Movie Poster Benchmarking

Priyanka Chakraborti *University of Nebraska-Lincoln*
Bryan Moore *University of Nebraska-Lincoln*

I. INTRODUCTION

With the advent of online streaming production houses are facing as much, or more, competition from independent movies in garnering an audience base. Billion dollar revenue figures are commonplace for production houses, placing many major movie budgets alongside that of revenues collected by companies such as Facebook. With so much riding on a film's success, creative designers are constantly looking to redefine their strategies in designing interesting and iconographic movie posters. It is paramount that viewers to get a feel for what type of movie they will see, yet show just enough to get them hooked.

Seeing as there have been hundreds of thousands of movies released, and a corresponding poster for nearly every one, is there a trend which we can use to determine which movies a new movie poster will most strongly evoke a response? Neural networks have traditionally found tremendous success with image recognition. The goal of this project is to see if a similar network can be used to make movie recommendations by making visual discoveries in the movie posters themselves. What makes this all the more interesting is the classification functionality of our algorithm hinges on auto-tagging of the movie posters by users which makes this a multi-label problem.

The input to our algorithm is any poster from a movie that our customer is interested in. To train our model we harvested over 80,000 movie posters, and other important information, from IMDB. We then utilized transfer learning from the well known convolutional network, VGG-16, to classify the movie poster into its most likely genres. These label predictions will be boosted by passing a feature vector through a graph-based embedding and classification pipeline. The key functionalities in our algorithm are as below:

(a) **Classification**: A feature vector is extracted from the 2nd to last layer and used to compute similarity metrics with respect to all other movies in our database that belong to the classified genre(s). The final recommendations are the top five movies based on the closeness of their feature vectors from the stored pool of movies for the predicted genre(s).

(b) **Recommendation**: The efficacy of our recommendation is optimized using a flask app hosted on an AWS server. This employs a multi-armed bandit approach to determine whether using Manhattan Distance, Euclidean Distance, or Cosine Similarity is best in practice for finding similar posters. The reward is supplied by users submitting their feedback via simple to understand buttons in the app. Over time the bandit accumulates rewards from live feedback to learn the best metric to decide on closeness of feature vectors.

Of course, this approach has many other applications. Although here we are focusing on movie posters, the same approach could be used for benchmarking advertising campaigns. Instead of users providing feedback directly we could instead use the click rate of buttons as user feedback, and over time choose to show the ads which are more interesting to our target audience(s). That said, movie posters are our first attempt at a reinforcement driven neural network recommendation, and the focus of this project.

The article is arranged as follows. The scraped data is described in detail in Section II. Section III describes the neural network architecture. The graph based embedding and classification pipeline is described in Section IV. The reinforcement learning algorithm, and corresponding flask app, are described in Section V. We elaborate our results in Section VI, and our conclusions are presented in section VII.

II. Data Summary

When scraping the movie posters we utilized Selenium to replicate a headless browser that can automate clicking on pages. We then made queries to each of the 18 available genres using IMDB's search functionality. These genres are Action, Adventure, Animation, Biography, Comedy, Crime, Drama, Family, Fantasy, History, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, and Western. These subsume the major autotags used by most recognized streaming webservice. From the linked pages we scrape the movie poster, the name of the movie, the release year, the genres it is tagged with, the IMDB rating, and the number of people who rated the movie. The posters vary slightly in size, with most tending to be about 180x280 pixels.

For each of the genre searches mentioned we continue scraping movies until we have examined 12,000 movies tagged with that genre, or have run out of tagged movies to scrape. Because this problem is fundamentally multi-label, this process of course leads to us obtaining many duplicate movie posters. After adjusting for this we find that we have scraped information for 81,266 unique movies.

III. Neural Network Architecture

For this project we are utilizing the VGG-16 pre-trained network. VGG-16 has shown significant success in separating foreground from background in images, with the foreground being the primary object for the classification (such as bird against the sky) [1]. VGG-16 contains 3 convolutional layers, with max-pooling every 2 or 3 convolutional layers. This is followed by 3 fully-connected layers and softmax as the final layer. The original model was trained to classify images in the ImageNet-ILSVRC-2014 contest, where there were 1000 categories. When hosting a neural network on cloud services network memory size can often be a restriction. To this end VGG-16 is compact, yet powerful.

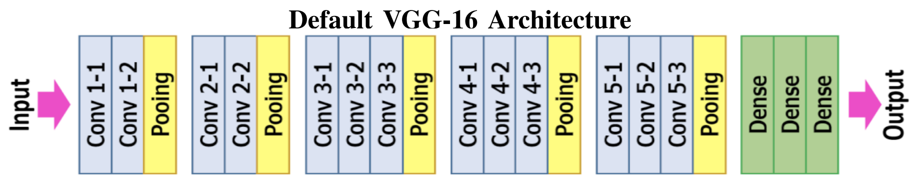

Fig. 1: This shows the layers of the VGG-16 standard architecture. The picture is sourced from Hassan's article [2]. For our architecture we removed the shown dense layers and replaced those with dense layers of our own.

We freeze the training of conv blocks 1 through 4 and strip off all layers starting with the very last max-pooling layer to the softmax layer at termination. We then add four dense layers on top of VGG-16's conv block 5. The last layer retained from VGG-16 has an output dimension of 8x8x512. We follow this with a dense layer of 10,000 neurons, followed by a dense layer of 2,000 neurons, a dense layer of 512 neurons, and a dense layer of 100 neurons. Each of these layers is separated by a dropout layer with a dropout rate of 0.5

For our case we have 18 possible genres, although as discussed later in this paper we will reduce this further to 6 genres. Thus, the final dense layer will have the same number of neurons as the number of genres we are trying to predict (either 18 or 6). The classification is done by either a Sigmoid or Softmax layer. Which activation layer will be utilized depends on whether we consider the problem in the multi-class space or the multi-label space. This is discussed in detail in the following subsections. For gradient optimization during back-propagation we use the Adam Optimizer, with default hyperparameters, and a batch size of 32.

A. *Image Augmentation*

For use in our neural network each poster has been resized to 100x100, as this has been found during our investigation to be a good balance between predictive power and memory management. During training each batch of images goes through random data augmentation. These include horizontal flips, random zooming using a ratio of up to 0.5, shifting all pixels horizontally up to a fraction of 30% of the width, shifting all pixels vertically up to a fraction of 30% of the width, and randomly rotating the image by up to 30 degrees. From our investigations, applying augmentation to the train posters did show a small improvement in predictive power on the test set without significantly affecting training time.

It is very important to note that each of these movies can be tagged with anywhere from 1 to 3 genres. After one-hot-encoding the genres each movie is thus associated with a sparse label array. There are two main ways to tackle a problem like this. We can choose to remain in this multi-label space or we can choose to instead work in the associated multi-class space. In the multi-class space we consider every possible combination of genre labels to be its own unique class. Below we briefly discuss the results of our exploratory data analysis while working in each space.

B. *Working in a Multi-Class Space-Label Power set*

Noting that there are a finite number of unique combinations of these labels we could choose to work in a space where each movie is tagged with only a single class. In this case each class is the unique combination of genres it is tagged with by IMDB. Using all 18 genres previously mentioned, there are 757 distinct classes. This is clearly an intractable problem. Thus, if we are to consider the multi-class space it is clear we must reduce the number of genres we consider. While superficially the exclusion of some genres may appear to alter the basic premise of the problem, our justification is that the many of the omitted genres were sparsely tagged and hence were left with very few movies such that they could have been removed by naive filtering. Those which were not sparsely tagged could be considered subsets of those genres we choose to keep. The auto-tags are primarily meant to preserve context. For example, a biographical movie is more often classified as ’drama’ and seldom as ’biography’. To that end, the posters should like-wise reflect the context with little difference than for any other drama movie.

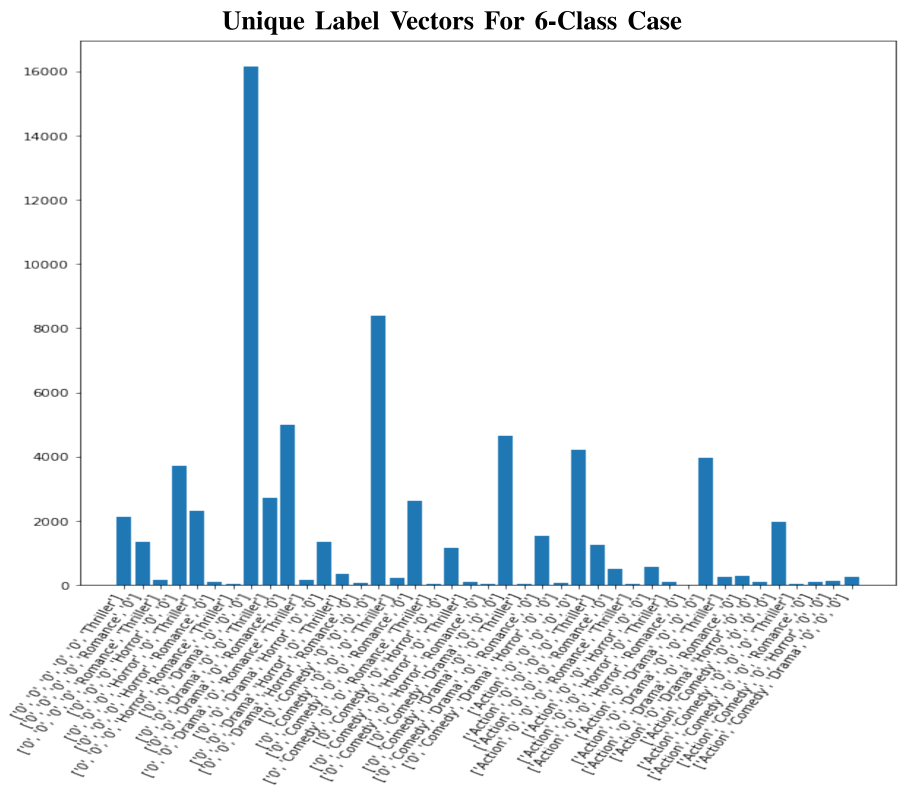

Fig. 2: This shows the number of images tagged with each of the unique combination of class labels. There are 41 unique label combinations.

Using only the genres Action, Comedy, Drama, Horror, Romance, and Thriller there are 68054 movies tagged with at least one of these genres. This results in 41 distinct label combinations. However, as shown in Fig. 2, there is an intractable amount of class imbalance. This cannot merely be fixed with random removal of images from certain classes and creating noisy copies of other classes. In this multi-class space we would use cross-entropy as our loss function.

We do not have a strong argument for removing even more genres. Removing even more would exert an unfair bias towards the genres we personally consider to be important. Thus, the only robust option remaining appears to be to remove many of the label vectors altogether. We have explored this to see if the removal of multiple classes provides a more powerful model in terms of poster predictions than the inherently multi-label model. The resultant prediction accuracy is, on average, quite low. Thus, working in the multi-class space for this dataset is not a reasonable approach. We have found that working in the multi-label space provides much more powerful results.

C. *Working in a Multi-Label Space*

For the multi-label case we will first consider the case where we use all 18 genres. For multi-label problems we used a Sigmoid layer for our terminal activation layer. This is equivalent to computing the probability to tag the poster with each genre label. For our loss function we utilized binary cross-entropy as for each label the network needs to decide whether or not it should be recommended independent of all other labels. This differs significantly from standard cross-entropy, as that compares the entire label vector and assumes each image belongs to only one distinct class.

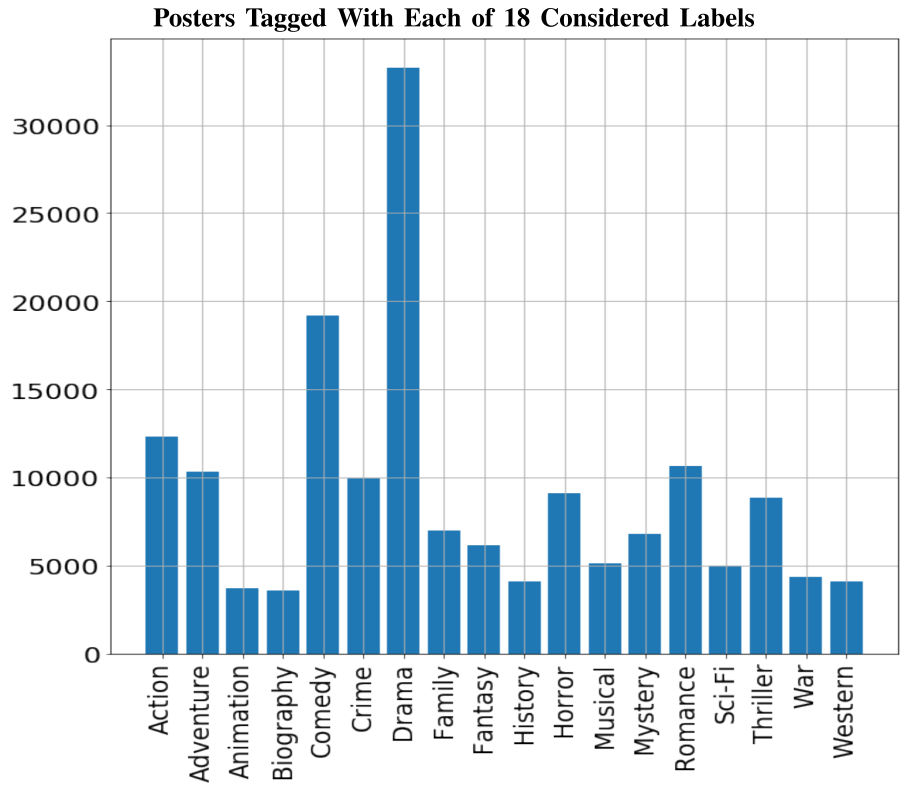

Fig. 3: This shows the number of posters tagged with each of the 18 considered genres. There is extreme label imbalance between many of the genres.

Running the train set through our model with a learning rate of 10^{-5}, which we found to be optimal for this configuration, results in a micro avg f1-score of 35% after 30 epochs. When training we weighted the labels by inverse of the number of posters tagged with each to reduce the impact of the over-represented labels. We use the micro average as the macro average would treat all classes with equal weight which, due to the previously mentioned imbalance, is not a good assumption. Examining the results for each label shows that some labels have an f1-score of zero. Fig. 3 shows the number of posters tagged with each label. Seeing as there are so many different combinations of label vectors it can be seen that it is not possible to adequately correct this imbalance without severely limiting the number of samples we have left for training.

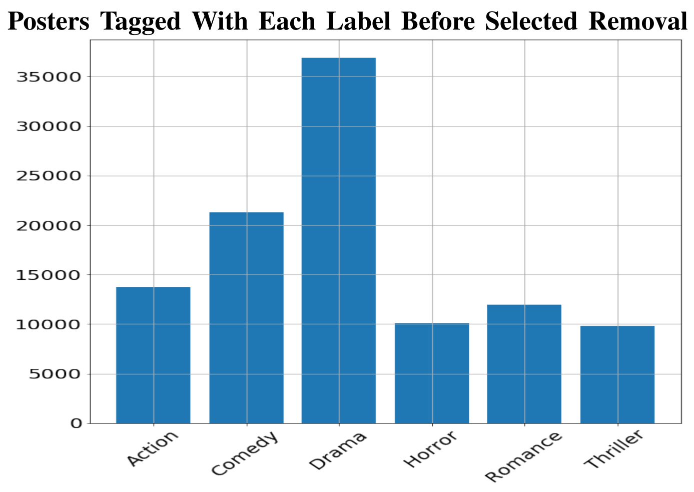

Fig. 4: This shows the number of posters tagged with each of the 6 considered genres without removing any movies during training.

We thus investigate the 6 genre multi-label case, also using a learning rate of 10^{-5}. The training produces a micro avg f1-score of 33% after 30 epochs. We again weighted the labels by the inverse of the number of posters tagged with each label during training. This f1-score still quite poor, and there are also multiple labels with a f1-score of zero. As seen in Fig. 4, there is still significant imbalance between the labels. However, in this case we can now help reduce this imbalance through random removal of movies.

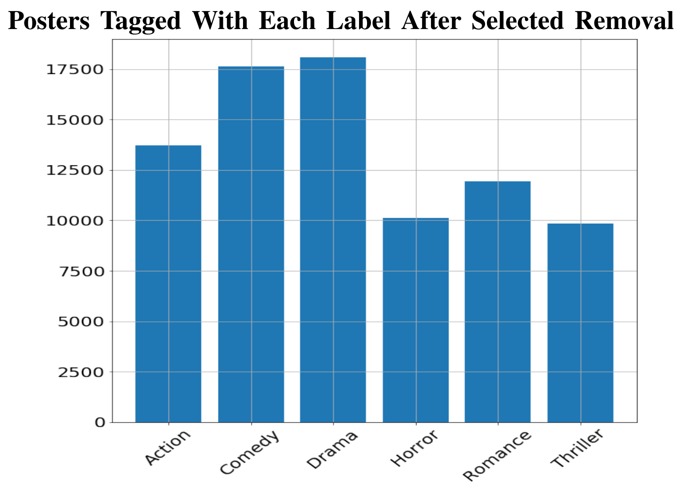

Fig. 5: This shows the number of posters tagged with each of the 6 considered genres after removing all but 1,000 tagged exclusively with Drama and all but 1,000 tagged with Drama and Comedy. Note the label imbalance is decreased significantly after removal.

By removing all but 1,000 movies tagged with only Drama, and all but 1,000 tagged with both Drama and Comedy, we are able to reduce the imbalance significantly, as shown in Fig. 5.

This is the label distribution we use for the remainder of our investigation. We have set aside 10% of the dataset for testing, and 10% of the remaining train-set for validation. This leaves us with 39901 train samples, 4433 validation samples, and 4925 testing samples. Using this our micro avg f1-score is 39% after 30 epochs. There are also now no posters with f1-scores of zero, although there is still significant room for improvement. Further details of these results are presented in Section V.

IV. Graph-Based Embedding and Classification

As noted previously, our trained neural network performs adequately. To improve on this we added an additional step. This is a pipeline which incorporates contextual information from the tagged labels for each movie poster. The goal here is to be able to capture the correlation between co-occurring labels. Perozzi et al's Deep-Walk algorithm [3] shows that latent label representation in a manifold eigen space can be an extremely powerful tool for recognizing textual correlations in social network platforms. Due to the limited scope of this project we explore a condensed version of this approach via an input graph G_{vgg}, whose nodes represent each movie label. The edges of the graph are weighted by the frequency of node co-occurence. The weighted graph for our dataset is shown in Fig. 6

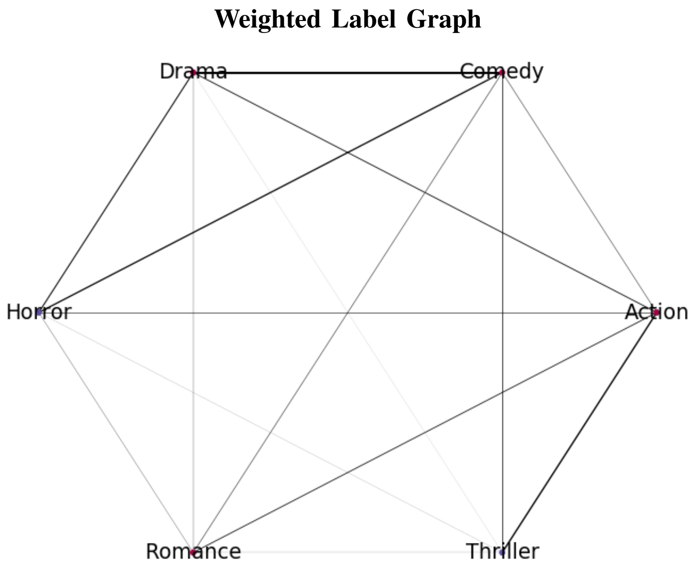

Fig. 6: This shows the weighted label graph for our 6 genre dataset. Thicker connecting lines indicate stronger correlation between labels.

Within this graph space we define an embedding classifier whose learning encompasses two step:

a. Find an appropriate cluster partition for this space.

b. Identify which cluster a new movie instance belongs to.

By nature, a graph is a discrete algebraic structure. An embedding maps it into a continuous space as shown below.

$$E_i = \zeta(x_j \in X_{vgg}; y_{i,j} \in G_{vgg}) \tag{1}$$

Here ζ is the embedding function and X_{vgg} is a matrix holding all features. Given E_i as the ground truth, the random forest regressor is trained to map a new instance of X_{vgg} to its embedded representation within the constructed space. A KNN classifier is then used to predict the multi-label tags for the input poster by identifying the nearest cluster to $E \times X$

The multi-label embedding package we used is CLEMS (Cost Sensitive Label Embedding with Multi-Dimensional Scaling) [4]. The training stage of this architecture works to identify an optimum embedding in which each instance is penalized based on the Hamming distance between the predicted label vector and the ground truth. One key drawback to this pipeline is that it cannot be concurrently trained with VGG-16 due to the inability to propagate gradients through either KNN or the random forest regressor. This would have been optimal, but in lieu of that, an expectation maximization approach which alternates training between the CNN and the label embedding pipeline can be used. Due to time constraints this has bnot yet been implemented. We have trained the pipeline separately with the feature output of VGG-16, and generated predictions from that. In Section VI we present preliminary results which already show marked improvement over using only VGG-16.

V. Bandit Learning and Recommendation

Even after we have optimized our neural network and implemented the graph-based embedding and classification pipeline, the task of finding the most similar movie posters to the movie poster uploaded by the user still remains. As we do not have pre-recorded data indicating how well any of the similarity methods being considered (Manhattan, Euclidean, or Cosine Similarity) actually work, we have decided to implement a multi-armed bandit approach. This assumes an equal prior probability for each arm. The main advantage of utilizing this reinforcement algorithm is that it is able to minimize the number of "bad" predictions we make during the time the algorithm is learning the optimal approach. We have set up the bandit as follows.

Each time a recommendation is made the app chooses a similarity metric to evaluate. The output of the graph-based embedding label predictions is used to restrict the movie posters to choose for comparison. If the output of the graph-based embedding pipeline indicates that the probability for any particular label is greater than 50% we include any posters tagged with that label to be included for comparison. This helps exclude movies which are nearly certainly unrelated to our uploaded poster, thus reducing the noise in our comparison.

As we have three considered similarity methods we will obviously be choosing to implement one each time a user seeks a recommendation. Choosing a similarity metric is equivalent to pulling an arm of the bandit. The goal is to encourage the bandit to initially explore different arms, and over time learn from live user feedback to pull the arm that gave it the most positive feedback from the user. To this end we implement the epsilon-greedy approach. Each time the user requests a recommendation the program randomly generates a probability. This probability determines whether the bandit will 'explore' or 'exploit'.

Epsilon is typically a fixed value. We have initialized it as 1 for the first prediction and set it to decrease with time towards 0. If a randomly generated number between 0 and 1 is greater than epsilon the program will use the similarity measure which, so far, has the highest value. If the randomly generated number is less than or equal to epsilon the program will randomly choose a similarity measure to utilize. Epsilon is updated as:

$$\epsilon = \frac{\epsilon_{decay}}{num_{preds} + \epsilon_{decay}} \tag{2}$$

where num_{preds} is the number of movie posters for which user feedback has been received. This means that the probability of exploring similarity measures, other than that which has the highest value, will decrease inversely with the number of times feedback is received. Note that we have intentionally chosen to not count recommendations for which no feedback has been provided. This is because without this feedback we have no way of knowing whether the recommendations provided are good or bad, and thus no way to determine whether the method used works well or not.

When updating the reward value for any particular action we use the equation:

$$action_{value} = \left(\frac{n-1}{n} * prev_{value} + \frac{1}{n}\right) * new_{reward} \tag{3}$$

where n is the number of recommendations rated by users, $prev_{value}$ is the value for that action before this update, and new_{reward} is the reward given by the user ($+1$, -1, or 0).

This information is communicated to the user via an app we have created using Flask. This is currently hosted on AWS here:
http://posterbenchmark-env-1.tvdz8pnqtu.us-east-2.elasticbeanstalk.com

The interface allows users to upload a movie poster they have sitting on their computer, link to an IMDB page (whereby the backend will automatically locate and scrape the relevant movie poster), or click a button to see the results for one of the 100 sample movie posters we have uploaded with the app.

The flask app then runs the movie poster through the neural network, with appropriate reshaping of the image, and predicts the genre labels. It also extracts the predicted feature vector and uses the chosen similarity metric (as described previously) to find the most similar 5 movie posters. These are displayed to the user. Due to memory constraints the posters for all movies in our train set are not stored on AWS, but are scraped dynamically from IMDB after the model has completed its predictions.

In addition to displaying the posters, there is also a scatter plot showing the distribution of years vs IMDB rating of the similar movies found. This should be useful to someone creating a movie poster as it provides a quantitative metric for the decade their poster may be evoking a feeling of nostalgia to, and the quality of movies it is reminding people of.

In Fig 7 we see a screenshot of the app in practice. The user is provided with three buttons to provide feedback about these predictions. These ask whether the suggestions were good, bad, or okay and are translated into rewards of +1, -1, and 0 for our bandit. There is also an additional tab displaying plots which show how the bandit has been learning over time.

Screenshot of Poster Benchmarking App

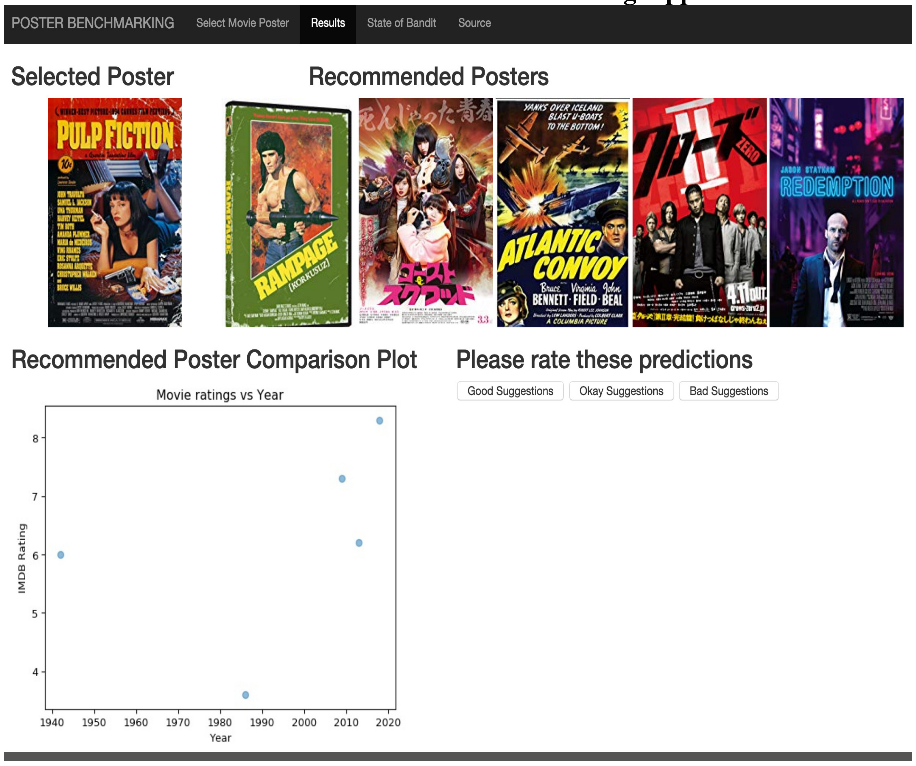

Fig. 7: Screenshot of poster recommendation application in practice on AWS.

VI. Results

Using the genres Action, Comedy, Drama, Horror, Romance, and Thriller we have obtained fairly reasonable results so far for the predictions out of the VGG-16 neural network. Fig. 8

shows the learning curves for the 6-genre dataset. Note that for a multi-label case such as this accuracy is a poor metric to utilize. Thus, to investigate how the performance of the model changes over time we have instead chosen to use the hamming distance. A well-performing model should show a decreasing hamming distance with time, which is what we see in our learning curve. The hamming distance tells us the number of bit positions in which the two bits disagree. The Hamming Distance is explicitly defined as:

$$HammingDistance = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} xor\left(y_{i,j}, z_{i,j}\right) \tag{4}$$

where y_i is the actual label vector and z_i is the predicted label vector for the movie at index i. j represents the index corresponding to a particular genre. Table 1 shows the classification report, including the precision, recall, and F1 score for each genre.

Learning Curves for 6-Genre Case

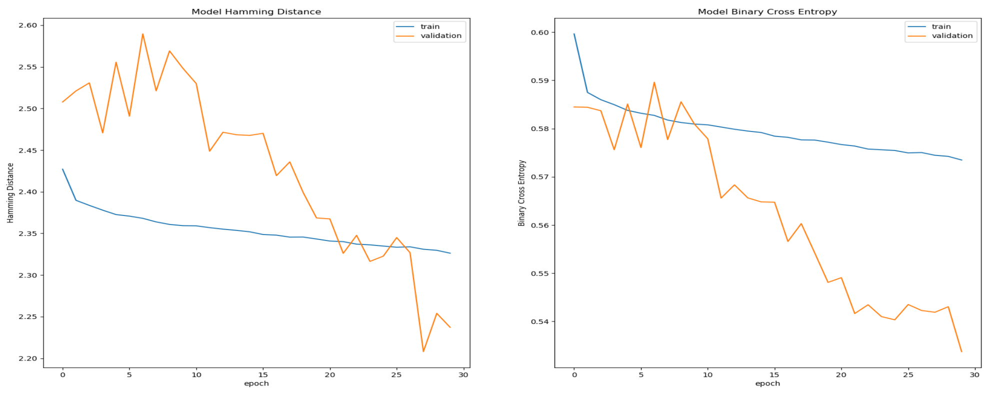

Fig. 8: Learning curves for the hamming distance and binary cross-entropy.

TABLE I: Precision, Recall, and F1-Score for Labels

	Precision	Recall	F1-Score
Action	0.39	0.24	0.30
Comedy	0.45	0.76	0.57
Drama	0.69	0.21	0.33
Horror	0.46	0.47	0.47
Romance	0.32	0.25	0.28
Thriller	0.43	0.04	0.08
Micro Avg	0.46	0.34	0.39

The precision, recall, and f1-score are presented for each label. The micro avg is included. This utilizes only the VGG-16 multi-label predictions.

TABLE II: Result With Same Architecture Through Graph-Based Model

	Precision	Recall	F1-Score
Action	0.37	0.56	0.45
Comedy	0.57	0.57	0.57
Drama	0.69	0.37	0.48
Horror	0.42	0.67	0.52
Romance	0.33	0.51	0.40
Thriller	0.31	0.36	0.33
Micro Avg	0.46	0.48	0.47

The precision, recall, and f1-score are presented for each label. The micro avg is are included. This uses the graph-based embedding and classification pipeline predictions.

The above results seem reasonable. Fig. 4 shows that at this moment there is still a significant amount of imbalance between movies tagged with each label. We believe that this is explains why Thriller is doing so poorly compared to the rest, even after passing through the graph-based embedding pipeline. The F1-score for most of the genres is significantly improved after passing through the pipeline. We believe these are extremely promising results.

As seen in Fig. 7, the poster recommendations coming from our model can sometimes make a lot of sense. In practice however, we have found that for some posters the recommendations make sense, but for many other posters it is quite difficult to understand what similarities are used for making the recommendations. Adjusting parameters has been shown to produce no significant improvements. We believe this is likely explainable as there is no agreed-upon standard for designing posters depending on which genres the movie adheres to. Thus, the spread of poster designs per genre labels can be expected to be quite large, thus limiting us in our approach.

As there is less in common between posters with similar genre labels we have begun exploring an alternative approach. We are currently training an autoencoder to reproduce the movie posters. This utilizes the same VGG-16 architecture described in Section III. However, we are exploring with adding different combinations of transpose-convolutional layers to reconstruct the image instead of utilizing dense layers after the last pooling layer. At this time the reconstruction is only able to produce very blurry representations, which are unacceptable for our use. Once this is working we will utilize the code layer for comparing posters, and feed this into the graph-based embedding pipeline for label predictions.

VII. Conclusion

We have implemented the entire pipeline from scraping of raw data through similar posters, including utilizing the graph-based embedding pipeline to provide improved predictions for genre labels. The app interface is easy to understand and use, and is currently hosted on AWS.

For a 6 label multi-label prediction problem a micro avg F1-Score of 0.47 is quite good. Movie posters tagged with the same genre labels can still vary significantly in poster design. To overcome this hurdle we have begun modifying the output of VGG-16 such that the model works as an autoencoder. This is still in progress, but as the autoencoder will have to learn all aspects of the poster in order to reproduce it, we believe that the comparison of movie posters should be much more robust, and lead to increased performance in label predictions after the graph-based embedding pipeline.

This project has made significant progress already, and found some very interesting and useful results. However, due to the relatively lackluster performance of the neural network the recommendations have not been sufficient to see the muluti-armed bandit determine a particular similarity metric as optimal. We believe that with the introduction of the autoencoder the recommendations will be of high enough quality that the similarity metric becomes highly important.

References

[1] X. Chen, P. Zhao, J. Xu, Z. Li, L. Zhao, Y. Liu, V. Sheng, and Z. Cui, "Exploiting visual contents in posters and still frames for movie recommendation," *IEEE*, vol. 6, pp. 68874–68881, 2018.

[2] M. ul Hassan, "Vgg16 – convolutional network for classification and detection," 2018.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), pp. 701–710, ACM, 2014.

[4] K.-H. Huang and H.-T. Lin, "Cost-sensitive label embedding for multi-label classification," *Machine Learning*, vol. 106, no. 9-10, pp. 1725–1746, 2017.