CS109B Final Project

Milestone 4

Group 26

Dominick Delucia , Nina Iftikhar, Nisreen Shiban and Hany Bassily

Deep Learning Classifier for Movie Genres

Purpose:

Main objective of the Milestone is to build a deep learning classifier for movie genres. This part details the Convolutional net architecture adapted for this purpose

Data preprocessing:

Al poster images (9988) were transformed into numpy tensor with RGB decomposition. The images were cropped and rescaled with same aspect ratio to a reduced size of 85 x 128

Reduced tensors were transferred to an AWS EBS storage available to be mounted at any instance initiation. For this purpose, a private instance was implemented and shared among the group

Selected Architectures

All architectures were derived from the main architecture introduced in the lab and applied with the MINST dataset. Additional layer and rearrangement were done after few initial trials with runs that did not exceed 20 epochs which led to the first version

1- **Version 1:**

Data preprocessing:

- Channel rearrangement
- 14 labels multi label classification
- 20% validation
- Centered features

Main Architecture

- Multi layer CNN :
  - Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization
  - MaxPool 2x2

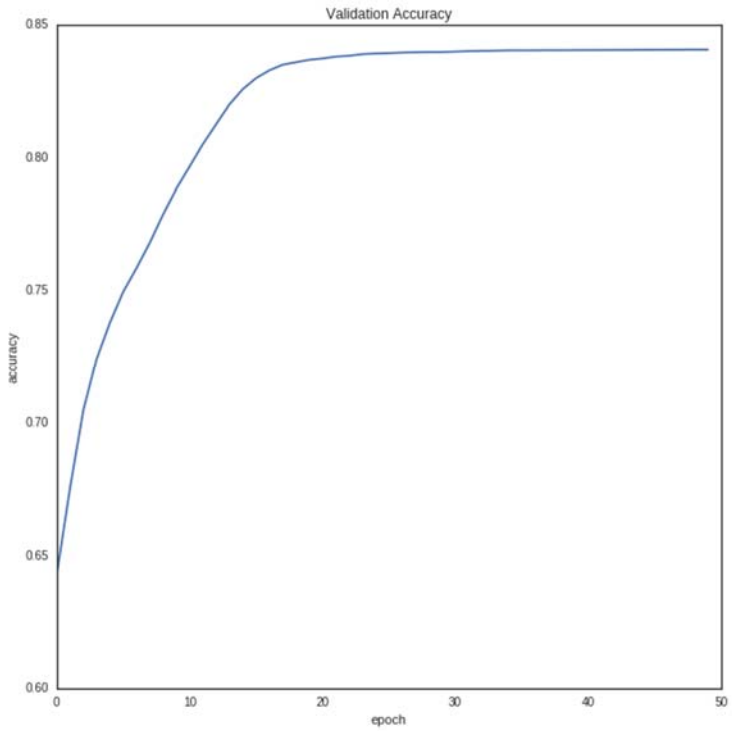
- Conv2D - Relu depth 16 and 3x3 kernel - He uniform initialization
- MaxPool 2x2
- Conv2D - Relu depth 16 and 3x3 kernel - He uniform initialization
- FC 128 , Relu , He uniform initialization
- FC 64 , Relu , He uniform initialization

- SGD optimzer with 1e-6 learning rate and 0.99 momentum
- Binary cross entropy loss function
- Batch size 64
- Training convergence after 50 epochs

Main Features:

- Combination of batch size and learning rate proves to give reasonable training behavior
- Initialization introduced for consistency in training
- Loss function was selected to be Binary cross entropy to fit th multilabel classification problem

Performance:



2- Version 2:

Data preprocessing:

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

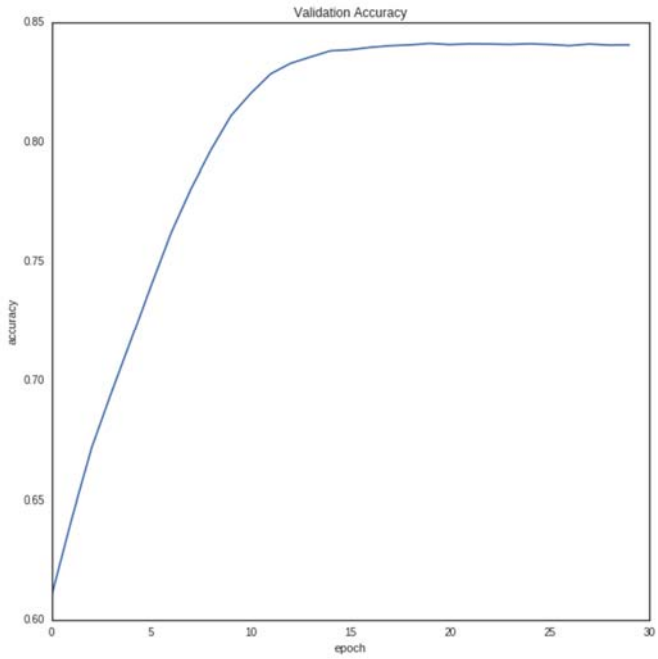
Main Architecture

- Multi layer CNN :
  - Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
  - FC 128 , Relu , He uniform initialization
  - FC 64 , Relu , He uniform initialization
- SGD optimzer with 1e-6 learning rate and 0.99 momentum
- Binary cross entropy loss function
- Batch size 64
- Training convergence after 30 epochs

Main Features:

- Data augmentation was introduced to give more flexibility in increasing the complexity of the model without high risk of overfitting. Results were more depth in the conv layers

Performance:



3- Version 3:

Data preprocessing:

- Channel rearrangement
- 14 labels multi label classification
- Data augmentation by shift and zoom for 8000 training sample and 2000 test sample
- Centered features

Main Architecture

- Multi layer CNN :
  - Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
  - FC 128 , Relu , He uniform initialization
  - FC 64 , Relu , He uniform initialization
- Adam optimzer with 1e-6 learning rate
- Binary cross entropy loss function
- Batch size 64

- Training convergence after 30 epochs

Main Features:

- This version is to test the Adam optimization which is clearly did not contribute a lot
- Decision point: Current architecture with the SGD optimization is basis for any other variation

Performance:

- Figure not available. Decay is not consistent

4- Version 4:

Data preprocessing:

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

Main Architecture

- Multi layer CNN with Batch Normalization Layers :

  - Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization
  - Batch Norm
  - MaxPool 2x2
  - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization
  - Batch Norm
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
  - Batch Norm
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
  - Batch Norm
  - FC 128 , Relu , He uniform initialization
  - Batch Norm
  - FC 64 , Relu , He uniform initialization
  - Batch Norm

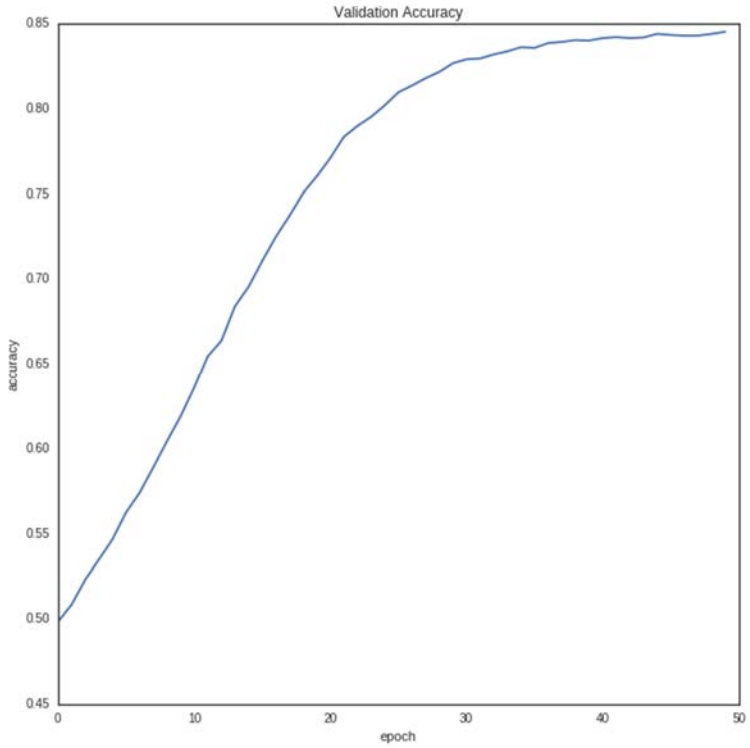
- SGD optimzer with 1e-4 learning rate and 0.99 momentum
- Binary cross entropy loss function
- Batch size 128

- Training convergence after 50 epochs

Main Features:

- Batch normalization was introduced after each ReLu activated layer
- Batch increased to improve statistical inference

Performance:



5- Version 5:

Data preprocessing:

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

Main Architecture

- Multi layer CNN with Batch Normalization Layers and Dropout :

  - Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization

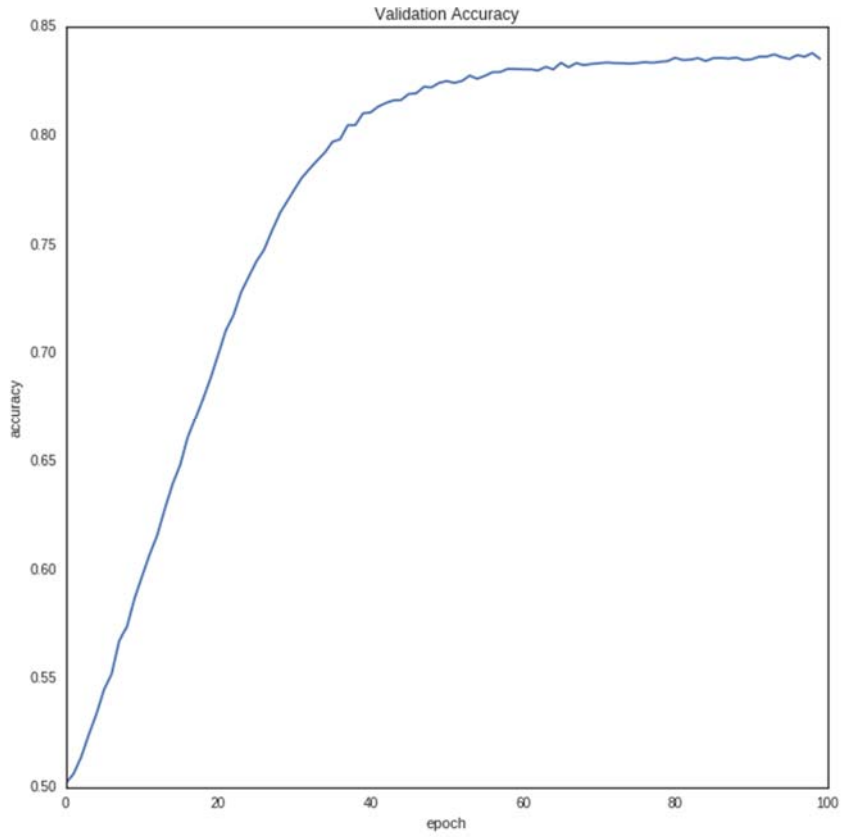
- 30 % Dropout
- Batch Norm
- MaxPool 2x2
- Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization
- 30 % Dropout
- Batch Norm
- MaxPool 2x2
- Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
- 30 % Dropout
- Batch Norm
- MaxPool 2x2
- Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization
- 30 % Dropout
- Batch Norm
- FC 128 , Relu , He uniform initialization
- 50 % Dropout
- Batch Norm
- FC 64 , Relu , He uniform initialization
- 50 % Dropout
- Batch Norm

- SGD optimzer with 1e-4 learning rate and 0.99 momentum
- Binary cross entropy loss function
- Batch size 128
- Training convergence after 100 epochs

Main Features:

- Aggressive dropout ratios were chosen to test the effect – no change in accuracy
- Training was extended to 100 epochs to account for the drop out

Performance:



6- Version 6:

Data preprocessing:

- Channel rearrangement
- 14 labels multi label classification
- data augmentation by shift and zoom for 8000 training samples and 2000 test samples
- Centered features

Main Architecture

- Multi layer CNN with Batch Normalization Layers and Dropout and 0.1 L2 Kernel Regularization

  - Conv2D - Relu depth 32 and 7x7 kernel - He uniform initialization - 0.1 L2 Kernel Regularization
  - 30 % Dropout
  - Batch Norm
  - MaxPool 2x2
  - Conv2D - Relu depth 32 and 5x5 kernel - He uniform initialization - 0.1 L2 Kernel Regularization
  - 30 % Dropout
  - Batch Norm
  - MaxPool 2x2
  - Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - 0.1 L2 Kernel Regularization

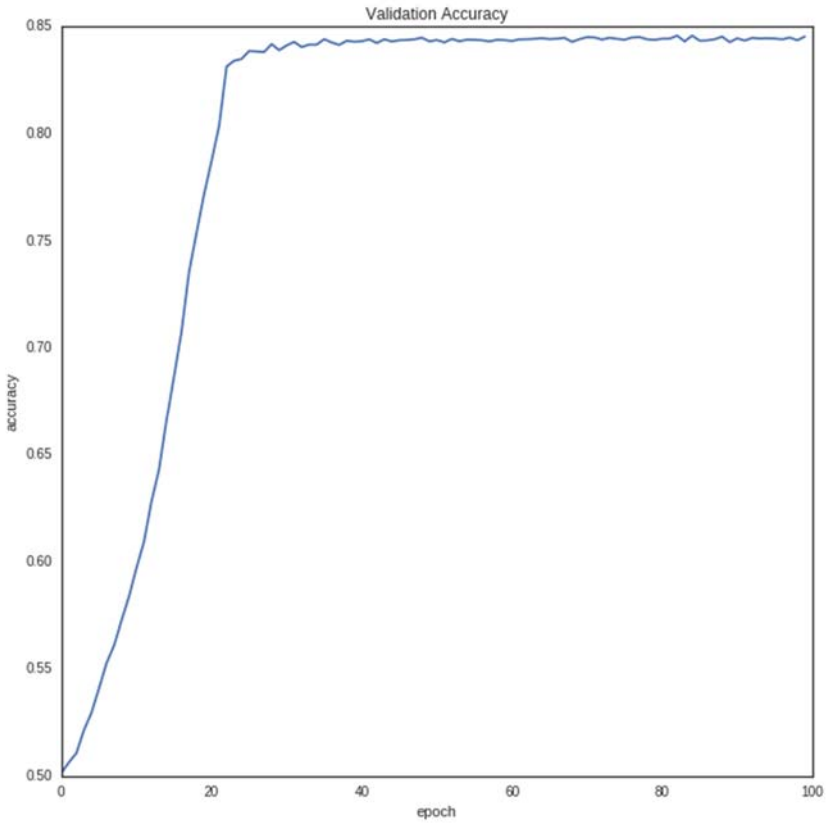
- 30 % Dropout
- Batch Norm
- MaxPool 2x2
- Conv2D - Relu depth 64 and 3x3 kernel - He uniform initialization - 0.1 L2 Kernel Regularization
- 30 % Dropout
- Batch Norm
- FC 128 , Relu , He uniform initialization - 0.1 L2 Kernel Regularization
- 50 % Dropout
- Batch Norm
- FC 64 , Relu , He uniform initialization - 0.1 L2 Kernel Regularization
- 50 % Dropout
- Batch Norm

- SGD optimzer with 1e-4 learning rate and 0.99 momentum
- Binary cross entropy loss function
- Batch size 128
- Training convergence after 100 epochs

Main Features:

- Two values of the L2 regularizer were selected 0.01 and 0.1 and no difference in the overall accuracy was noted
- Similar training approach was applied with faster training convergence as an effect of restricting the weights freedom to adapt during the backpropagation adaptation
- Slower training rate could be an improvement for this version
- This is the main objective of a cross validation effort that will be subsequently implemented

Performance:



Final Conclusion:

- No improvement was observed when trying to adapt all the previous versions and the accuracy remained at about 84% on the validation/test set
- This could be an artifact of the data
- Some improvement to be considered is to merge other features with the image features, perform cross validation and try other architecture less deeper but different initializations

Discussion of the results, how much improvement you gained with fine tuning, etc.

(Note: This discussion is also included in the RGB_MLP.ipynb file, where it can be read in context).

In **model** we only utilize the color data (RGB means and sd for posters). Here, we utilized the Sequential model in Keras in order to create an MLP model for multi-level softmax classification. Instead of using the "relu" activation parameter, we chose to utilize the "sigmoid" activation because it was better across the board. We experimented with the learning rate by hand to examine the process along the epochs. Experimenting with (1e-5,1e-4,1e-3,.01,.1) we concluded that .01 was the best learning rate. We decided on the .01 rate by looking at the training data even though when evaluated with the test data the total accuracy was roughly the same. We also experimented with some momentum values (.5 - .99), but .9 seemed to be the best. They were often rougly the same, but other values seemed to have more volatile output accuracies. So, .9 seemed to be a pretty typical value that was reliable.

Overall, the accuracy for the color vectors is not very good. On the test set it comes out to about 20% accuracy. However, this is reasonable because our color analysis is extremely elementary. We scraped the mean value for each of the three RGB vectors and also recorded its standard deviation. You can imgaine many scenarios where this kind of analysis fails to pick up differences between genres or the character of the movie, but it may have some predictive power when added to a larger deep learning model for the posters themselves.

In **model_full** we utilize all meta_data scraped from movies. This includes all color data as well as budget, release year, popularity score, revenue, and runtime.

Currently getting odd behavior with a lack of loss calculation... Unsure what is causing this. And it always ends up with the same final train and test accuracies no matter the hyperparameter settings.

Although model_full has higher test accuracy, I am in favor of using the first model because this behavior is odd and we cannot seem to tune hyperparameters...

Complete description of the pre-trained network that you fine tuned, including parameter settings, performance, features learned, etc.

As one of our models, we wanted to use the Keras VGGNet, which provides the a design for the network structure. VGG adopts the simplest architecture by using only 3x3 convolution and 2x2 pooling throughout the entire network. The design also shows that the depth of the network plays a significant role. While deeper networks give better results, we chose the VGG16 model rather than the VGG19, both of which are available through Keras. This method was developed by Oxford's Visual Geometry Group (VGG), which achieved very good performance on the ImageNet dataset. It provides the definition of each layer along with pre-trained set of weights. In other words, the network already has learned features which come in handy for most computer vision problems, and leveraging such features would allow us to reach a better accuracy than any other method using only the available data. One setback of

VGGNet is that this network usually provides about 160M parameters, which are often consumed in the fully connected layers.

Unfortunately, we encountered problems when we tried to run the model, so we will present our findings in the next milestone. After we opened poster files and transferred them to fixed-dimension thumbnails, we realized that we needed to turn the thumbnails to those with the default input size of 224x224x3 rather than our 128x85x3, which was acceptable for our CNN model. This realization came early enough for us to fix the size of the thumbnail, but we made the mistake of not adjusting the size of the poster to 500x500 from the 500x700 nominal, which was also used for the CNN model. Due to issues with storage and leaky instances, this could not be fixed in time, and we were not able to ultimately run the model. This small mistake will, however, be quickly corrected in time for the final project. For this milestone, we will talk about how we plan to run this model by the time of our final submission.

Since our output is multinomial, we considered two options for how we can train our model. One option is to train a multi-class image classifier in Keras, and the other option is to train 14 different models, one for each movie genre. I will focus on the former and talk about how we plan to achieve both efficiency as well as high validation accuracy.

Our strategy will be to instantiate the convolutional part of the model using the fully-connected layers only. This will be run on our training and validation data once, recording the output from the VGG16 model in two numpy arrays, followed by training a small fully-connected model on top of the stored features. For computational efficiency, we will not add the fully connected model directly on top of a frozen convolutional base then run the entire model but rather we will store the features first. This is because VGG16 is computationally expensive, as our group has painfully realized, so we will try as best as we can to to only do it once (which unfortunately prevents data augmentation like adding noise or applying transformations).

Along with the top-level classifier, we plan to fine-tune the last convolutional block. Broadly speaking, we start from a trained network and retrain it on a new dataset using small weight updates. To achieve this, we will instantiate the convolutional base of VGG16 and load its weights, then add our fully-connected model on top and load its weights, and finally, we will freeze the layers of the VGG16 model up to the last convolutional block (we will make sure to properly train the weights and train the top-level classifier starting the fine-tuning of the convolutional weights alongside it). If the weights are not properly trained, the large gradient updates due to the randomly initialized weights would wreck the learned weights in the convolutional base.

Since the entire network would have a very large entropic capacity and great tendency to overfit, we would encounter overfitting if we do not only fine-tune the last convolutional block. Additionally, it would make sense to finetune the last convolutional block since the feature learned by low-level convolutional blocks are more general and should remain fixed. Finally, we

can avoid ruining previously learned features at each update by keeping our learning rate very slow which keeps the magnitude of our updates very small. Therefore, we should avoid any adaptive learning rate optimizers and just stick with the SGD optimizer.

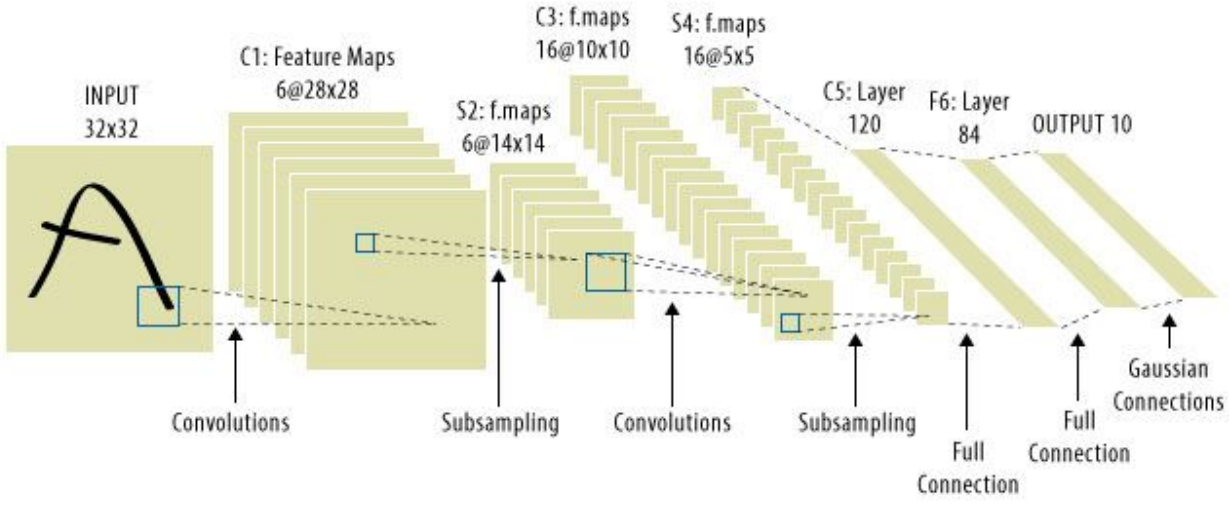
We will likely try 50 epochs or more. We may ultimately need to do data augmentation if our validation accuracy is low. Additionally we may need to use L1 and L2 regularization and perhaps even fine-tuning one more convolutional block.

Discussion of at least one additional exploratory idea you pursued

A multi-layer perceptron model (MLP) was pursued as an additional exploratory idea for this study since MLPs can be good for data that is not linearly separable; the goal was to construct performance evaluations for relatively similar architectures and parameters for the MLP models to compare with those from the CNN models.

In contrast to models with CNN architecture, MLP models are harder to train at scale because of a lack of translation-equivariance- for images this means that if there's a signal in one part of the image that the MLP needs to be sensitive to, it would need to relearn its capacity for that signal's sensitivity it if that signal moves; this wastes the capacity of the neural net, thus making the model more difficult to train[2].

CNN models, meanwhile, use convolution, and pooling, for each input signal by utilizing a kernel, and thus are sensitive to the same feature in all possible locations[2].



This means that the CNN models are expected to perform better relative to the MLP models in terms of model accuracy.

An MLP, in its turn, is a feedforward (data flows in one direction from input to output layer (i.e., forward)) neural network with one or more hidden layers between the input and the output layers. An MLP is trained with the backpropagation learning algorithm[1].

For the MLP model in this study, image data of 9988 movie posters was pre-processed into numerical tensor values for the images, and the shape of the Genre Labels matrix was used to define the number of classes in the data set. The Genre Labels matrix hold a 0-1 value in binary format for each movie where applicable in the appropriate genre field.

The following table describes the architecture and training parameters of the MLP 1 model; the activation function ReLu was used.

Table 1. MLP 1 is composed of the following architecture and training parameters:

Architecture	Training
- 3 hidden layers with one encoding layer, and 19 inputs - Loss function: Binary cross-entropy - Validation size: 250 - Dropout fraction - tunable - Regularization factor (L1/L2) - tunable	- Epochs: 20 - Learning rate: 0.01 - Batch size: 64 - SGD momentum 0.9 - Using Adam

Issues with Amazon Web Services prevented running the model, and thus collecting reportable data. The cause of this issue has been identified, and will not prevent running this, or other models, for further study; specifically, it was found that a volume with the data was not mounted to the AWS instance for running the model.

Three hidden layers were chosen to start; for further study, different MLP model variations will be tried. Both the dropout and regularization factors will be tuned in the final MLP 1 model.

Different optimizers will be tried for further studies as well, but the Adam optimizer was used with the SGD method and momentum 0.9 to start to produce faster convergence[4]; the goal for this initial trial was to run a relatively fast model. Towards that end, it can be noted that the SGD optimization algorithm, which will be tried in a future study, will progress faster if it utilizes larger mini-batches; this will be configured rigorously for the future MLP models[3].

Lastly, this MLP model attempted to utilize real-time data augmentation via the ImageDataGenerator function from Keras in order to increase validation accuracy.

References:

1:http://scikit-learn.org/stable/modules/neural_networks_supervised.html

2:https://stats.stackexchange.com/questions/234891/difference-between-convolution-neural-network-and-deep-learning

3:https://www.quora.com/Intuitively-how-does-mini-batch-size-affect-the-performance-of-stochastic-gradient-descent

4:https://stats.stackexchange.com/questions/220494/how-does-the-adam-method-of-stochastic-gradient-descent-work