Genre Prediction using Deep Learning

Himani Garg, Mohit Jain, Sri Chaitanya Somanchi, Vijaya B Thangella

Harvard CS 109B

May 3rd 2017

ABSTRACT

Movie Genre[1] represents the grouping of motion pictures based on either the narrative elements or the emotional response. Needless to say, that it is one of the most important aspects of a movie/TV series and hence is widely discussed across the arenas of the internet. Since there isn't a science underneath the idea of genre identification, platforms like IMDb or Rotten Tomatoes, let users assign them based on their perception of the movie. We tried to apply principles of machine learning and formalize a scientific approach for genre identification.

KEYWORDS

Deep learning, Image processing, Genre Classification

1. INTRODUCTION

One of the biggest source of entertainment in the world that caters to most of the audience would be the movies. People have different likings and prefer to watch certain genre of movie more frequently than other. In this paper, we are presenting two approaches to predict movie genres - feature-based analysis and image based analysis. Feature based approach requires analyzing all available attributes of a movie and choosing relevant ones to run traditional models like SVM, RF to predict the genres from a movie. Image based approach involves using movie posters to predict movie genres. Deep learning models like Convolutional Neural Network (CNN) provide robust mechanism to classify high dimensional image data. A movie is often categorized to multiple genres based on user's perception of the movie thus making this to a multi-label classification problem. However, for the scope of this project, we have transformed this to a multi-class problem by encoding correlated genres to a new 'Custom Genre'.

2. FEATURE BASED APPROACH

In this approach, we explore attributes of a movie and determine genres of the movie

2.1 Data Extraction

Multiple potential sources were considered to retrieve data for the movies - IMDb, TMDb, Rotten Tomatoes, Wikipedia etc. However, we decided on using the data from TMDb and IMDb for the following reasons:

1. The scope of the project that we finalized required a certain specific set of attributes that were readily available from the two sources
2. While other sources – like Rotten Tomatoes had an API as well, the APIs provided by TMDb and IMDb were extraction efficient

Since movies are usually categorized into more than one genre we chose genres as varied as possible to avoid repetition of movies under different genres. Following are the genres that were considered: Horror, Comedy, Drama, Action, Mystery, Science Fiction, Romance, History.

TMDb provides a key (upon request) to access their data. While there are some wrapper APIs like TMDbsimple that are available, we used the core TMDb API to retrieve the data. We used TMDb data as the core set and augmented it with IMDb data. Both the sources have a restriction of 40 consecutive calls per second followed by a lag of 10 seconds[2]. Further, the movies were extracted from the year 2017 and going backwards each year until the total count per genre was achieved. The movies identified were of English language. TMDb returned the output in JSON format and it was transformed into pandas data frame for ease of performing data exploration.

2.2 Data Exploration

Data obtained from TMDb and IMDb was clean for the most part except that we observed certain columns were missing data. For example, the ratings of certain movies were missing or the poster path was missing. Such records were dropped because the amount of complete data available for other movies in both the repositories renders the imputation process redundant. To get a sense of the scale of the ratings of TMDb and IMDb, we picked a few movies and compared the ratings. The ratings from both the sources were comparable.

Further, we looked at the genres and the popularity of each of them and found that movies in genres – Action, Mystery and Thriller are most popular according to the data from both the sources (Refer to Figure 1). The data showed different categorizations of the movies from TMDb and IMDb. For example - movie "The Bourne Supremacy" has TMDb genres as Action, Drama, Thriller while IMDb genres are Action, Mystery, Thriller. Since we chose TMDb as the driver data source, we retained the genre labels from it. We started by extracting 1000 movies from each of the genres with and a total estimate of getting 8000 movies. However, we realized since same movie would feature under different genres our dataset reduced to just over 5600 unique movies. This dataset was used to train the traditional models.

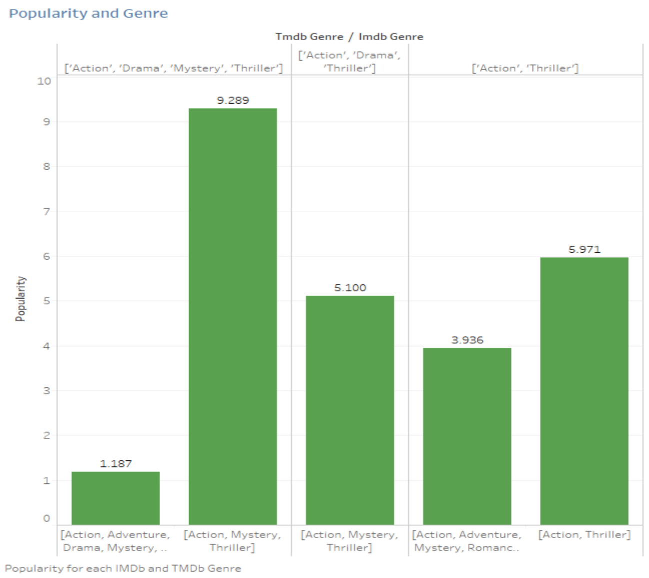

Figure 1: Genre Popularity

2.3 Feature Selection

Our initial approach was to include a lot of categorical features along with quantitative features. We planned to include 'Director', 'Writer', 'Cast' as categorical features because certain director directs movie of a specific genre most of the time, or a writer works on specific genre. But soon we realized that these categorical features cannot be used directly as the number of distinct values will be too many to convert them to quantitative variables (like one-hot encoding). Then we started searching other ways to add features and we have mainly explored two areas.

Bag of words: The data from TMDb and IMDb contained features like keywords, overview and plot. These features are a goldmine of essential information to perform text/sentiment analysis and could be used to determine the genre of the movie. For example – we observed that words like aliens, space occurred very frequently for movies with genre 'Science Fiction'. Thus, we performed text analysis on the keywords column from TMDb by eliminating stop words and calculating the frequency of occurrence for rest of the words. From the generated set of words, we picked top 40 bag of words and translated them to columns using one hot encoding.

Dominant colors in posters: One of the features we decided to use was dominant color of a poster using clustering. We got three attributes 'R', 'G', and 'B', to be included, but these three attributes independently were not useful. So, we converted these three features to 'brightness level' as 'High', 'Medium' & 'Low' based on Photometric/ digital ITU BT.709 standards[3].

After performing the above feature selection criteria, Table 1 shows final list of features to be included for Feature Based Approach

Features	Source
popularity	TMDB
release_year	TMDB
vote_average	TMDB
revenue	IMDB
runtime	IMDB
Rating	IMDB
Votes	IMDB
brightness_level	Posters
40 Bag of words	TMDb - keywords

Table 1: Selected Features

2.4 Class Generation

A particular movie can fall under multiple genres. This makes current problem as multi-label prediction. We have created a "Custom Genre" for each movie with combinations of most frequently occurring Genres. Using genre heat map (Figure 2), domain knowledge, and referring to Netflix[4] categorizations we have come up with "Custom Genres' as shown in Table 2.

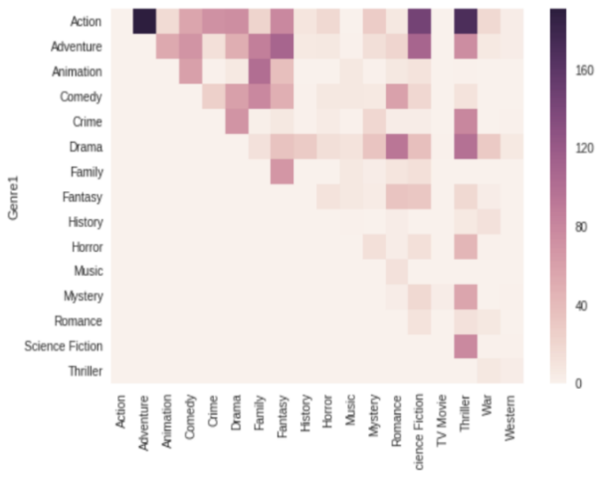

Figure 2: Genre heat map

We implemented a majority vote algorithm for labels to identify the class a movie falls into. Thus, a movie would get a 'Custom Genre' if it has most of the genres tagged for that class. In case of a tie, we chose one of 'Custom Genre' at random.

Custom Genre	Genre	Genre	Genre	Genre
AcAdDr	Action	adventure	drama	
CoFaDr	comedy	family	drama	
RoCoDa	Romance	Comedy	Drama	
AcThMi	Action	Thriller	mystery	
AcCrTh	Action	Crime	Thriller	
ScFa	Sci-Fi	Fantasy		
Ho	horror			
Ot	Music	War	History	Western

Table 2: Genre Class Mappings

2.5 Imbalanced Data

The problem of imbalanced data is usually common in realistic data science environments. Even in our current movie genre classification problem, the data is skewed towards certain genres which occur more frequently than others. Such scenarios would create a bias in predicting majority class. There are different ways of tackling this imbalanced problem. Common ways are under sampling the majority group, over sampling the minority group, adding class weights if the modeling library provides an option. Other techniques like SMOTE to overcome this imbalanced dataset by synthesizing the data for minority classes from the available minority group data.

While extracting the data, we ensured to have at least significant proportions for each of the genres. However, the ratio of highest class to lowest class is still around 25:1. We oversampled minority classes by a significant fold while avoiding even balancing. We did not under sample majority classes to prevent impact its impact on model performance. Instead of evenly balancing the classes we have used class weights along with oversampling while training the model. The class weights are assigned by giving more logarithmic weightage to the minority classes and less logarithmic weightage to majority classes. The log function is used to smooth out the weights (based on proportions) instead of assigning huge weights or very low weights

2.6 Traditional Models

There are multitude of options that are available to train a model to perform the classification on the data at hand. We ran multiple models – GLM Multinomial (GLM), Random Forest (RF), Gradient Boost (GBM), SVM Radial (SVM_RAD) and SVM Linear (SVM_LIN) to determine the genres of the movies. Before running the models our empirical guess was that Random Forest or Gradient Boost would perform the best among all the models because of the heuristics provided by the ensemble of underlying decision trees. Typically, SVM performs better for classification, but with the underlying problem of genre prediction it would not have been so successful because the genres are very close and overlapping.

Of all traditional models, Random Forest performed the best with an accuracy of 0.42 followed by Gradient Boost with an accuracy of 0.40. Figure 3 shows the accuracy of Random Forest over the course of grid search.

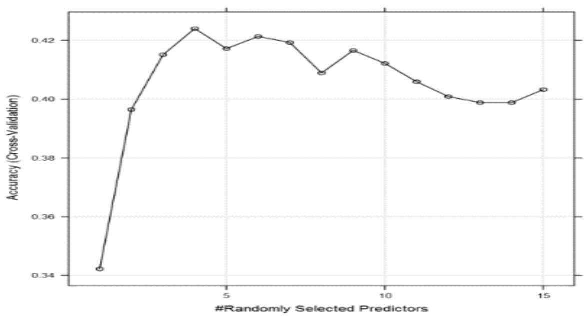

Figure 3: Performance of Random forest

Table 3 shows the performance comparison of different traditional models that were ran

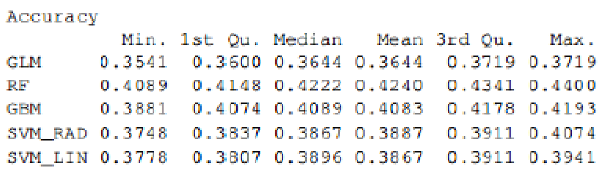
```
Accuracy
              Min. 1st Qu. Median    Mean 3rd Qu.    Max.
GLM      0.3541  0.3600 0.3644 0.3644  0.3719 0.3719
RF       0.4089  0.4148 0.4222 0.4240  0.4341 0.4400
GBM      0.3881  0.4074 0.4089 0.4083  0.4178 0.4193
SVM_RAD 0.3748  0.3837 0.3867 0.3887  0.3911 0.4074
SVM_LIN 0.3778  0.3807 0.3896 0.3867  0.3911 0.3941
```

Table 3: Performance comparison of traditional models

3. IMAGE BASED APPROACH (DEEP LEARNING)

In this approach, we use poster images available from TMDb and fed them to CNN and Pre-Trained Models and tune them to predict the genres

3.1 Data Extraction

3.1.1 Poster Extraction: All posters were downloaded from TMDb site using TMDbsimple package by concatenating base URL, size (w500) and poster path from TMDb data. Because the performance of Deep Learning increase with increase of train data, we downloaded up to 19,000 images. The poster download process has same download limitations as discussed in the feature extraction. We ignored the data for which poster URL is not available. In total, we have collected 18,566 unique images.

3.1.2. Resizing the posters data: The images we have downloaded were w500, that is the width of these images is fixed to 500 pixels. The height of image can vary based on the way the poster was created. Most of the collected images are 500 pixels width and 750 pixels height. If we standardize every poster to 500x750, we would end up having 375,000 pixels per layer (R, G, B layer) and 1,125,000 pixels to work with. With the number of predictors at hand, we have decided to reduce the number of pixels to 128 x 192 and this reduces the number of pixels to 73,728. Along with resizing, we have also eliminated images that are grayscale only. This leaves us with a dataset of 18,534 posters 128 pixels wide, 192 pixels high, and 3 layers of data.

3.1.3 Splitting data to Train and Test: We have used stratified shuffled split python library to split the data to train and test proportionately across classes. Once split, we have saved all 4 datasets (X_train, X_test, y_train, and y_test) to single numpy (np.savez) dataset, so this data set can be distributed between all members of the team.

Note on size of data: The size of 18,534 posters at the resolution of 128 x 192 pixels is close to 150MB. After translating images to numpy dataset with the process described above, the resultant size of dataset is close to 5GB. For this reason, we have used compressed functionality (np.savez_compressed) while saving dataset before distributing within the team.

3.2. Deep Learning Models

Deep Learning techniques like CNN, RNN provide a robust mechanism to train models for high dimensional data. CNNs are built to learn feature rich representations of the images and with multiple convolutions they can be trained to classify different types of images. In this section, we are going to build a CNN from scratch and compare it with a pre-trained model for image classification. We chose and tuned Inception -V3 as the pre-trained model in this process. The problem of imbalanced data was handled in a similar way as was done for traditional models.

3.2.1. CNN Model: A naive model was initially built with 2 convolution layers and 2 dense layers. We applied 16 filters of kernel size 5x5 with random weights set to default values. In other words, 16 convolution filters run through the image to obtain 16 unique features of the image. In general, every kernel will be assigned weight to highlight specific feature of the image. In this case, we configured the model to choose features at random. Max pooling is applied after the convolution layer to pick the maximum highlighted weight derived after convolution with the kernel. Max pooling helps eliminating pixels that highly correlated with the neighboring pixels. This way we retain the necessary features only pertaining to the corresponding filter.

Initially, we conducted a simple test on a sample of 100 images to over-fit the model on the training data by trying out different combinations of number of layers, filter sizes and number of epochs while validating the classification on test set. After obtaining a satisfactory outcome from sample test, we started training the model on all image data and obtained accuracy of 0.44. However, further analysis revealed that the model had classified all the images to a single majority class. Our next step was to design deeper and wider models to capture more feature information from the images.

In the initial model's two convolution layers, all the features were fully connected to 128 neurons in the first dense layer and it resulted in loss of features for movie image data. So, we added two more fully connected dense layers and the output of the last convolution layer was fed to 1024 filters in a fully connected network. While we could obtain the multi-class predictions the accuracy dropped to around 0.3. The process of tuning a model involves using different learning rate, batch sizes, number of epochs, number of filters etc. During tuning, we observed that a learning rate lower than 0.001 did not yield good accuracies. So, we chose the learning rate between a range of 0.001 and 0.1 with varying batch sizes. Finally, the learning rate of 0.01 resulted in a good accuracy and thus we proceeded with that value.

The next step in tuning this model was to initialize the weights to proper values. Since the input is an image with large number of features, a good initialization was to set the weights to a lower value and choose 'He' initialization. Further, batch normalization was added to keep all the weights normalized to standard values. This prevented the magnitude of data becoming too big or too small while weights and parameters adjust the data. Batch normalization also helped in faster learning and higher accuracy at the expense of the computation. These steps increased the accuracy to around 0.37. As part of fitting the model 20% of the train data was set aside for validation and the final accuracy was calculated on the test dataset. Since ours is a multi-class model we used 'Softmax' to determine the genre of the images.

While exploring other ways to improve this model, we tackled the imbalanced classes. By adding proper class weights and running against different batch size-learning rates. That yielded an accuracy of 0.40

Note on processing power: Since, deep learning is computational and memory intensive process, they require machines with high processing power which might not be available on personal machines. Since keras[8] library provides an option of utilizing GPU we used Amazon Web Services (AWS) p2.Xlarge EC2 instances for training and tuning the CNN models. As a benchmark, we ran one of the CNN models on a local desktop with NVIDIA GTX1080 graphics card and compared it with p2.Xlarge instance. The processing time on p2.Xlarge 27 folds faster compared to local desktop.

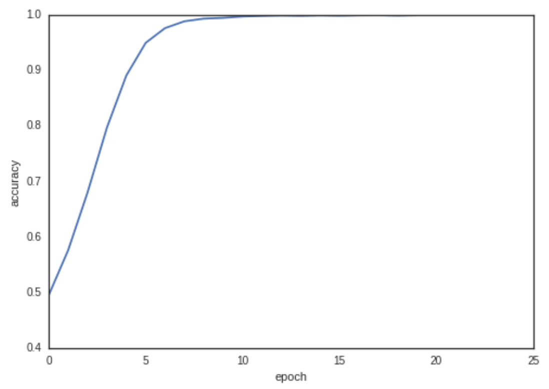

Figure 4: Scratch Model - Accuracy variation per epoch

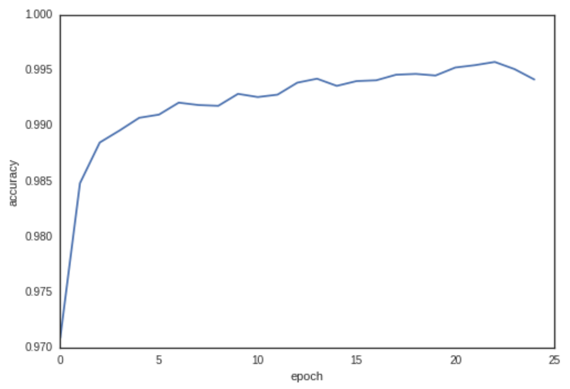

Figure 5: Pre-Trained Model-Accuracy variation per epoch

3.2.2: Pre-Trained Model: The pre-trained deep learning model that we used was Inception-v3[5]. It was developed by Google and was trained for the ImageNet Competition using the data from 2012. We chose this model because of its high classification performance and ease of availability in TensorFlow. The Inception architecture of GoogLeNet was also designed to perform well even under strict constraints on memory and computational budget.

We added fully connected layers and a global spatial average pooling layer because it is a structural regularizer. It natively prevents overfitting for the overall structure which allowed us to have the input image of any size and not just a fixed size because it averages every incoming feature map. For example, with a 192x128x3 incoming tensor of feature maps, it takes an average of 15x15 matrix slice, resulting in an 8-dimensional vector. This vector was then fed into fully connected layers.

This model was trained on few epochs and after the top layers were well trained, we fine-tuned the convolutional layers from Inception-v3. We chose to train the top 2 inception blocks by freezing the first 172 layers and unfreezing the rest. The whole model had to be recompiled for the modifications to take effect and the model was re-trained using a low learning rate to fine tune the inception blocks.

After doing further parameter tuning like trying the various combination of batch size (512-800), lower learning rate (0.01, 0.001, 0.0001) ConvNet weights were relatively good and we decided not to distort them. The best combination turned out to be a batch size of 512 and a learning rate of 0.0001. The accuracy reported was: 0.44. After fine tuning the pre-trained model, accuracy increased by 4% compare to scratch model.

3.2.3: Regular CNN vs Pre-Trained Model: Pre-trained models are the models that have been trained on a large amount of data over a long period of time and thus the convolutional layers of the model hold the essential information. Since they are usually large models their weights are often useful even for cases where the actual data is different from the data pre-trained models used. Depending upon the volume of data available for training it is determined if using a pre-trained model is a good idea or not. Generally, the application of pre-trained models fall into four broad categories:

1. If the data at hand is small but similar to pre-trained data. It is not advisable to use pre-trained models because of overfitting.
2. If the data volume is huge and the data is similar to the pre-trained data, it is very useful to apply pre-trained models
3. If the dataset is small and different than the pre-trained data. In this case, it is not advisable to apply pre-trained models
4. Data volume is high but the data is very different than the pre-trained data. In this case, initializing the weights from the pre-trained models is helpful.

Model Comparision	Time/epoch	Test Loss	Test Accuracy
Scratch Model	41 sec	3.74	0.40
Pre-Trained Model	87 sec	2.08	0.44

Table 4: Performance comparison of CNN models

We used InceptionV3 and we fall into fourth category. Since we utilized the convolutional layers from this model and added fully connected layers on top of it, our accuracy on the test set came out to be 0.44 while achieving an accuracy of 0.98 on the train set when we ran it for 25 epochs with a batch size of 512. This is a better accuracy than the accuracy we achieved by training our own model.

3.2.4. Additional Exploration: Data pre-processing and data augmentation: All of the above methods deal with providing images as arrays to the input layer of CNN. Another method to feed data to models is using data pre-processing and data augmentation[6]. This method has two advantages, the number of images we are working with can be increased without collecting more data (helps in the cases where the number of images is less), and the images are fed to models in batches.

Thus, the execution of training multiple layers will be fast. We have seen a significant reduction in execution time of each epoch. Holding every other factor (hardware, data samples etc.) constant we observed a decrease in execution time from 42 sec/epoch to 1 sec/epoch.

We used ImageDataGenerator from keras library. The data preparation required for this package is different as compared to traditional methods. We need to place train/validation/test data in separate folders and within each of these folders class specific folders need to be created. This way we directly work with images and need not specify classes for each image (‘y’) specifically. The number of classes and data per class are automatically inferred from folder structures.

Using stratified split, we distributed the images proportionately per class between ‘train’, ‘validation’, and ‘test’ datasets. The classes are converted from names to numeric starting with 0 to 7 (alphabetically) and thus class folders are named 0 through 7.

Image Generators: We augmented images on train data using the parameters shown in Equation 1. While validation and test data were just scaled. Once augmentation parameters were set, we created three generators, one for each train, test and validation sets.

```
train_datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

Equation 1: Image Augmentation parameters

Figure 6 is a representation of a single image augmented 20 different ways using above parameters.

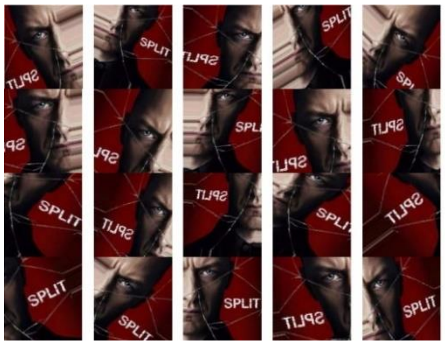

Figure 6: Image Augmentation

Prediction and performance: Like the naïve CNN model, an untuned model on augmented data predicted the majority class for all the images with an accuracy of 0.44. However, after applying similar fine tuning parameters as done on scratch CNN model we obtained an accuracy of 0.42.

Advantage: The biggest advantage with this approach is that execution time taken for each epoch was less than a sec on AWS p2.Xlarge instance, whereas the traditional model took 42 sec/epoch.

4. CONCLUSION

Assigning genre to a movie is subjective rather than objective[7]. It is driven by user’s perception of a movie and hence is not as straightforward as identifying an image of a cat as ‘cat’. Since a movie can be categorized under multiple genres, identifying relevant genres out of over 20 categories could be a daunting task. While applying deep learning algorithms like CNN and using just a poster image, we were able to achieve an accuracy of up to 0.44. Though traditional models gave comparable accuracy, the data preparation process is tedious because of data collection, exploration, and feature selection. The data preparation process is not extendable to other kind of videos (Music, TV Shows, etc.) due to difference in features.

A Deep Learning Approach, like the one implemented by us, could be used by movie review websites (like TMDb and IMDb) to provide better suggestions to users for choosing relevant genres. It could also be used by services like Netflix or Amazon Prime Video, to identify genres of different movies. Since this approach uses only image as predictor, the solution can be extended to any kind of videos.

5. FUTURE WORK

We plan to expand our work to following avenues:

- Implement multi-label classification
- Collect more posters and genres
- Classify using multiple pre-trained models like VGG16
- Tune hyperparameters over a wider range of values

6. REFERENCES

1. https://en.wikipedia.org/wiki/Film_genre
2. https://www.themoviedb.org/faq/api
3. https://en.wikipedia.org/wiki/Rec._709
4. www.netflix.com
5. https://pdfs.semanticscholar.org/0626/908dd710b91aece1a81f4ca0635f23fc47f3.pdf
6. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
7. https://contribute.imdb.com/updates/guide/genres
8. https://keras.io/getting-started/faq/