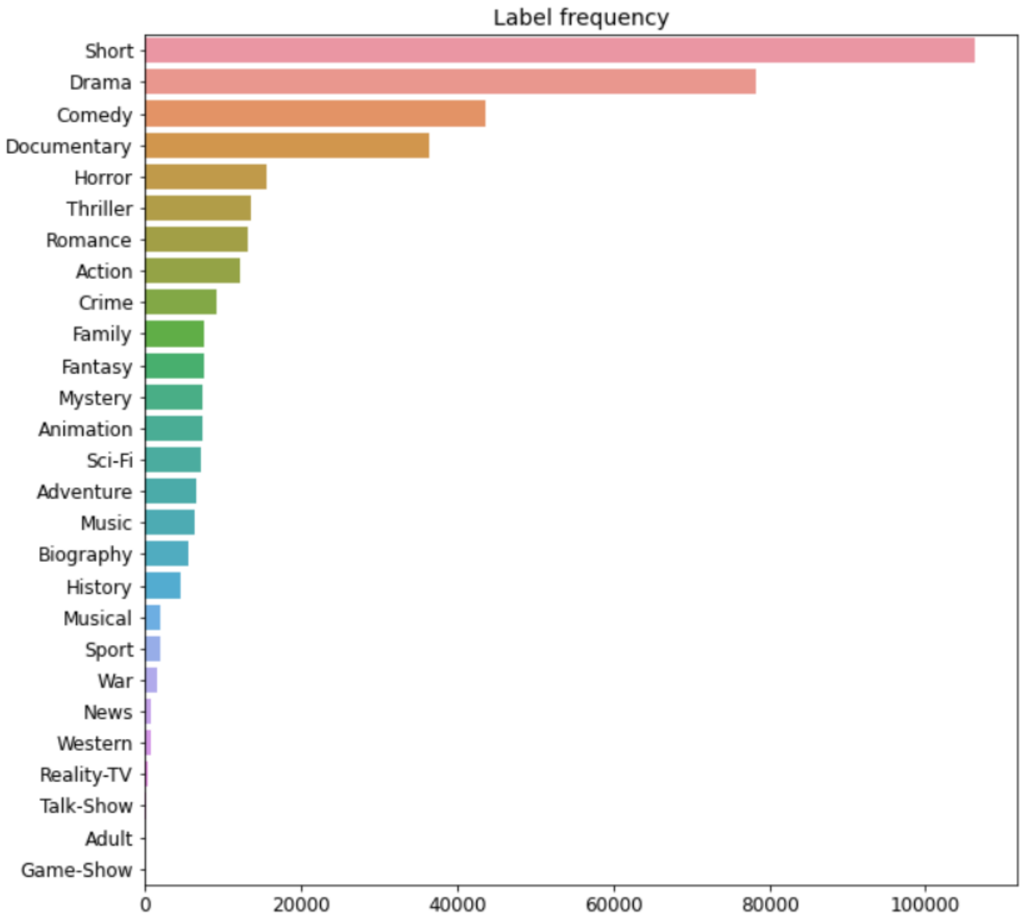
Multi-label Classification

In the bibliography, exist many studies that try to solve our project, to predict the movie genre from the cover image, for a binary class. Sequentially, we present some of the previous studies and theirs results. Dhruvil Shah published a study where he try to predict the movie genre with a CNN model. For his study, only the cover images from action, drama and comedy movies were used with a sample size of 3900. The final model has 50% of accuracy.

(https://towardsdatascience.com/cnn-approach-for-predicting-movie-genre-from-posters-95f122f88bc2). Moreover, there is a kaggle competition where they tackle the issue from a sample dataset and the best f1 score so far is 0.46299.

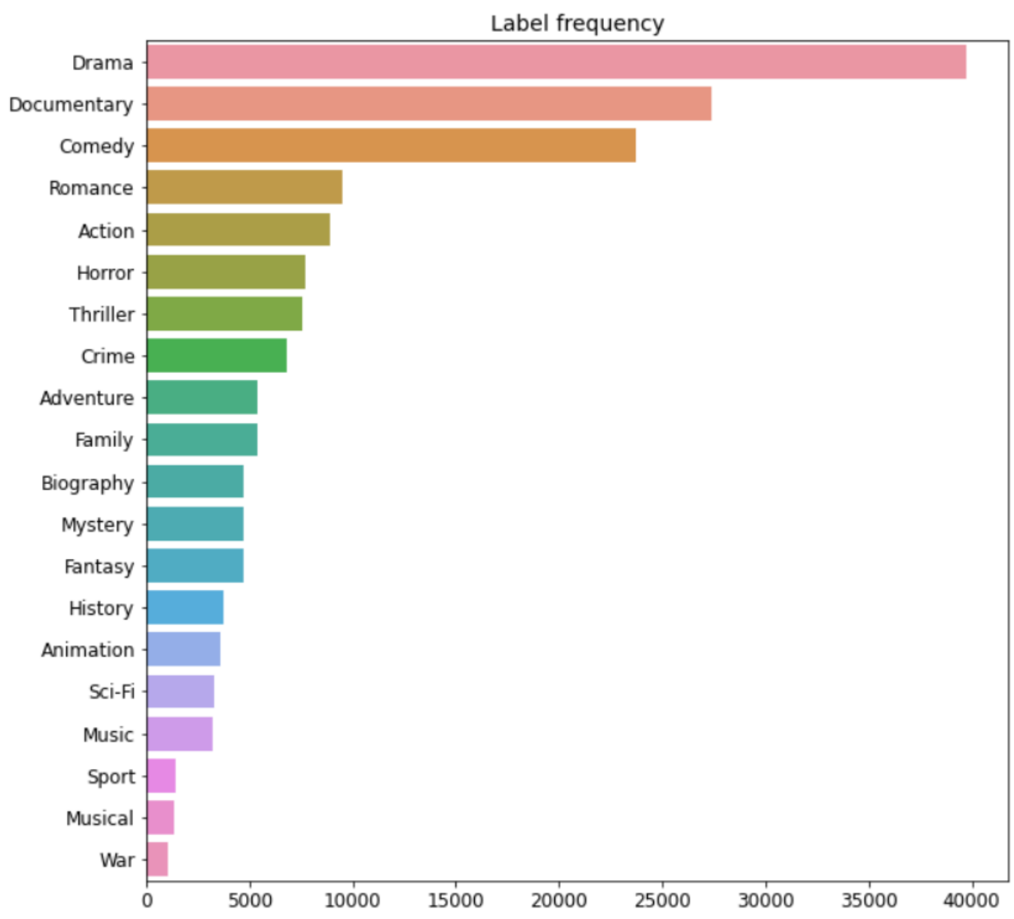
3. Data

Our data come from the IMDb (Internet Movie Database), which is a popular online movie platform. First, we downloaded the title.basics.tsv file from the imdb datasets. We excluded the categories tvSeries, tvSpecial, tvEpisode, videoGame, video and kept movies only from 2000-2010 from the original file and then we used web scrapping to get the image and the short description from IMDb. First, we created a list of urls from the movies above and then we scrapped these urls for the two required fields. After downloading the films and loading the created file the categories frequency appears as follows. We have downloaded 210000 photos and used 100.054 for our model.

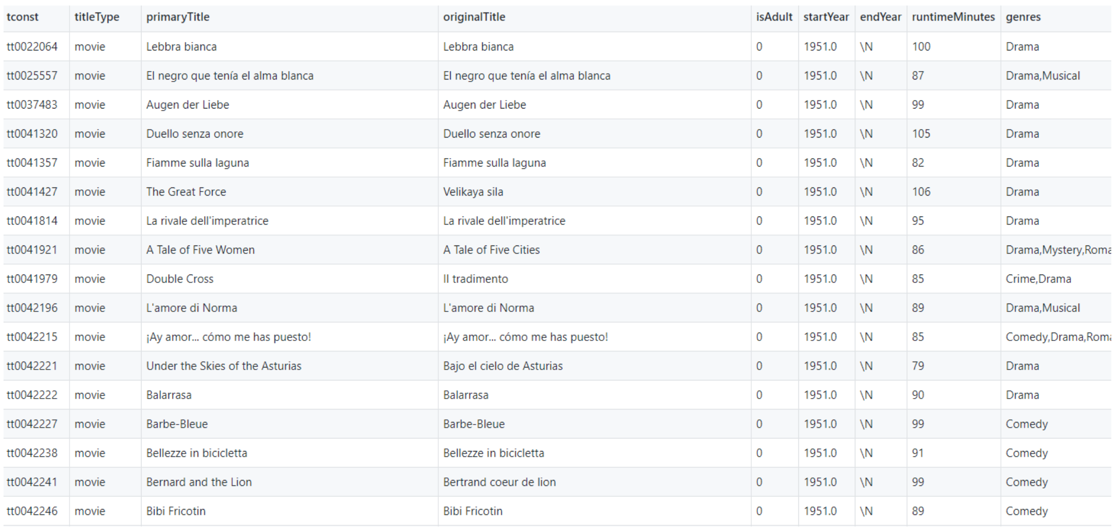


Because the dataset is imbalanced, we kept only the genres with more than 1000 observations and excluded the category “Short” since it holds no information about the movies genre, rather than its length. The dataset remained imbalanced, because many categories appear with lower frequency than others. This is the first problem we encountered.

The final dataset genre frequency table is as follows:



Sample Dataset:



tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
tt0022064	movie	Lebbra bianca	Lebbra bianca	0	1951.0	\N	100	Drama
tt0025557	movie	El negro que tenía el alma blanca	El negro que tenía el alma blanca	0	1951.0	\N	87	Drama,Musical
tt0037483	movie	Augen der Liebe	Augen der Liebe	0	1951.0	\N	99	Drama
tt0041320	movie	Duello senza onore	Duello senza onore	0	1951.0	\N	105	Drama
tt0041357	movie	Fiamme sulla laguna	Fiamme sulla laguna	0	1951.0	\N	82	Drama
tt0041427	movie	The Great Force	Velikaya sila	0	1951.0	\N	106	Drama
tt0041814	movie	La rivale dell'imperatrice	La rivale dell'imperatrice	0	1951.0	\N	95	Drama
tt0041921	movie	A Tale of Five Women	A Tale of Five Cities	0	1951.0	\N	86	Drama,Mystery,Roma
tt0041979	movie	Double Cross	Il tradimento	0	1951.0	\N	85	Crime,Drama
tt0042196	movie	L'amore di Norma	L'amore di Norma	0	1951.0	\N	89	Drama,Musical
tt0042215	movie	¡Ay amor... cómo me has puesto!	¡Ay amor... cómo me has puesto!	0	1951.0	\N	85	Comedy,Drama,Roma
tt0042221	movie	Under the Skies of the Asturias	Bajo el cielo de Asturias	0	1951.0	\N	79	Drama
tt0042222	movie	Balarrasa	Balarrasa	0	1951.0	\N	90	Drama
tt0042227	movie	Barbe-Bleue	Barbe-Bleue	0	1951.0	\N	99	Comedy
tt0042238	movie	Bellezze in bicicletta	Bellezze in bicicletta	0	1951.0	\N	91	Comedy
tt0042241	movie	Bernard and the Lion	Bertrand coeur de lion	0	1951.0	\N	99	Comedy
tt0042246	movie	Bibi Fricotin	Bibi Fricotin	0	1951.0	\N	89	Comedy

4. Methodology

The issue we are facing is a multi-label classification and we have created 2 different models in order to compare them with each other. First model is a CNN with takes the poster of a movie as input the second model will be a CNN-LSTM which takes as input a short description of the movie. Both models then predict the classify the movies to genres.

Image Classification CNN

First, we load the data from the csv and we transform the data as mentioned in the previous step. Because all images were of different size and our cnn model, requires standard size inputs, we set the size of all the pictures to 224x224, while maintaining all 3 RGB channels. The matrix that will derive from this procedure will have a size of 224x224x3.

Because the model cannot understand a categorical variable, we transformed all genres to a binary pseudo variable with OneHotEncoding. This means that for each movie we created a vector with 20 binary variables, same count as our total genres in the dataset, which are 0 when the genre of the movie is not included and 1 when the genre of the movie is included.

```
In [7]: all_genres = [[genre for genre in cel.split(',')] for cel in labelsCsv.genres]
        one_hot = MultiLabelBinarizer()
        y_s =one_hot.fit_transform(np.array(all_genres))
        y_s

Out[7]: array([[0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               ...,
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0]])
```

This means that the problem we are facing is a Multi-Label Classification problem, resulting in a movie belonging to one category or more.

In order to start building our model, first we need to split our dataset to train-80% and validation-20% while shuffling the data

```
# creating and formating the dataset that we are going to use
d = {'filename' : [tconst + ".jpg" for tconst in labelsCsv.tconst],
    'labels' : [[genre for genre in cel.split(',')] for cel in labelsCsv.genres]}

img_metadata_df = pd.DataFrame(d)

# splitting to train and test
train, test = train_test_split(img_metadata_df, test_size=0.20, random_state=9, shuffle=True)
```

Because the volume of the data was huge, it was not possible to load them into the memory all at once in order to fit them to our model, we used generators, where the we loaded the images into batches of 100 each time.

After the creation of the training and validation dataset, we rescaled-normalized the pixels of each image dividing with the max value (255). Gradient Clipping was implemented to make the model training faster and to avoid the exploding gradient, which is the exponential increase of the gradient as we propagate down through the model, resulting in poor loss or NaN loss during training. This was done both for the training and validation set (80%-20%).

We also created a test generator, for the test dataset we kept to evaluate our model.

Because we had an imbalanced dataset, we assigned weight to each class, proportional to the size of each class.

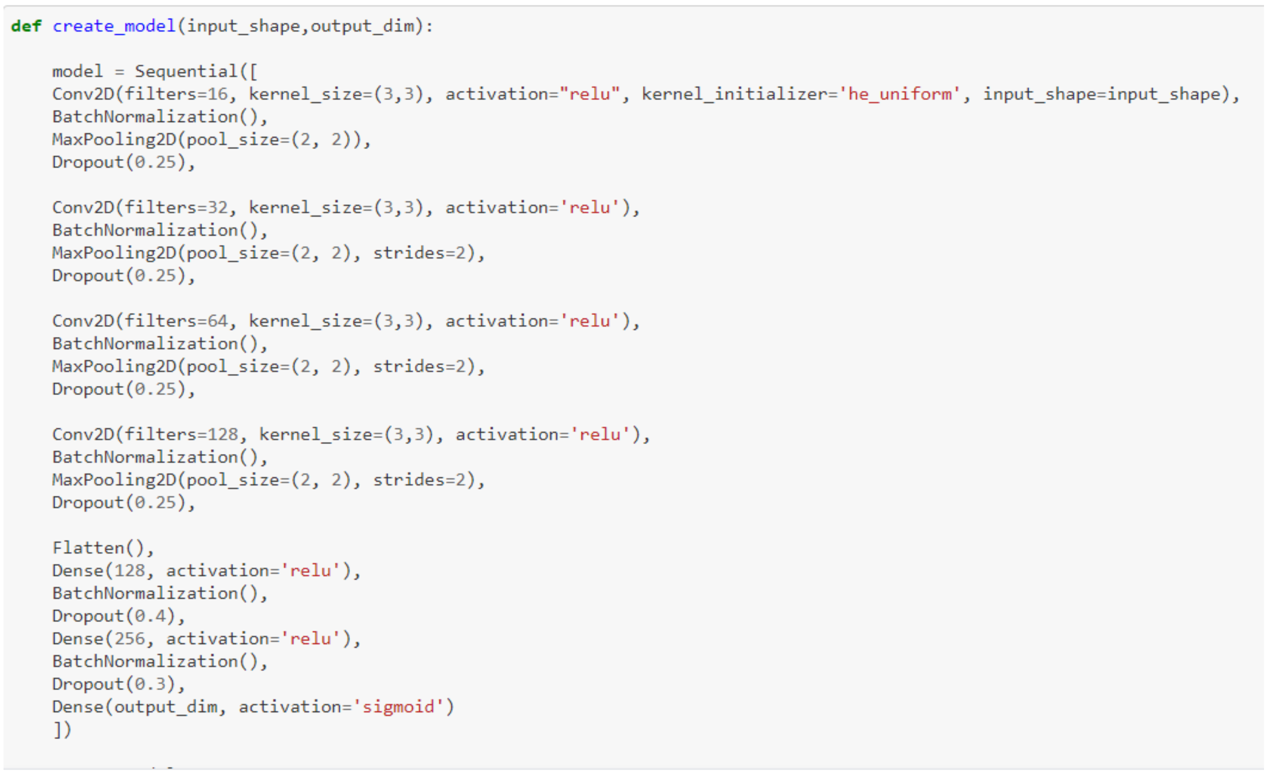
$$W_i = n_{largest\ genre}/n_i$$

This weight is the reciprocal number of the frequency and assigns a higher weight to less represented genres.

Our Model uses the sequential API from Keras and creates 4 convolutional layers, which are used as filters over the images. At first, we wanted to use less filters, in order to extract low level features such as lines. As our model progressed, we used more filters in order to extract higher level and more complex features such as corners and windows. To continue, we flattened the layers outputs, to pass them from a one-dimensional vector, in order to obtain the final probabilities of each image belonging to the corresponding genres.

In between of each layer there is a dropout layer in order to avoid overfitting our model. This basically switches off 25% of the neurons used. We have also included a max pooling layer, which has a 2,2 filter. This means that over every image, the max pooling layer extracts the highest value of 4 pixels. The last 3 max pooling layers have also stride=2, meaning that while moving over the pixels, it skips 1 pixel at each move.

Batch Normalization normalizes the outputs of each layer.

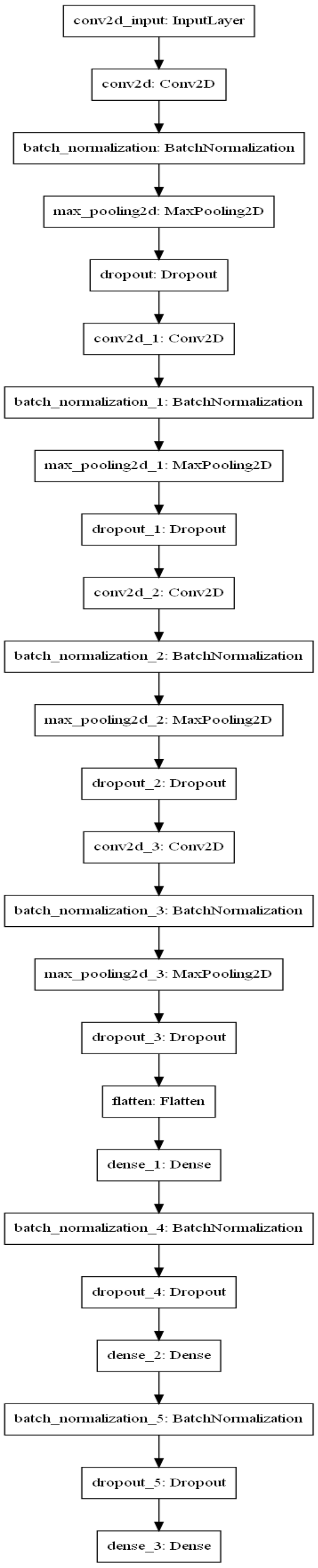


```python
def create_model(input_shape,output_dim):

    model = Sequential([
    Conv2D(filters=16, kernel_size=(3,3), activation="relu", kernel_initializer='he_uniform', input_shape=input_shape),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Dropout(0.25),

    Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Dropout(0.25),

    Conv2D(filters=128, kernel_size=(3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.4),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(output_dim, activation='sigmoid')
    ])
```

To compile our model we used Adam Optimizer with learning rate from 0.001, which is getting lower using ReduceLROnPlateau with a factor of 0.1, which is activated if, after 3 epochs, our model has not obtained a higher validation Recall. Also, we have used early stopping, which is also based on our model recall and if this metric has not been increased by 0.0001 after 20 epochs, it stops in order to avoid overfit. Finally, because we have a Multi-Label Classification problem and each class is not mutually exclusive, we have used a sigmoid function.

Model Summary & Plot

```
Found 16008 validated image filenames belonging to 20 classes.
Found 20012 validated image filenames belonging to 20 classes.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 222, 222, 16)      448       
_________________________________________________________________
batch_normalization (BatchNo (None, 222, 222, 16)      64        
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 16)      0         
_________________________________________________________________
dropout (Dropout)            (None, 111, 111, 16)      0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 32)      4640      
_________________________________________________________________
batch_normalization_1 (Batch (None, 109, 109, 32)      128       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 54, 54, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 64)        18496     
_________________________________________________________________
batch_normalization_2 (Batch (None, 52, 52, 64)        256       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 26, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 26, 26, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 128)       73856     
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 128)       512       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 128)       0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 12, 128)       0         
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2359424   
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024     
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 20)                5140      
=================================================================
Total params: 2,497,524
Trainable params: 2,496,276
Non-trainable params: 1,248
```


conv2d_input: InputLayer
conv2d: Conv2D
batch_normalization: BatchNormalization
max_pooling2d: MaxPooling2D
dropout: Dropout
conv2d_1: Conv2D
batch_normalization_1: BatchNormalization
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_2: Conv2D
batch_normalization_2: BatchNormalization
max_pooling2d_2: MaxPooling2D
dropout_2: Dropout
conv2d_3: Conv2D
batch_normalization_3: BatchNormalization
max_pooling2d_3: MaxPooling2D
dropout_3: Dropout
flatten: Flatten
dense_1: Dense
batch_normalization_4: BatchNormalization
dropout_4: Dropout
dense_2: Dense
batch_normalization_5: BatchNormalization
dropout_5: Dropout
dense_3: Dense

Results

Metrics

The accuracy of the Image Classification CNN is 25%. This means that our model did not perform well overall since it is the ratio of the correctly predicted observations to the total observations.

Precision which is the ratio of the correctly predicted positive to the total predicted positive observations was 60%. (Not accurate - we can reject this measure from the confusion matrices, since we have a small amount of total predicted positive observations, hence higher precision)

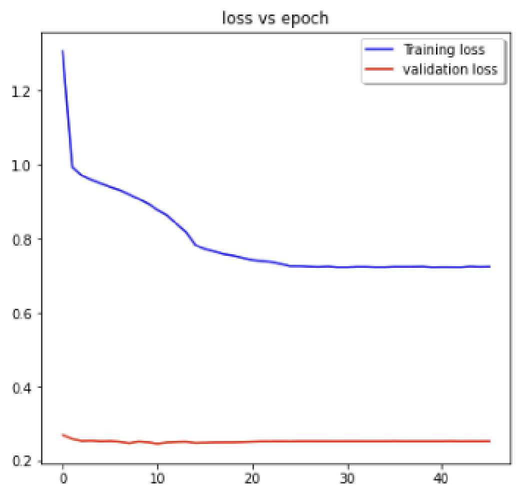
Recall was 26% which is the ratio of correctly predicted to all the observations in the actual class.

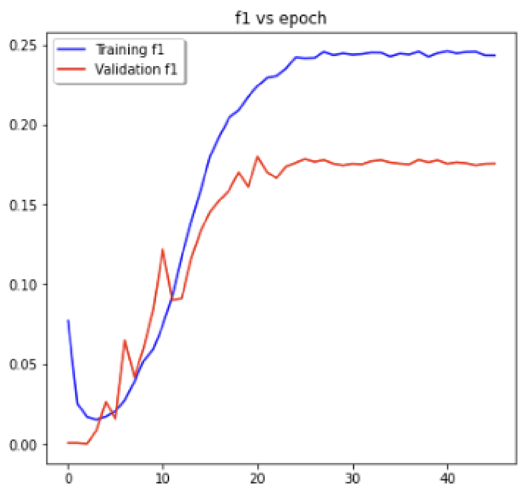
F1=14% which is a weighted average of precision and recall and works better when we have uneven class distribution, which is the case in our dataset.

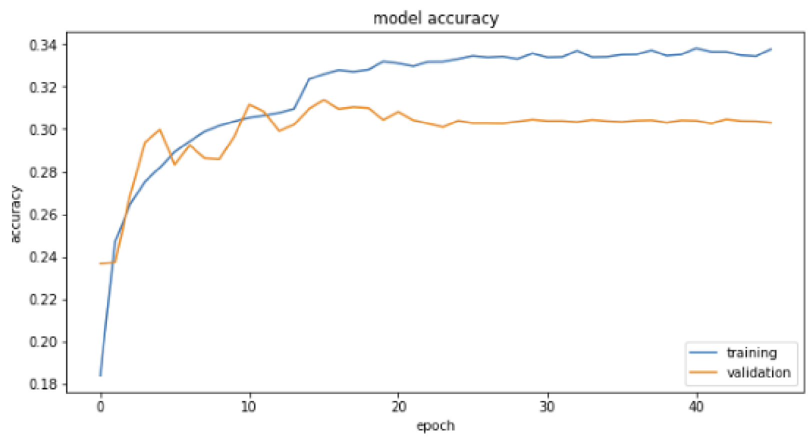
We run the training process using a Threshold of 0,5 for precision and recall computation, which gave us really low results (recall: 0.0895 - f1: 0.0948). After some tries, we decided that for the CNN model we could use a threshold of 0.4, which leaded in slightly better results, since we had little positive classifications with higher thresholds.

Learning Curve

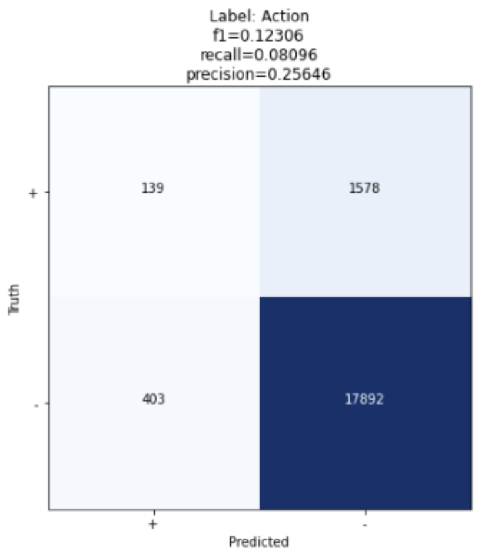
From the learning Curves, we can identify that the validation dataset appears easier for our model to predict than the training dataset.

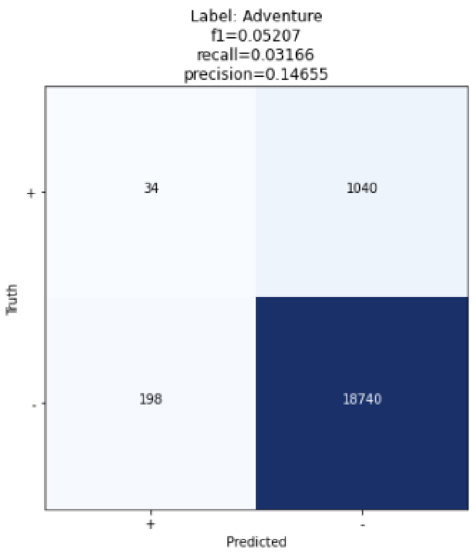


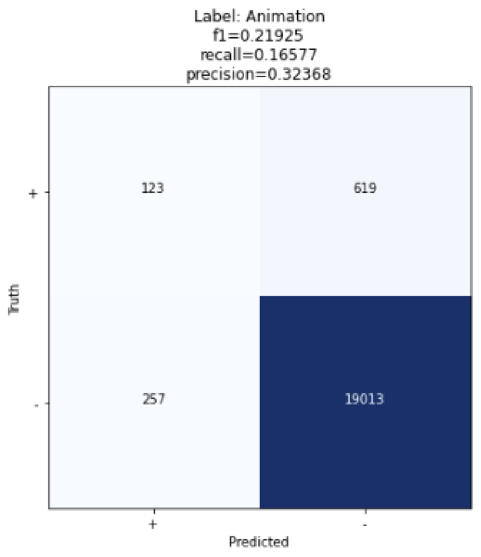


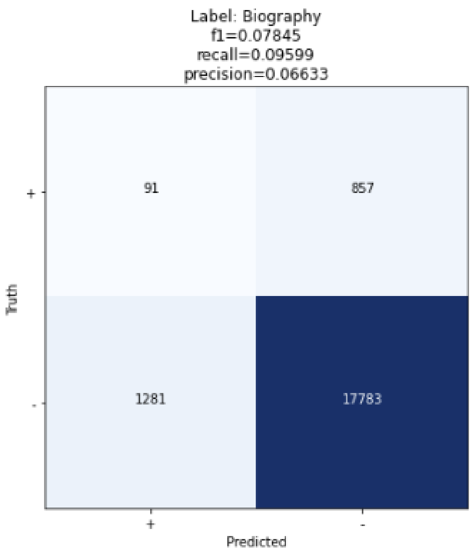


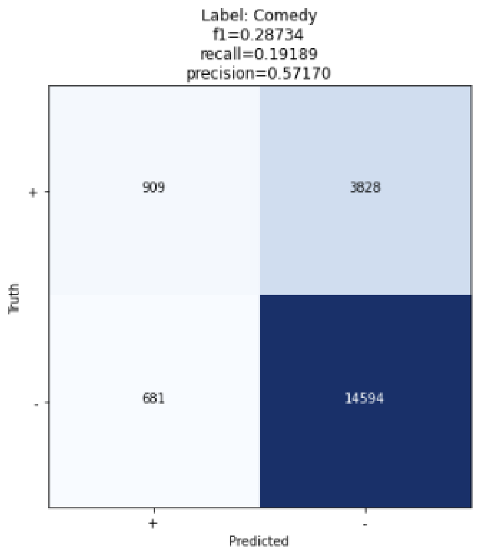
Confusion Matrix per class

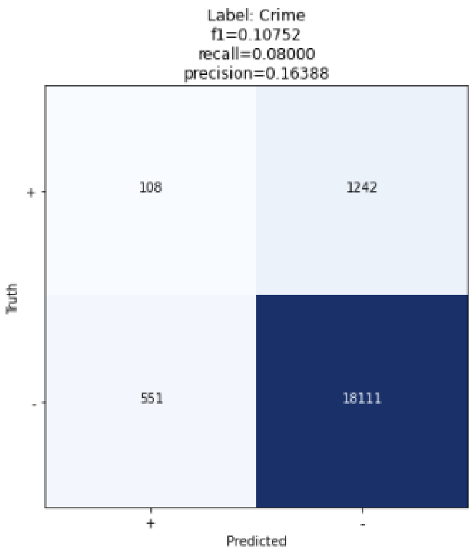


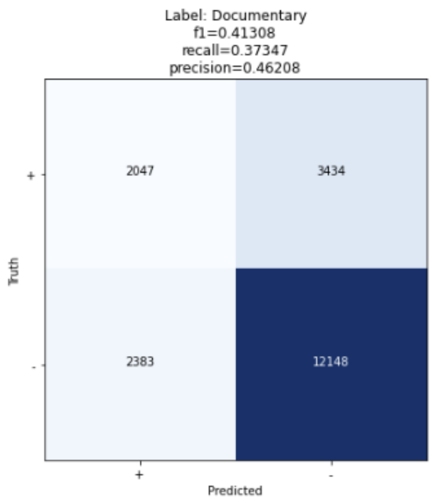


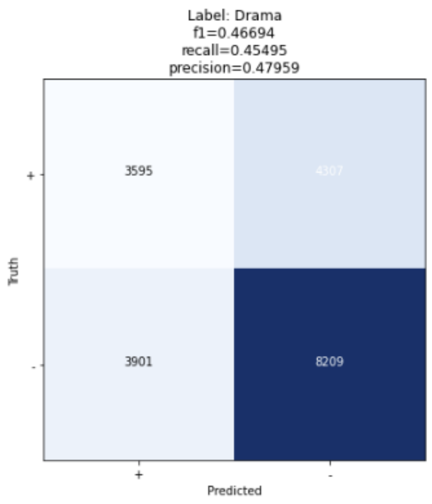


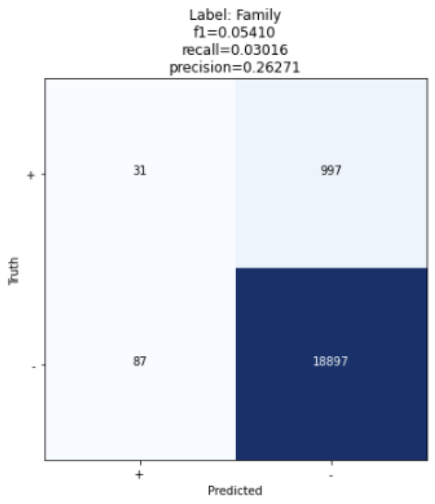


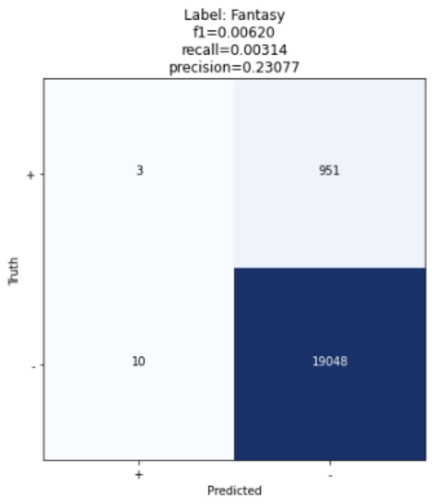


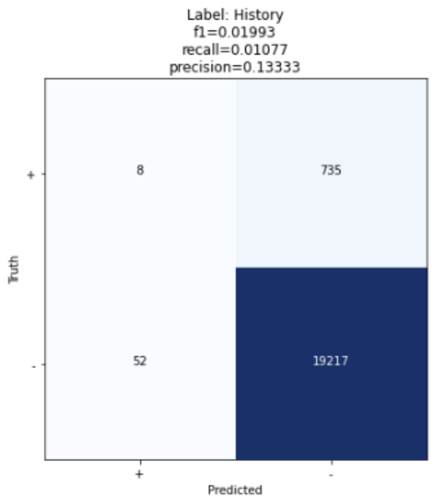


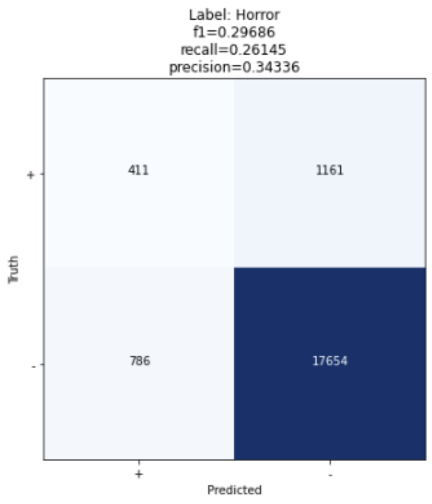

Label: Documentary
f1=0.41308
recall=0.37347
precision=0.46208
Truth
+
-
Predicted
+
-
2047
3434
2383
12148

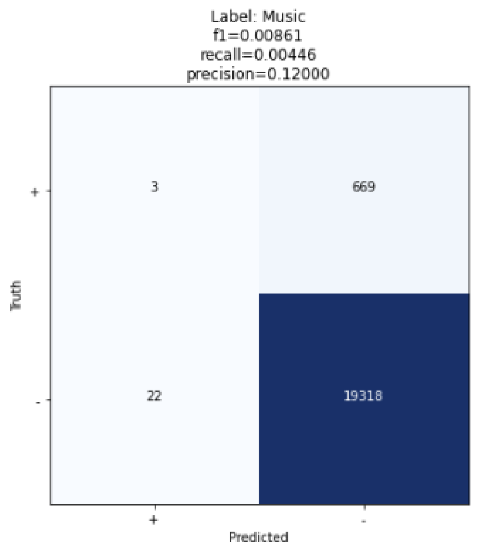

Label: Drama
f1=0.46694
recall=0.45495
precision=0.47959
Truth
+
-
Predicted
+
-
3595
4307
3901
8209

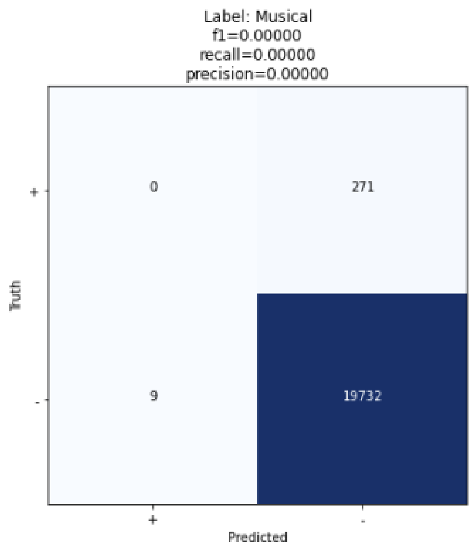

Label: Family
f1=0.05410
recall=0.03016
precision=0.26271
Truth
+
-
Predicted
+
-
31
997
87
18897

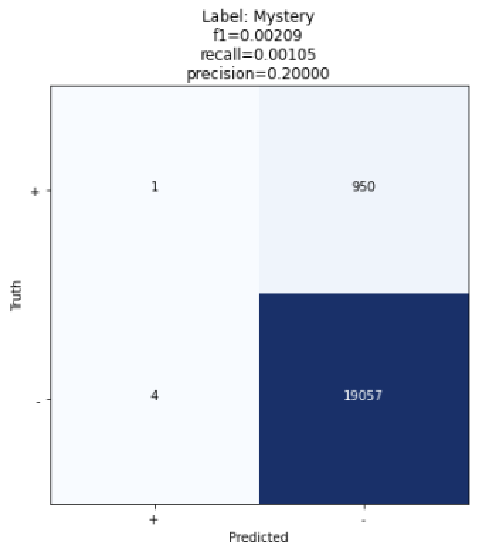

Label: Fantasy
f1=0.00620
recall=0.00314
precision=0.23077
Truth
+
-
Predicted
+
-
3
951
10
19048

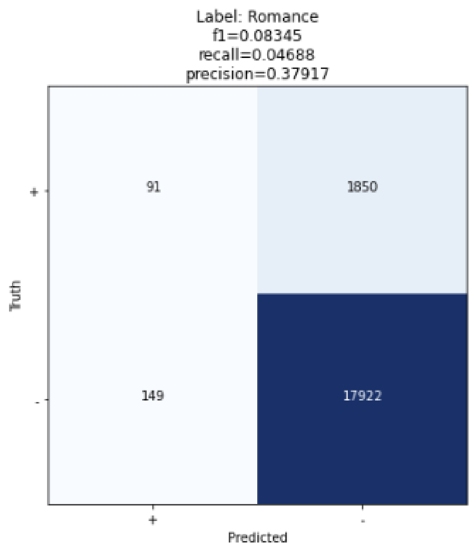

Label: History
f1=0.01993
recall=0.01077
precision=0.13333
Truth
+
-
Predicted
+
-
8
735
52
19217

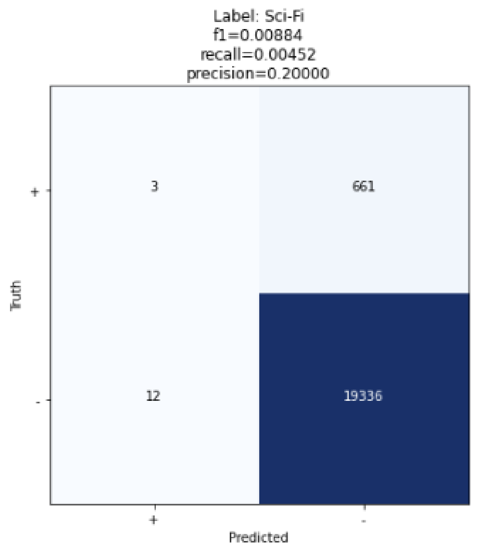

Label: Horror
f1=0.29686
recall=0.26145
precision=0.34336
Truth
+
-
Predicted
+
-
411
1161
786
17654

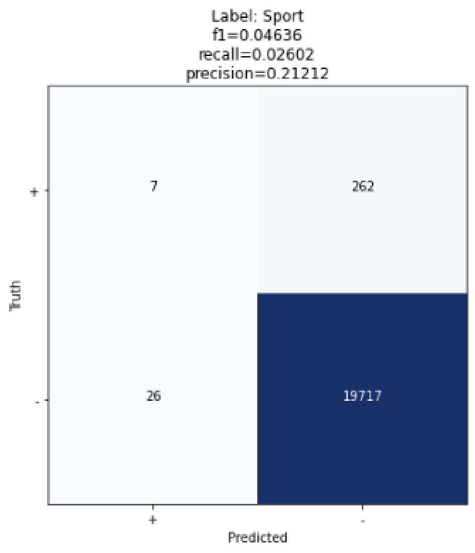

Label: Music
f1=0.00861
recall=0.00446
precision=0.12000
3
669
22
19318
+
-
Truth
+
-
Predicted

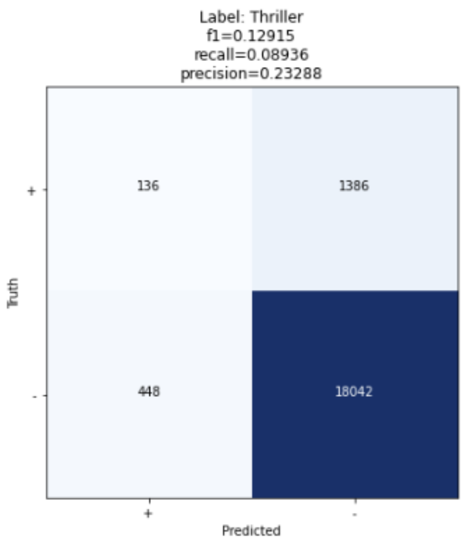

Label: Musical
f1=0.00000
recall=0.00000
precision=0.00000
0
271
9
19732
+
-
Truth
+
-
Predicted

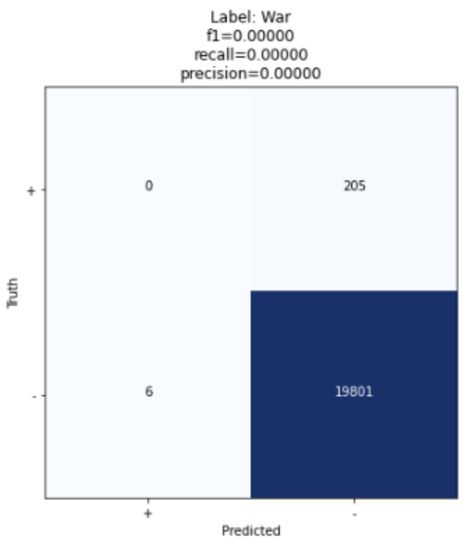

Label: Mystery
f1=0.00209
recall=0.00105
precision=0.20000
1
950
4
19057
+
-
Truth
+
-
Predicted


Label: Romance
f1=0.08345
recall=0.04688
precision=0.37917
91
1850
149
17922
+
-
Truth
+
-
Predicted


Label: Sci-Fi
f1=0.00884
recall=0.00452
precision=0.20000
3
661
12
19336
+
-
Truth
+
-
Predicted


Label: Sport
f1=0.04636
recall=0.02602
precision=0.21212
7
262
26
19717
+
-
Truth
+
-
Predicted





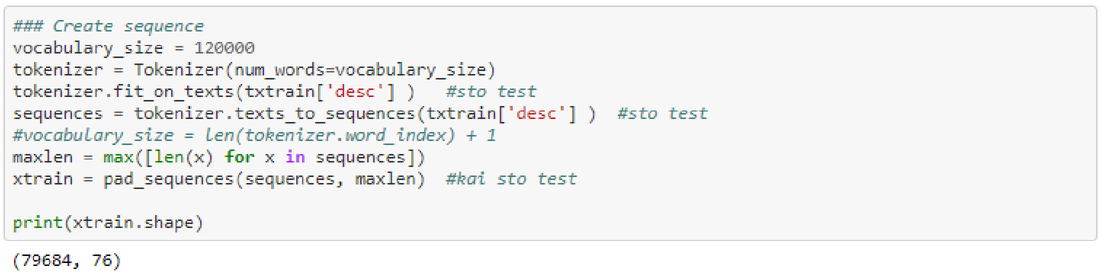
From the confusion matrix per class plot it seems that our model predicts the genre easier when dealing with posters from drama, documentary, comedy fims.

Text Classification CNN-LSTM

For this model, we had to preprocess our dataset based. We removed all movies that did not have a short description (NAs).

We split our data to training (80%) and test (20%) once more shuffling the data. Using regular expressions we cleaned the dataset and removed words such as “directed”, “by” and in general, words that appeared often in the majority of the descriptions. We also used stemmer, SnowballStemmer found and replaced each word with each stem respectively. We also removed stop words.

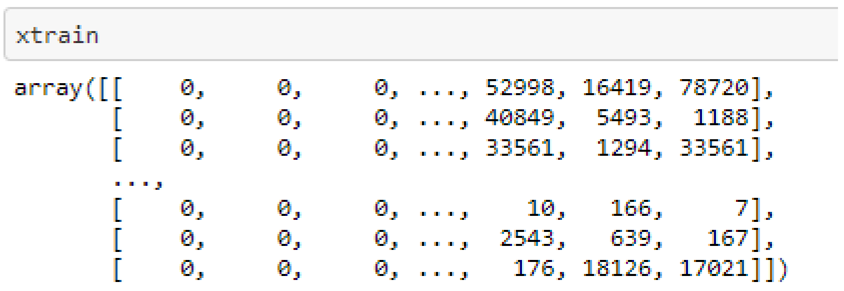
The words from each description cannot be used as inputs to our model, and so we had to tokenize our words and we kept a max vocabulary size of 120000. This means that we have kept the top 120000 words according to their frequency of appearance in our descriptions. The text was tokenized into tokens (sequences of numbers) based on the dictionary we have created and we also padded these sequences, because as we said the model has a standard input and we decided that the maximum input is 76 words.



```
### Create sequence
vocabulary_size = 120000
tokenizer = Tokenizer(num_words=vocabulary_size)
tokenizer.fit_on_texts(txtrain['desc'] )   #sto test
sequences = tokenizer.texts_to_sequences(txtrain['desc'] )  #sto test
#vocabulary_size = len(tokenizer.word_index) + 1
maxlen = max([len(x) for x in sequences])
xtrain = pad_sequences(sequences, maxlen)  #kai sto test

print(xtrain.shape)
```

```
(79684, 76)
```

This means that each text that had more than 76 words were replaced with 0.

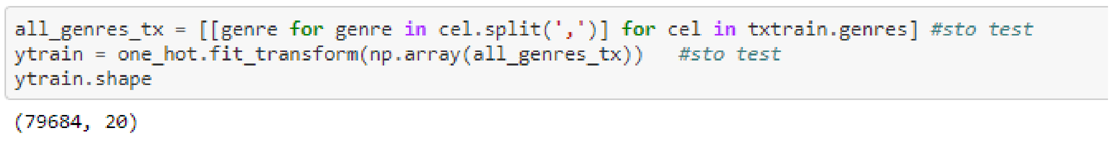


```
xtrain
```

```
array([[    0,     0,     0, ..., 52998, 16419, 78720],
       [    0,     0,     0, ..., 40849,  5493,  1188],
       [    0,     0,     0, ..., 33561,  1294, 33561],
       ...,
       [    0,     0,     0, ...,    10,   166,     7],
       [    0,     0,     0, ...,  2543,   639,   167],
       [    0,     0,     0, ...,   176, 18126, 17021]])
```

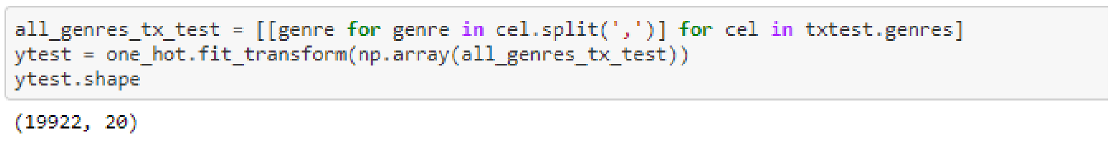
Each token basically is an index of the word.

OneHotEncoding was used in the movie Genre with the same procedure as before (1 vector with 20 binary variables, same count as our total genres in the dataset, which are 0 when the genre of the movie is not included and 1 when the genre of the movie is included.)



```
all_genres_tx = [[genre for genre in cel.split(',')] for cel in txtrain.genres] #sto test
ytrain = one_hot.fit_transform(np.array(all_genres_tx))   #sto test
ytrain.shape
```

```
(79684, 20)
```

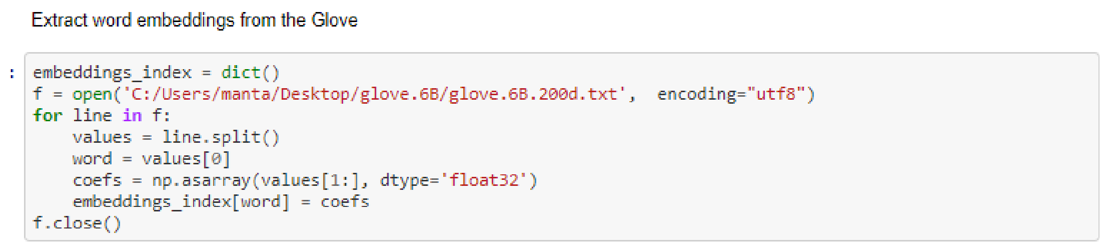


```
all_genres_tx_test = [[genre for genre in cel.split(',')] for cel in txtest.genres]
ytest = one_hot.fit_transform(np.array(all_genres_tx_test))
ytest.shape
```

```
(19922, 20)
```

To improve the model results we wanted to use the relationships between words, we replaced each token with one embeddings. The embedding of each word is a vector with length of 200 numbers, which describes this word, meaning that 2 words that are close, in terms of meaning to each other will have similar vectors. The Euclidian distance in a 200 dimensional space, which is the embedding size, will be smaller in comparison with two words, which are irrelevant with each other.

This replacement chanced the array of each text to 2 Dimensional.



Extract word embeddings from the Glove

```
: embeddings_index = dict()
  f = open('C:/Users/manta/Desktop/glove.6B/glove.6B.200d.txt',  encoding="utf8")
  for line in f:
      values = line.split()
      word = values[0]
      coefs = np.asarray(values[1:], dtype='float32')
      embeddings_index[word] = coefs
  f.close()
```

These arrays that derived from the text of each description were the inputs to the CNN we used.

To avoid training time and because the texts that we had were not domain specific and described diverse movies, we made the embedding not trainable.

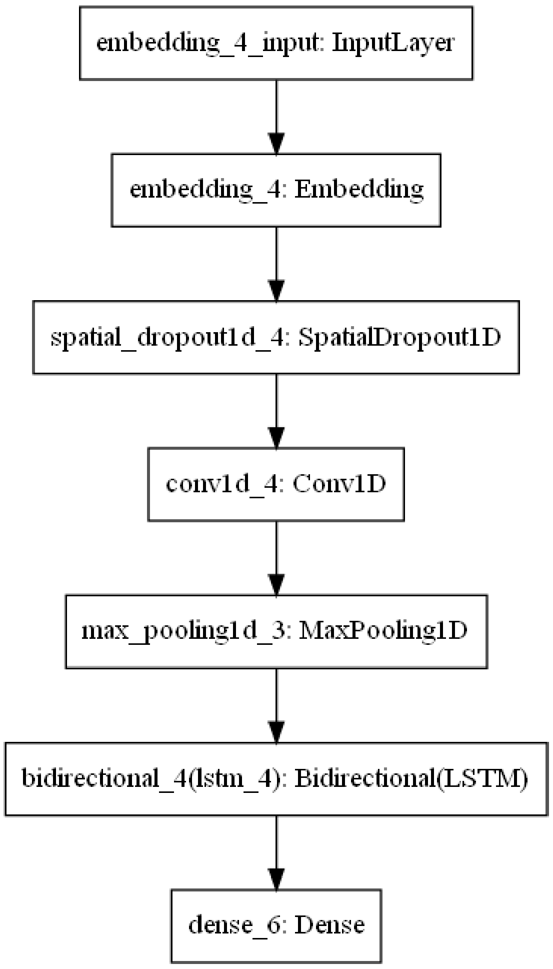
We used spatial dropout which deactivated 20% of the neurons and then created a bidirectional layer. This layer at each word, identifies the first word before and after.

The architecture that we used to classify the short description was a CNN-LSTM model.

First the Convolutional layer, works as topicalization, which is based on the features (words) that appear to be important and then the Bidirectional LSTM layer uses these features to find the connection of the words sequentially (words before and after). Then we used a MaxPooling Layer, to improve model speed and then it was passed through a dense layer, which works as the output layer with 20 neurons (same as our genres), with a sigmoid function. We used adam optimizer and the loss was binary-crossentropy.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_4 (Embedding)      (None, 76, 200)           24000000  
_________________________________________________________________
spatial_dropout1d_4 (Spatial (None, 76, 200)           0         
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 74, 64)            38464     
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 37, 64)            0         
_________________________________________________________________
bidirectional_4 (Bidirection (None, 256)               197632    
_________________________________________________________________
dense_6 (Dense)              (None, 20)                5140      
=================================================================
Total params: 24,241,236
Trainable params: 241,236
Non-trainable params: 24,000,000
_________________________________________________________________
None
```

Out[68]:


embedding_4_input: InputLayer
embedding_4: Embedding
spatial_dropout1d_4: SpatialDropout1D
conv1d_4: Conv1D
max_pooling1d_3: MaxPooling1D
bidirectional_4(lstm_4): Bidirectional(LSTM)
dense_6: Dense

Metrics-Learning Curve

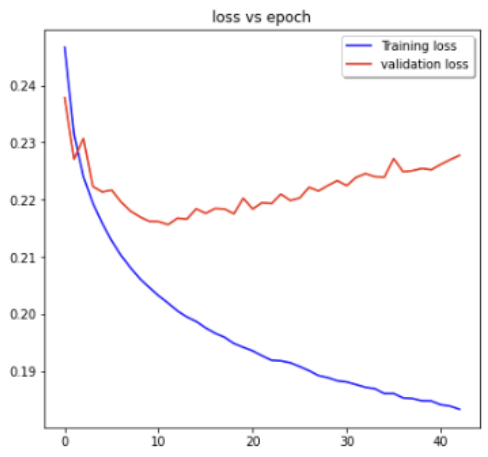
The accuracy of the Text Classification CNN-LSTM is 25%. This means that our model did not perform well overall since it is the ratio of the correctly predicted observations to the total observations.

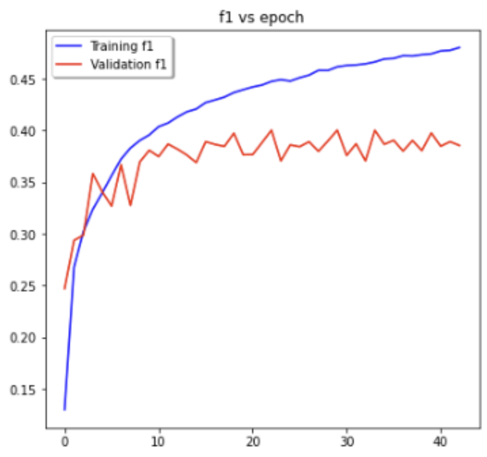
Precision which is the ratio of the correctly predicted positive to the total predicted positive observations was 34%.

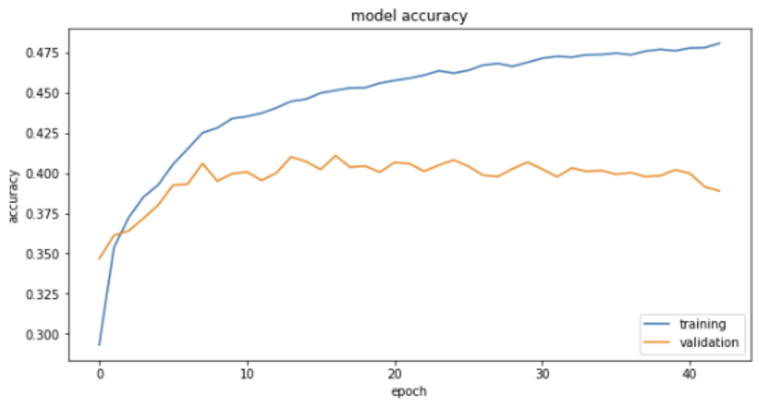
Recall was 25.62% which is the ratio of correctly predicted to all the observations in the actual class.

F1=25%, which is a weighted average of precision and recall and works better when we have uneven class distribution, which is the case in our dataset.

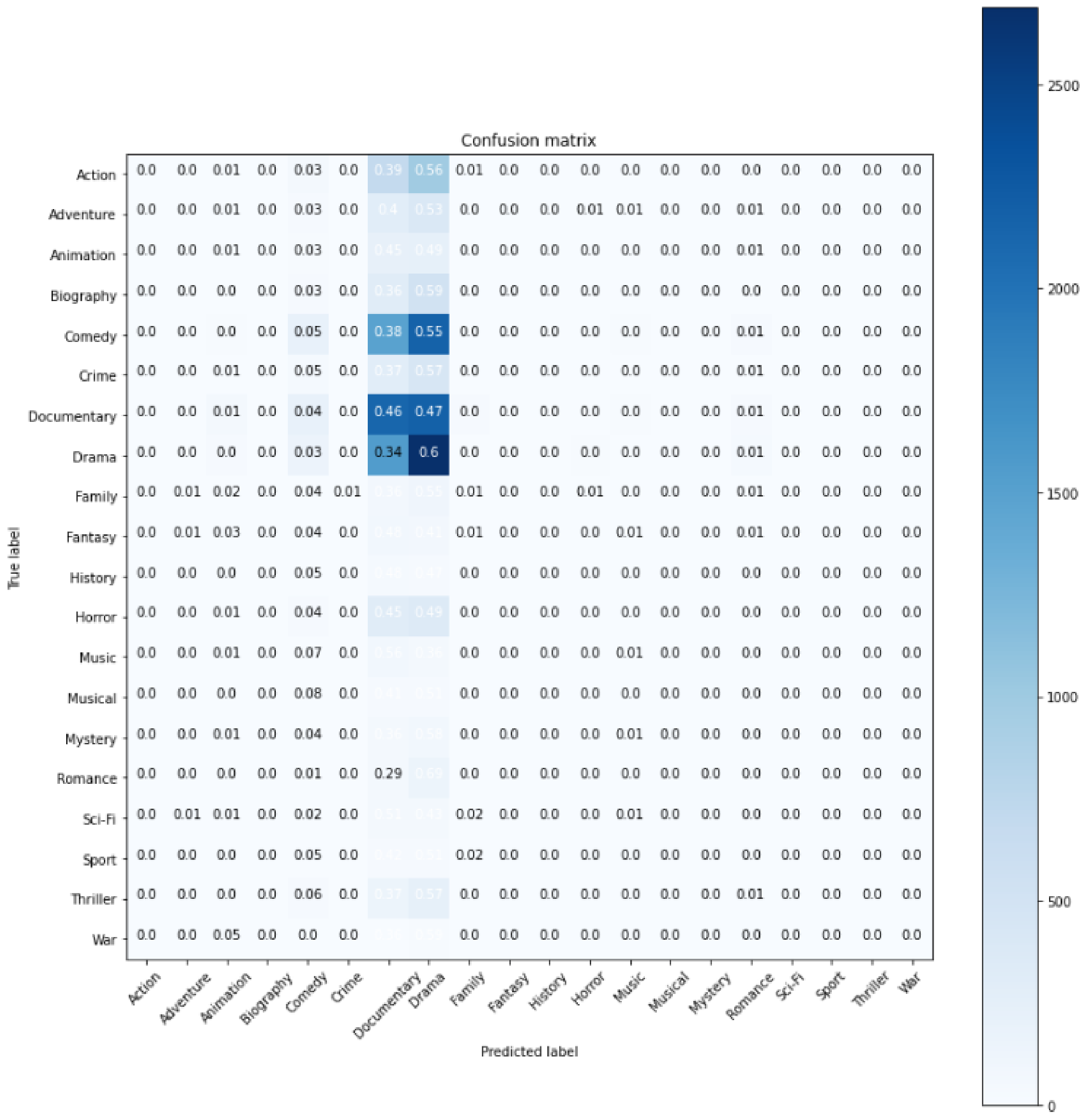
Regarding the Learning curve (loss vs epoch) we can observe that the training dataset has too few examples as compared to the validation dataset, hence the loss appears to improve over the first epochs, however a gap remains in both curves. F1 score and model accuracy suggest the same problem.







Confusion Matrix



From the confusion matrix, we can observe that the model performs better when dealing with comedy, drama and documentary film descriptions, which are the top 3 categories in our dataset in terms of frequency.

5. Members/Roles

Our team consists of 3 people and we have all agreed to participate in all the steps of this assignment.

- George Vogiatzis – Developer
- Michalis Prasinos - Business Intelligence Analyst
- Mirsini Zourou - Data Analyst

6. Bibliography/Links Used

https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11

https://github.com/keras-team/keras/issues/10371

https://link.springer.com/article/10.1007/s13748-014-0060-7

https://www.kaggle.com/hamishdickson/bidirectional-lstm-in-keras-with-glove-embeddings

7. TimePlan and Hardware Specs

Although we had a lot of time to prepare for this assignment, there were a lot of drawbacks, due to the very demanding second year of the part time program and the current situation with Covid pandemic (more working hours, hard to meet in person and work the assignment all together)

The TimePlan that derived from this was:

- Week 1: Web Scrapping
- Week 2-7: Model Development and Enhancements
- Week 8: Finalize Models, Reports and Presentation

Hardware: GPU Nvidia RTX2080, CPU Intel i7 6800k, 32Gb DDR4 Ram

8. Contact Person (τηλ. & email)

George Vogiatzis

Mob: 6978147098

Mail: vogias1988@gmail.com

9. Remarks/Comments

We consider this project worked well for the most part, except from the model's performance. By analyzing the nature of the problem and trying to pick the right technique to tackle each issue, we managed to strengthen the teamwork factor. Each member of the team had a valuable input to give in almost every phase. The most challenging part and maybe the source of the bad model performance was probably the dataset itself. Being greatly imbalanced, it was really difficult to produce good results. Maybe a solution to this would be to augment the data or try to scrape and create a more balanced dataset, although this would be more time consuming as the scraping part lasts for several hours. Lastly, we did not manage to implement the multi modal model. The idea was to deploy a multi branched model with mixed inputs of image and text data, where the first branch would perform the image classification while the second would perform the text one. To achieve this we should create a custom generator, which would, yield [image, text] batches out of our initial dataset and then feed this input to our model made using the Keras Functional Api, which allows the creation of more complex models. A first draft of our try is up on Github and we plan to continue and complete the third model also, apart from the deadline completion, because we consider it very interesting and it will add to our experience with more complex pipelines, a skill needed to deal with real life data.