Extracting shot complexity from movie thumbnail data

Ryan Amundson

Northwestern University
Edward Arroyo

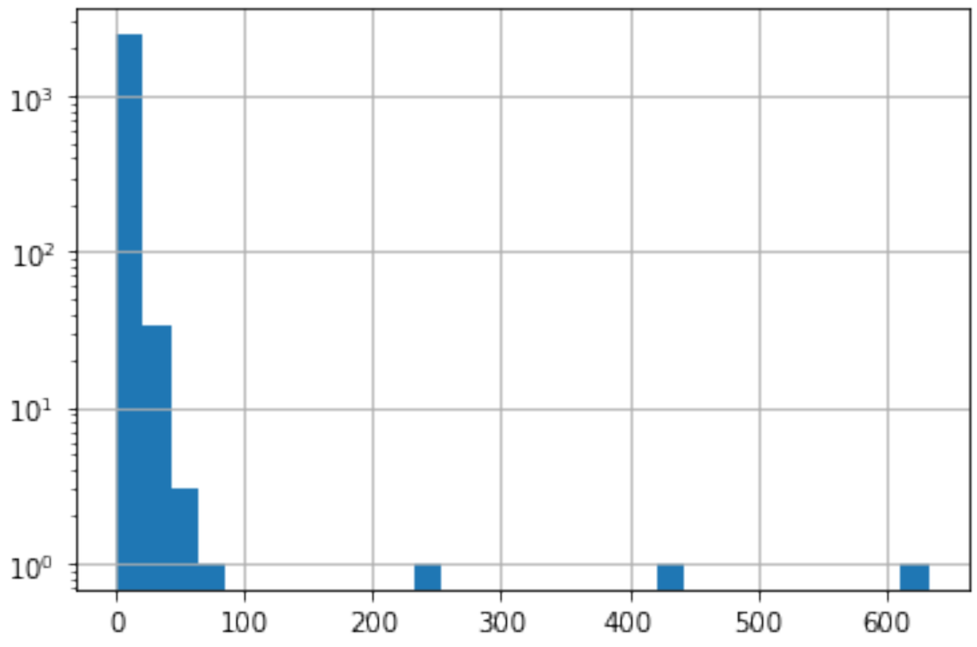
June 4, 2022

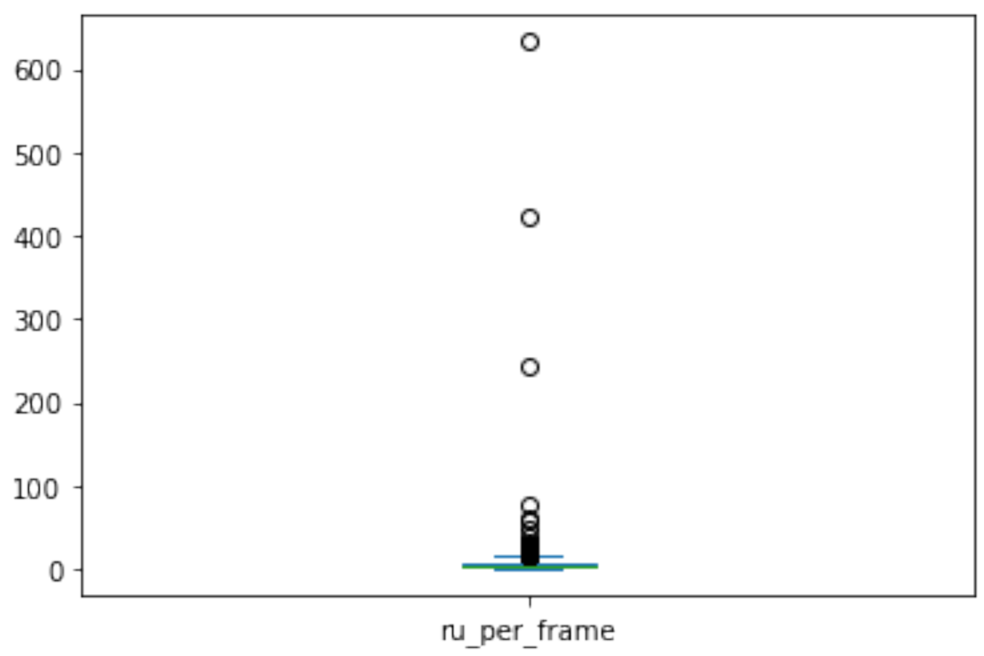
Abstract

For this assignment, I generated the dataset from our DreamWorks Animation farm data. The data consists of a metric called RU per frame. The RU is a render unit, and it is a normalized core hour divided by the number of frames in a shot, which gives us an estimate as to how expensive that particular shot is to render. I also pulled representative images of each shot from our thumbnail service. Using this image data, I am hoping to be able to train a network to be able to get close to a realistic regression estimate for the RU for unseen shots. This potentially could help improve our forecasting models for costing, as the render resources are some of the most expensive costs when making an animated film.

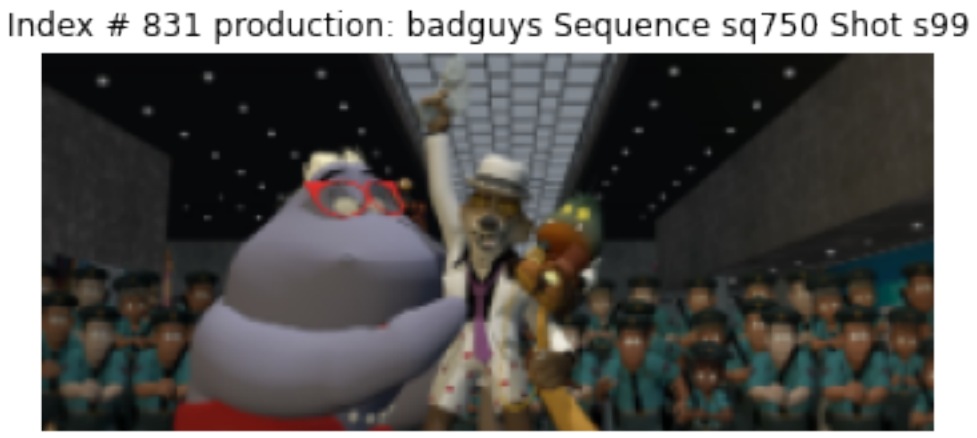
Introduction

For this project, I took a sample of 2,500 production shots from three completed films from DreamWorks, The Bad Guys (https://www.imdb.com/title/tt8115900/), Trolls 2: World Tour (https://www.imdb.com/title/tt6587640/) and The Boss Baby 2: Family Business (https://www.imdb.com/title/tt6932874). I matched each shot to a thumbnail in our thumbnail service, so each calculated RU per frame value has a corresponding image for that sequence and shot. In my data pull, the target data of RU per frame had significant right skew (with a skew value of 29.27) and some very clear outliers:





The largest outlier was from The Bad Guys, sequence 750, shot 99.



It makes somewhat sense to me that this would be a really costly shot for numerous reasons, first, because it would have been rendered multiple times since it was in the trailer, which is rendered well before the movie is finalized (You can see the shot 52 seconds into the trailer):

Secondly, this is a crowds heavy shot, with multiple characters in the background. Third, it would have both characters effects with the dress animation, and FX elements with the grappling gun and cord. All of which really probably adds up to a very expensive shot for that production.

Methods

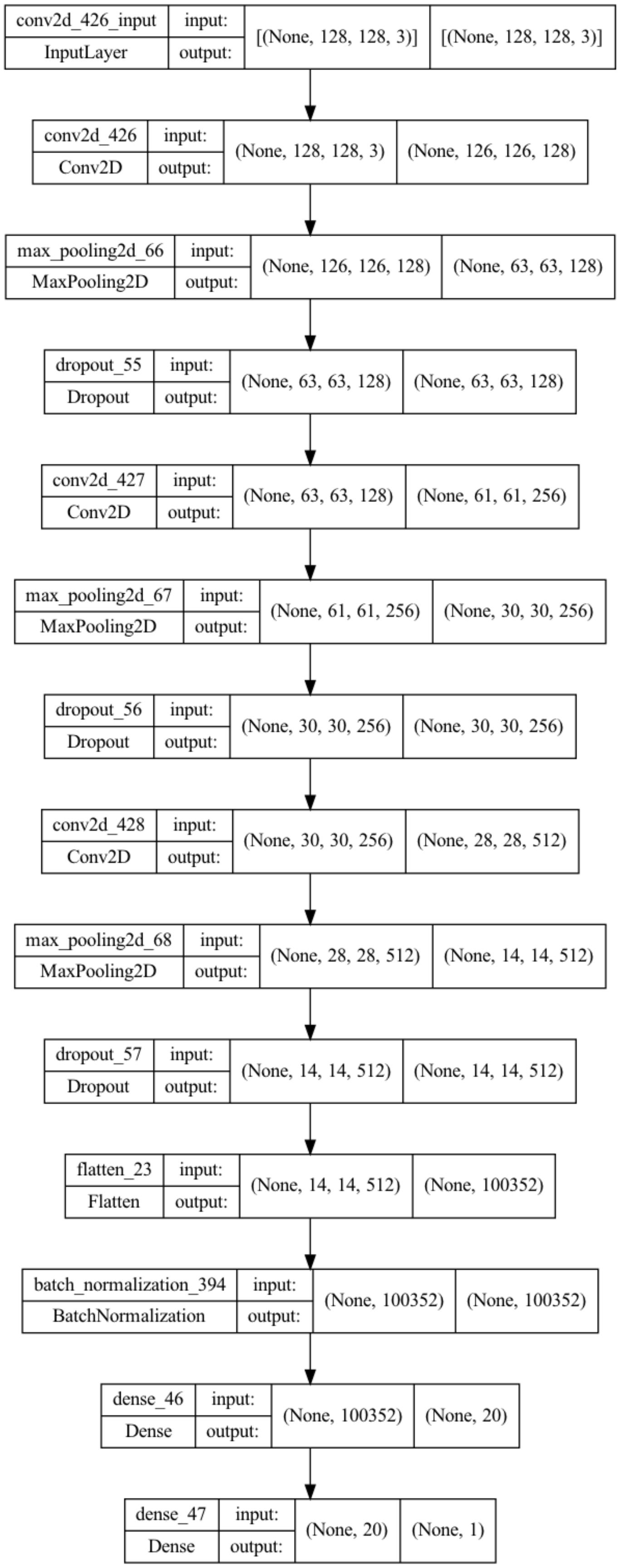
For each model, I am calculating the RMSE (root mean squared error), MAE (mean average error) and R squared value, along with a scatter plot comparing the actual value with the predicted. I am also attempting to use some pre-trained weights and the Resnet 50 (https://iq.opengenus.org/

resnet50-architecture/) and Inception Version 3 (https://arxiv.org/pdf/1512.00567v3.pdf) architectures, which are available in the Keras applications module (https://www.tensorflow.org/api_docs/python/tf/keras/applications). Because of the heavy right skew and some large outliers in the data that I pulled, so I imagine I would need to do significantly more cleanup on the data to get better results.

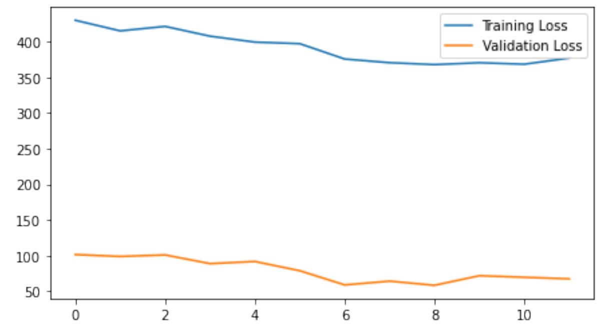
I reduced the images for each shot to 128 pixels in width to allow the networks to be trained in a reasonable amount of time, but kept the 3 color channels per image.

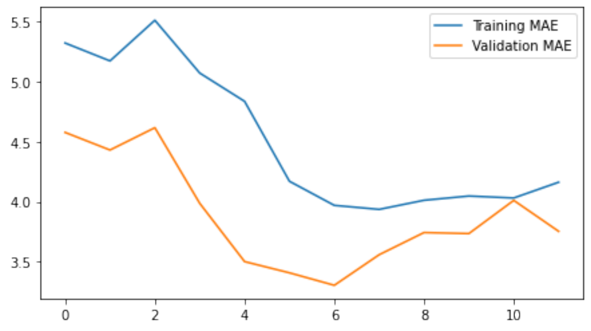
Results

The first model I used was a CNN with three filter layers and a single output for regression. This is the CNN structure:


conv2d_426_input | InputLayer | input: | output: | [(None, 128, 128, 3)] | [(None, 128, 128, 3)]
conv2d_426 | Conv2D | input: | output: | (None, 128, 128, 3) | (None, 126, 126, 128)
max_pooling2d_66 | MaxPooling2D | input: | output: | (None, 126, 126, 128) | (None, 63, 63, 128)
dropout_55 | Dropout | input: | output: | (None, 63, 63, 128) | (None, 63, 63, 128)
conv2d_427 | Conv2D | input: | output: | (None, 63, 63, 128) | (None, 61, 61, 256)
max_pooling2d_67 | MaxPooling2D | input: | output: | (None, 61, 61, 256) | (None, 30, 30, 256)
dropout_56 | Dropout | input: | output: | (None, 30, 30, 256) | (None, 30, 30, 256)
conv2d_428 | Conv2D | input: | output: | (None, 30, 30, 256) | (None, 28, 28, 512)
max_pooling2d_68 | MaxPooling2D | input: | output: | (None, 28, 28, 512) | (None, 14, 14, 512)
dropout_57 | Dropout | input: | output: | (None, 14, 14, 512) | (None, 14, 14, 512)
flatten_23 | Flatten | input: | output: | (None, 14, 14, 512) | (None, 100352)
batch_normalization_394 | BatchNormalization | input: | output: | (None, 100352) | (None, 100352)
dense_46 | Dense | input: | output: | (None, 100352) | (None, 20)
dense_47 | Dense | input: | output: | (None, 20) | (None, 1)

The loss function I used for all the models is mean squared error. Here is the graph of the training curves for the validation and training sets for the first CNN:



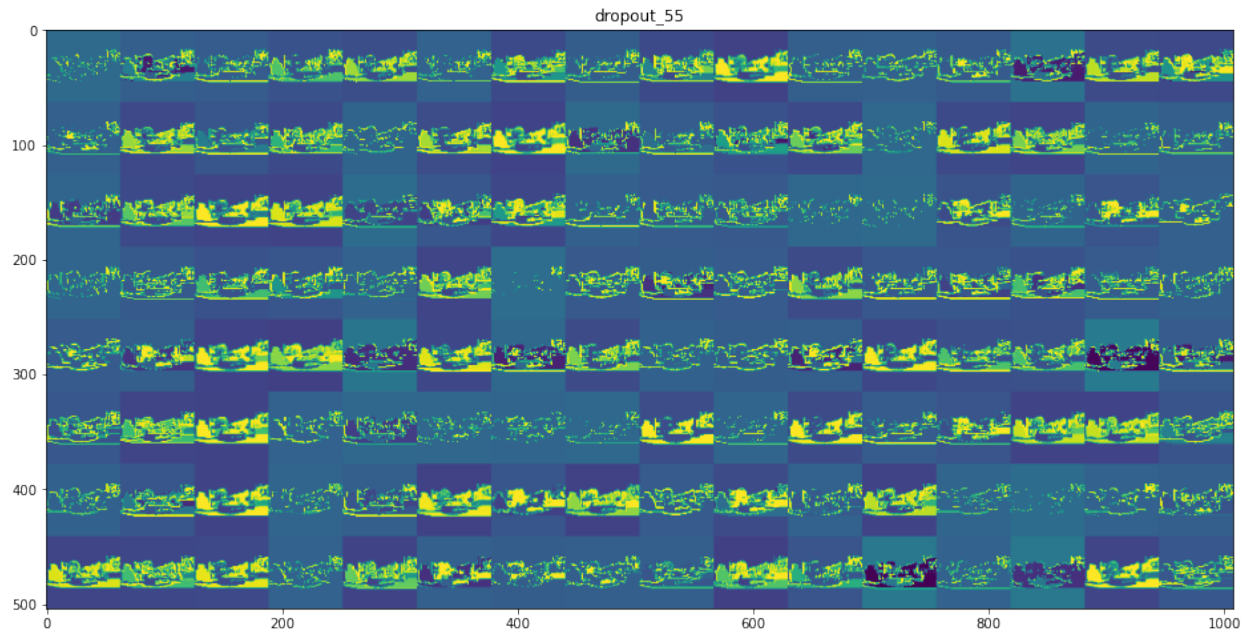


Interestingly, the training loss is much higher than the validation set, most likely because the outlier data lies mostly in the training set, but the loss did improve for both over the training epochs, so that is good.

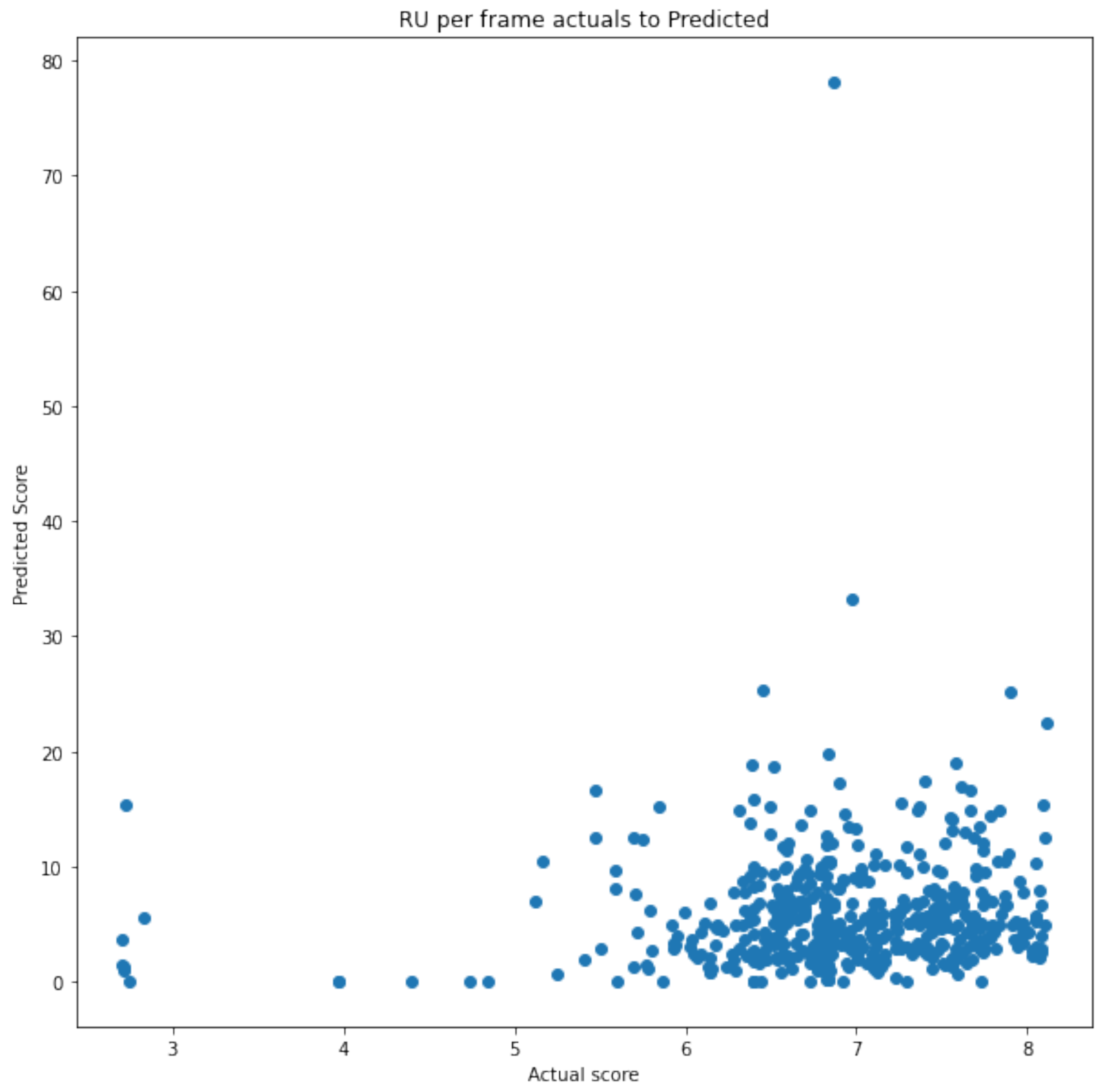
I also pulled a random shot to see what the activation layers for the CNN filters looked like, and you can see how the filters pulled out the larger features, like character and environmental structures. For instance in this thumbnail from Boss Baby 2:

Shot # 24 production: boss2 Sequence sq900

We can see the character blocking and outlines and room features and set pieces in the first activation layers filter data:



So the network is able to differentiate data that could account for the complexity of the rendered scene, which seems really encouraging to me. Here is the scatter plot of the predictions to actual values on the testing set for the first model:



For this final assignment, I also experimented to with two separate pre-baked network structures in Keras. Resnet-50 (https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50) and Inception V3 (https://www.tensorflow.org/api_docs/python/tf/keras/applications/inception_v3/InceptionV3). Both are more often used for classification tasks, but I thought it would be interesting to try to run them with regression data. I also ran one last experiment creating an ensemble of 10 CNN networks and averaging the output to see if an ensemble method would give better results.

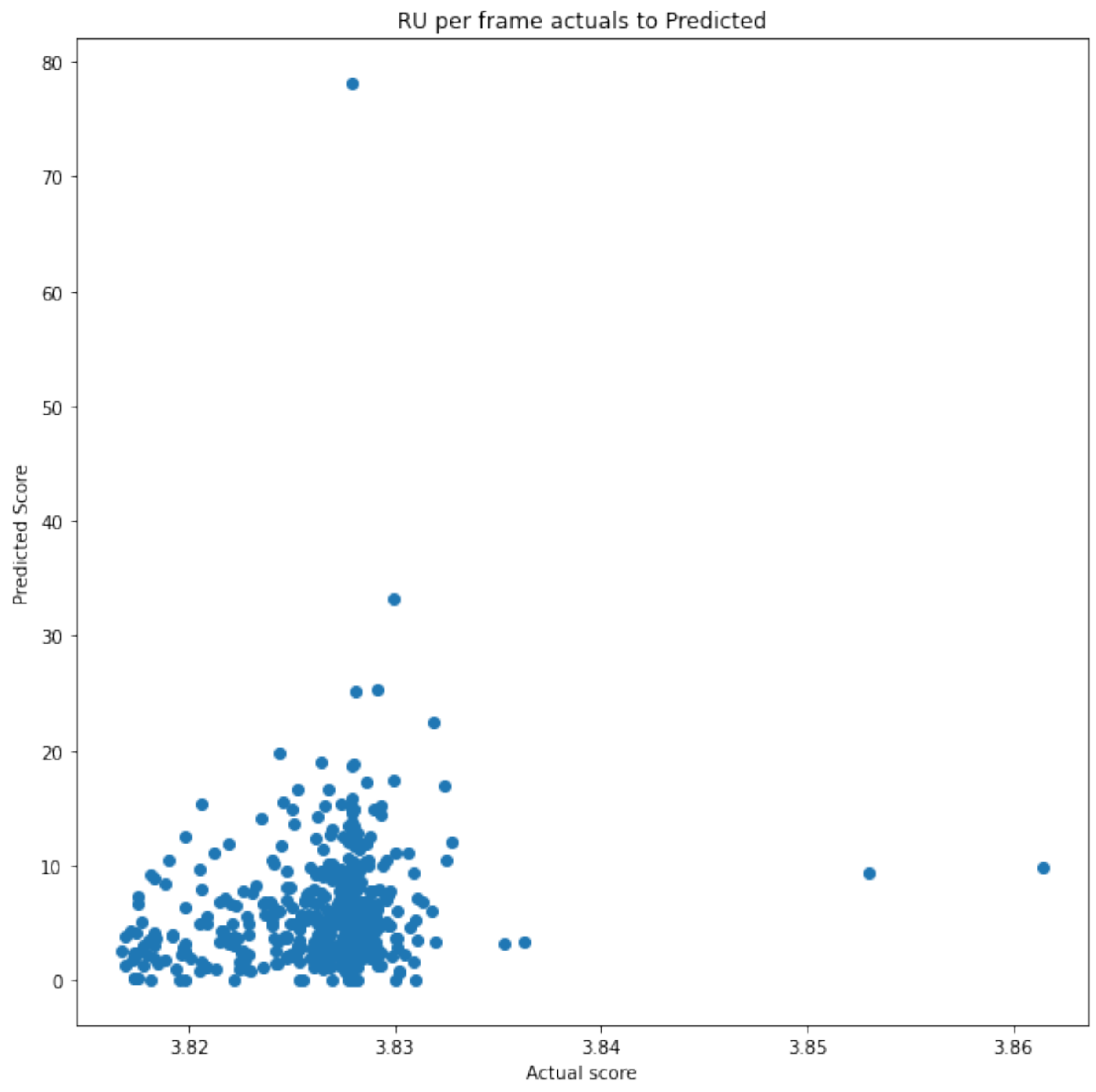
For The Resnet-50 model, the is the final metrics, learning curves and scatter plot on the test data:

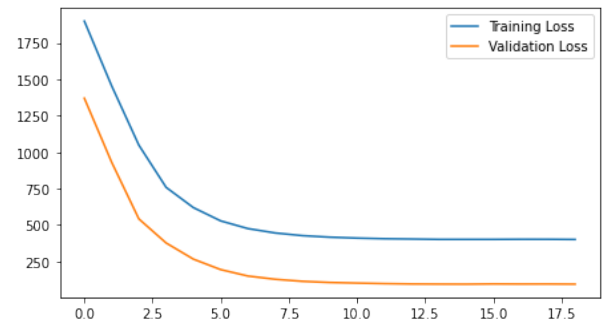
Root Mean Squared Error: 5.786627412803691

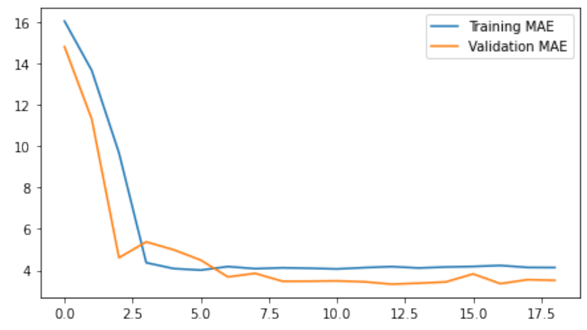
Mean Absolute Error: 1.8307093862121853

R squared score: -2268371.788027214

Explained variance: -1996146.1463323163


RU per frame actuals to Predicted
Predicted Score
Actual score
0
10
20
30
40
50
60
70
80
3.82
3.83
3.84
3.85
3.86


Training Loss
Validation Loss
1750
1500
1250
1000
750
500
250
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training MAE
Validation MAE
16
14
12
10
8
6
4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

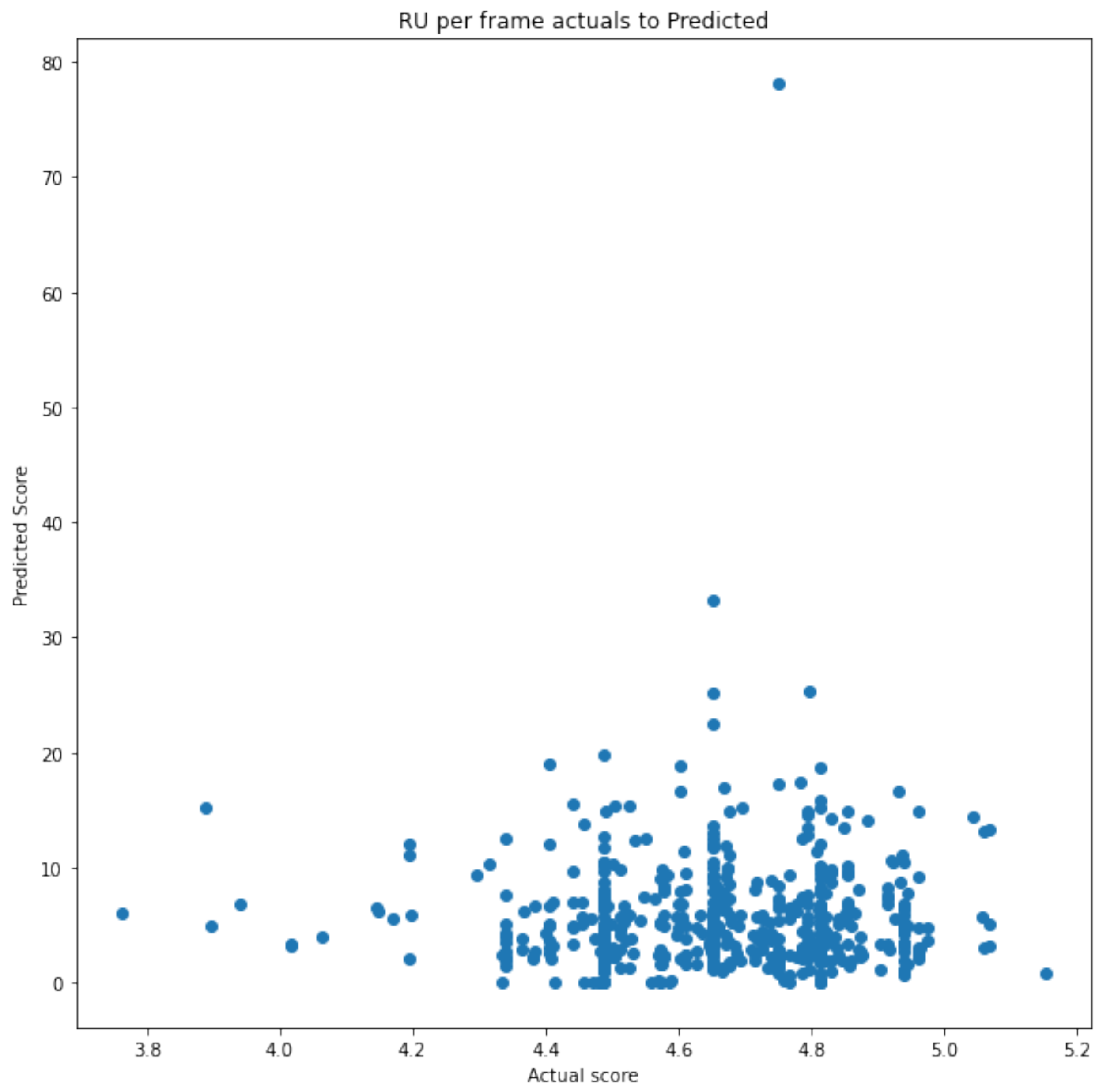
In the next trial, using Keras Inception V3 application, here are the corresponding metrics, plot and learning curves:

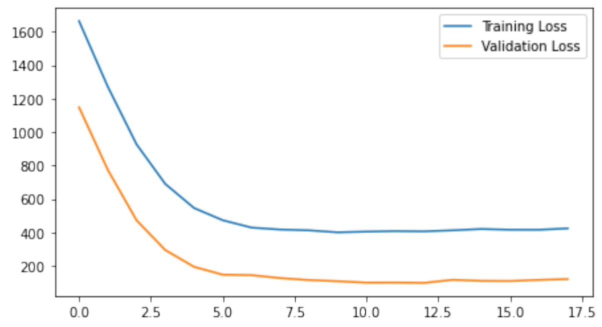
Root Mean Squared Error: 5.551600480842683

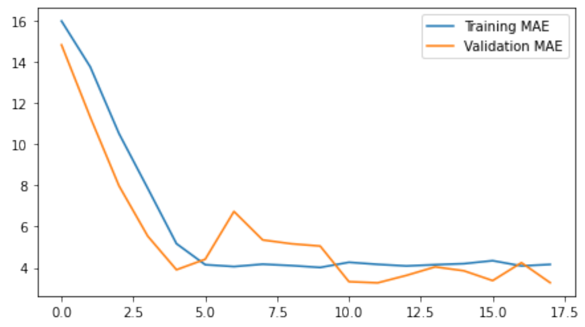
Mean Absolute Error: 1.8109273627162332

R squared score: -773.5725996714848

Explained variance: -739.4184658932888


RU per frame actuals to Predicted
Predicted Score
Actual score
0
10
20
30
40
50
60
70
80
3.8
4.0
4.2
4.4
4.6
4.8
5.0
5.2


Training Loss
Validation Loss
200
400
600
800
1000
1200
1400
1600
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training MAE
Validation MAE
4
6
8
10
12
14
16
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

The mean absolute error and RMSE are just slightly better in this experiment, but the R2 and explained variance are quite a bit better, and the scatter plot seems to fit a bit better, but there is still plenty of room for improvement I think.

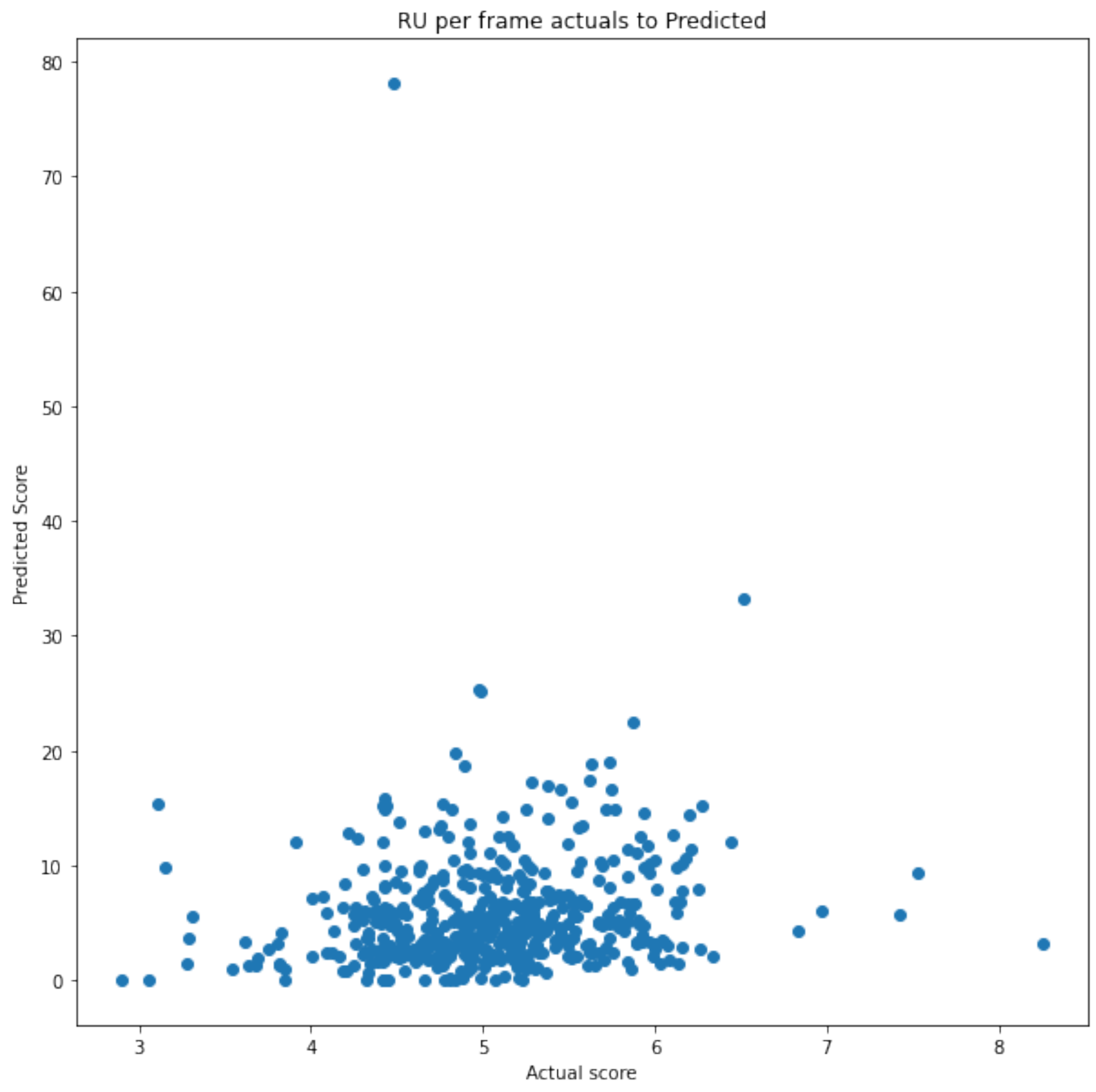
When using an ensemble technique with averaging 10 CNN's and some randomization of the training data for each model, it seemed to smooth out the error rate quite a bit, with a a small improvement to the RMSE, but the was another large improvement to the R2 scores, and the scatter plot.

Root Mean Squared Error: 5.433

Mean Absolute Error: 3.218

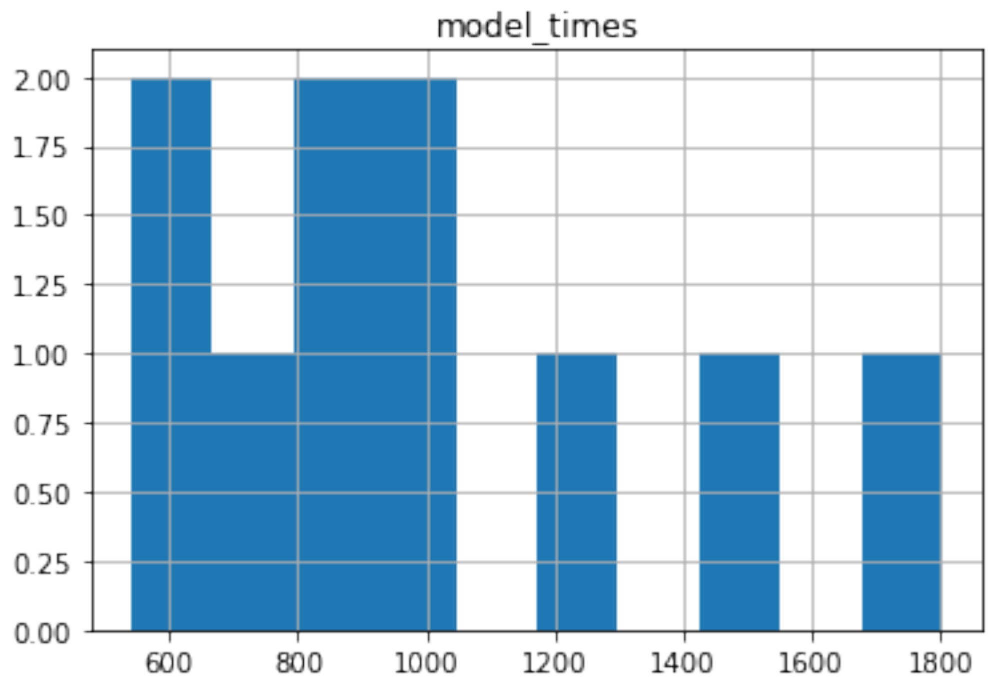
R2 score: -69.242

Explained Variance score: -67.757



The average time to train the 3 layer CNN for the ensemble was around 17 minutes per model on my Apple M1, so just under 3 hours total time. This method seems to have promise for improvement in accuracy, though, so in the future I would like to experiment with larger ensembles and some different network architectures.

Here is the histogram of model training times in seconds for the 10 CNN ensemble:



Conclusion

In conclusion, it seems like there is potential promise in determining some of the scene complexity information from the representative thumbnails using a trained CNN. I think, however, more time needs to be spent on the data preprocessing and cleaning the training data. I would also love to examine in progress movies to see how the predictions would pan out with unknown values. One other experiment I'd like to perform if I had additional time would be classify shots into classes like “FX-heavy” and “Anim-heavy” based on the department scores and perhaps using the z-score to delineate the classes, because I think converting this from a regression problem to a classification might help. All-in-all though, I think it’s a promising avenue to explore further.

Appendix/appendices

Final code:

https://github.com/ramundson/MSDS_458_Public/blob/master/MSDS458_Final/MSDS458_Final.pdf