Predicting Genre from Movie Posters

Manish Kumavat - 200050071
Nagendra Singh - 200050081
Arya Amin - 200050014
Lakshya Gupta - 200050066

Contents

1 Introduction

The film industry is incredibly reliant upon the use of posters to advertise movies in the hopes of increasing viewership and hence profits. A good poster is able to convey important qualities of a film such as theme and genre to make the movie seem as appealing to as wide of a viewership as possible. So we would like to train a model to learn features of a poster that could successfully predict the genre of the movie it represents. We found that a custom architecture was reasonably successful at predicting a poster's genres with around 90% accuracy. This implies that there is relative consistency in visual elements included in movie posters of a given genre.

1.1 Dataset

We used a dataset of around 7867 posters labelled with 25 different genres like War, Fantasy, Mystery, Science Fiction, Western, Comedy, Documentary, Crime, Action, Music, Adventure, Family, Thriller, History, Horror, Foreign, Drama, Romance and Animation.

1.2 Features (pre-processing)

We formatted each individual image into a 400x300 resolution image. This was done automatically using a preprocessing script. Our raw input data is the color of each pixel in the image expressed in terms of RGB values - a 400x300x3 matrix. Therefore, the specific features that we analyze are the RGB values at each of the pixels. The genres associated with a given movie are expressed as a length 25 vector. If a given movie belongs to a genre, the value at the associated index is 1. Otherwise, the value at the index is zero.

2 Baseline Algorithm

We implement an alogrithm which randomly guesses the genres of movies without even seeing the posters, we just randomly select 2 genres to label the movie with. This is useful because it serves as a baseline (and hence the name) or lower-bound for the performance of our other models and is a good metric to check if a particular strategy is worth following.
After testing this random baseline algorithm on a set of 2000 movies we got the following results,

All match	At least one match
1.22 %	17.388 %

Table 1: Baseline Algorithm Performance

3 ML-KNN Algorithm

We also implemented a Multi-Label k-Nearest Neighbor algorithm or ML KNN which uses the k nearest neighbors for every unseen training instance and then uses maximum a posteriori to determine the label set of this unseen instance.

This seemed to be a good approach for a few reasons. First, it seems like an intuitive approach towards identifying genre from movie poster - a multi-label classification problem. Second, experiments show ML-KNN achieves superior performance to some well-established algorithms in the literature.

We implemented KNN with hamming distance as the main metric for finding nearest neighours. Since kNN is computationally very expensive so we we were unable to run it on our computers with the full dataset. We were also unable to run on some large k-values, with the size of impossible to run k-values varying with dataset size. We maximized the amount of data that we could include in our analysis while also considering a reasonably large k value. We trained with a dataset of size 1000 with image preprocessed to 400x300x3 and got the following results,

K value	All match	At least one match
1	12.23 %	57.68 %
3	12 %	49.0 %
5	11.10 %	43.91 %
10	11 %	51.06 %
20	6 %	37 %
30	5.1 %	34.2 %

Table 2: ML-kNN Performance

4 Custom Architecture

Finally, we tried approaching the problem using a custom architecture as shown in figure 3. First, it employed the usage of 2D Convolutional Layers. We increased the number of filters as the network got deeper to extract increasingly specific features. Moreover, we utilized MaxPool layers in between the convolutional layers since it is common to do so. This helps in reducing the dimensions of our input, the number of parameters and also preserve important features while preventing overfitting. We also included Dropout layers, which ignore inputs from specific neurons with a certain probability. This counteracts the co-dependency that neurons develop in deep networks and regularizes our model. Our model uses a Flatten layer, which flattens our feature matrix into a column vector. This allows us to use fully connected layers. Additionally, we use two activation functions. The first is the Rectified Linear Unit function (ReLU) defined as y = max(0, x). We utilize ReLU after the flattening layer and after the convolutions since it is a common practice. The second activation function is the sigmoid, which is the final output function. Finally, we used an Adam Optimizer with a Binary Cross Entropy loss function to train our model.

$$loss = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

After training the model with number of epochs = 10, batch size of 64 and a learning rate of 10^{-3}, we got the following test results on the test set,

Dataset	Accuracy	All match	Atleast one match
Dev	90.4%	5.6 %	42.5 %

Table 3: CNN Performance

Genre	Recall	Precision	F1	Count
Drama	0.39	0.41	0.399	1412
Comedy	0.32	0.40	0.35	1009
Thriller	0.13	0.32	0.18	527
Horror	0.07	0.22	0.11	323
Action	0.06	0.19	0.09	458

Table 4: Best Classes by Performance

where, if *tp* is number of true positives, *fp* is the number of false positives and *fn* is the number of false negatives then,

$$Precision = \frac{tp}{tp + fp}$$
$$Recall = \frac{tp}{tp + fn}$$
$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

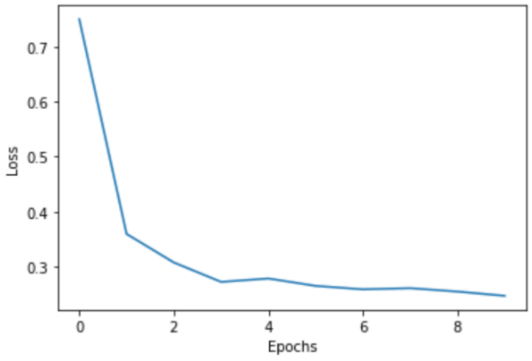

Figure 1: Loss vs Epochs

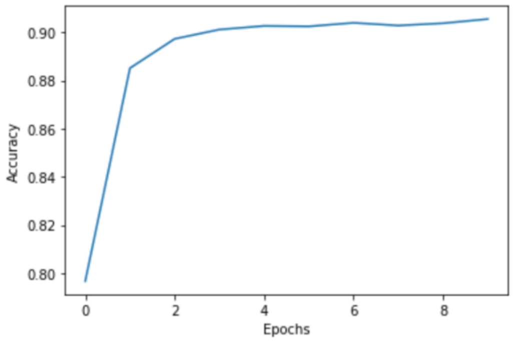

Figure 2: Accuracy vs Epochs

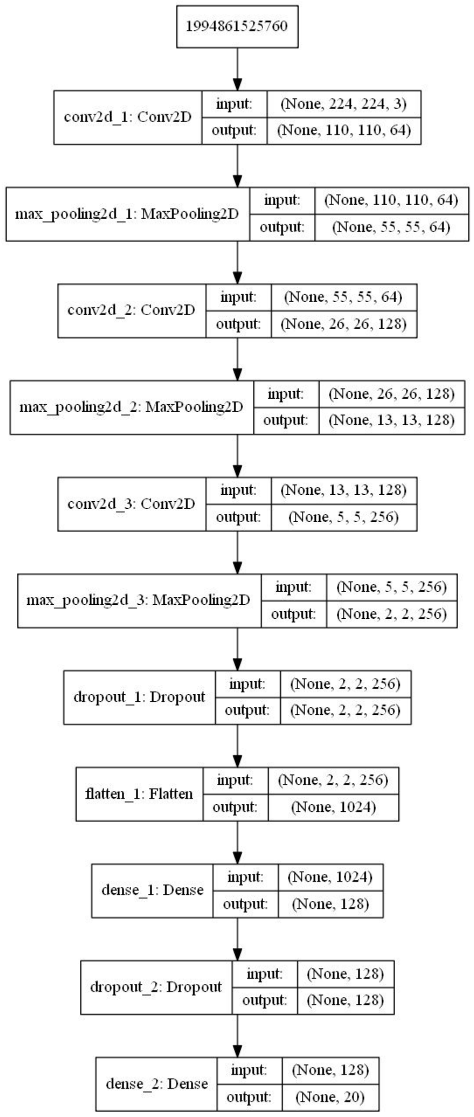
1994861525760
conv2d_1: Conv2D
input: (None, 224, 224, 3)
output: (None, 110, 110, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 110, 110, 64)
output: (None, 55, 55, 64)
conv2d_2: Conv2D
input: (None, 55, 55, 64)
output: (None, 26, 26, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 26, 26, 128)
output: (None, 13, 13, 128)
conv2d_3: Conv2D
input: (None, 13, 13, 128)
output: (None, 5, 5, 256)
max_pooling2d_3: MaxPooling2D
input: (None, 5, 5, 256)
output: (None, 2, 2, 256)
dropout_1: Dropout
input: (None, 2, 2, 256)
output: (None, 2, 2, 256)
flatten_1: Flatten
input: (None, 2, 2, 256)
output: (None, 1024)
dense_1: Dense
input: (None, 1024)
output: (None, 128)
dropout_2: Dropout
input: (None, 128)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 20)

Figure 3: CNN Architecture

5 Conclusions

As can be seen, our custom model fares decently on the test dataset with a pretty high chance of predicting at least one genre correctly. Though as is pretty apparent our model fails to predict all of the genres correctly which is partly due to the unbalanced dataset we've used which has overwhelmingly high number of dramas, comedy genres so the model defaults to these genres in case of uncertainty. Our model also shows that across posters, there seems to be a consistent set of features that occurs within those of a specific genre.
Another reason is that due to lack of technical resources we were forced to use only 2000 image dataset so the model likely missed some details here and there.

Algorithm	All match	At least one match
Baseline	1.22 %	17.388 %
CNN	5.6 %	42.5 %
ML-kNN	12.23 %	57.68 %

Table 5: Algorithms' comparison by their best performance

6 References

- Dataset
- ML-kNN
- CNN