Report

Javier A. Pollak (260446737)

March 1, 2021

1 Problem Statement

I intend to make a simple and pretty single-page webapp using React where the user uploads an image of a video thumbnail and is presented with the model's predictions on the success of their thumbnail in the form of a category of success. The information would include predicted view and comment counts as well as some example videos that are within that view and subscriber count range in order for the user to compare themself with them.

2 Data Preprocessing

I am using the Kaggle dataset at `https://www.kaggle.com/wchaktse/data-of-5132-youtube-videos`. I have downloaded all the thumbnail images –skipping urls returning 404– and categorized them into five directories numbered from 0 to 4 on Google Drive corresponding to five different categories of score. Each category contains roughly the same amount of thumbnails under it's directory. The even distribution of the thumbnails should help the model to have a more even amount of training for each category.

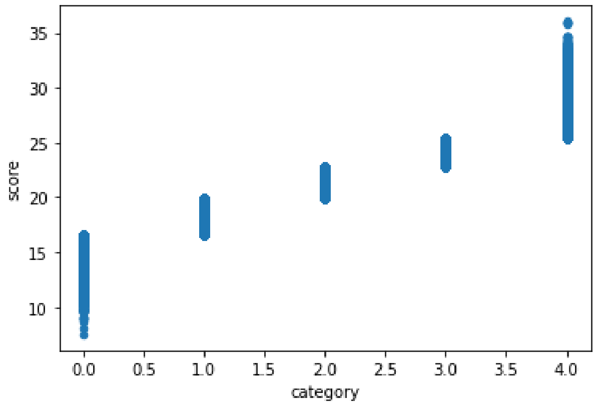

Figure 1: Thumbnail Score Distribution

The score for a thumbnail is calculated as: $score = \log(subcount \cdot viewcount)$. The reason for choosing this formula is that when looking at the Kaggle analysis of the dataset, there is a strong correlation between subsriber count and view count and a weak correlation with almost all other values. These two values are the best indicators of the success of a channel. The log is added so that the values fit in reasonably sized integers without overflowing and so that the data is more easily analyzed.

3 Machine Learning Model

I am using an Xception based Residual Convolutional Neural Network as the model for this project. I have chosen to turn the project from a regression problem to a classification problem as classification is much easier to accomplish.

I am using Google Colab to run a Python notebook with all the code for the AI part of the project. Google Colab offers TPU and GPU runtimes that significantly reduces model training times. Unfortunately the TPU runtime only works on files stored in Google Cloud Storage so I opted for the GPU runtime instead which achieves speed through using Nvidia Cuda cores.

The Xception model uses many optimization techniques; most notably, the use of Separable Convolution layers as opposed to basic convolution layers greatly reduces the parameter count and is apparently even more powerful [1] than the Inception v3 technique of factorizing convolutions [2] (e.g.: a 3×3 convolution can be factorized as 3×1 and then 1×3 to reduce learnable parameters and improve computational performance at almost no cost to the performance of the model). Separable Convolutions are also similar to the Inception block's concept of learning the best filter size through trying many in parallel although Separable Convolutions achieve this in a different way. Of course, I use batch normalization to improve the performance of the model by recentering and rescaling the output of (separable) convolution layers.

The main regularization technique used was L2 regularization on all the filters in the model. This helped the most to reduce overfitting with an optimal weight decay value I found of around 0.005. There is also a dropout layer at the end of the model that I kept from the Xception model, but I haven't tried changing its dropout rate too much.

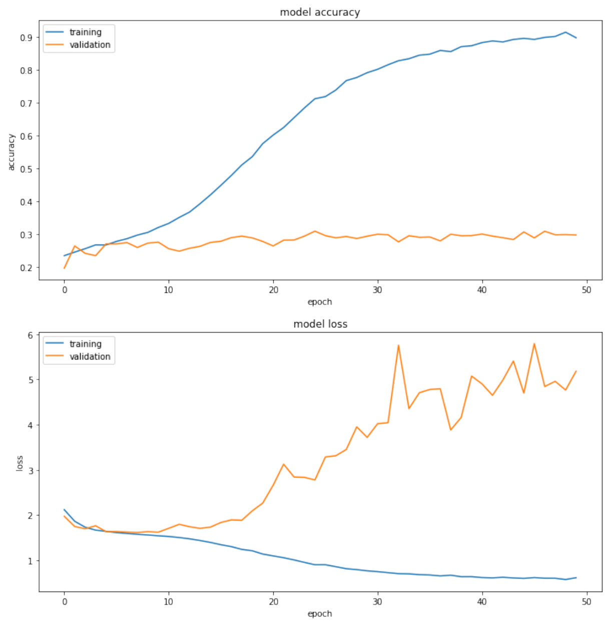

Figure 2: Mini Xception model before regularization

At first it seemed like my model was drastically overfitting with a training accuracy of 0.9 and a validation accuracy of 0.3 by the end of the 50th epoch. After regularizing my model, I was able to "fix" the overfitting but my model was barely learning anything from the training data and the accuracy very slowly rose to 0.33 for training and 0.28 for validation.

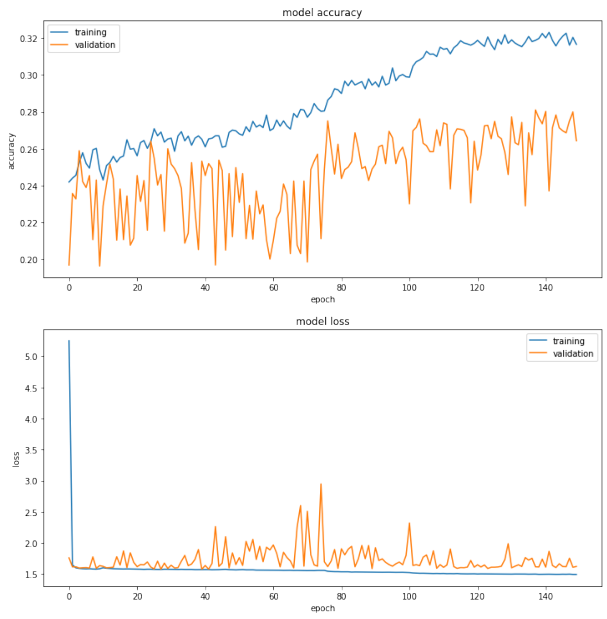

Figure 3: Mini Xception model regularized with `weigth_decay = 5e-3`

4 Preliminary Results

As can be seen in 3, my results were extremely underwhelming. My model is now either extremely slow at learning or will always plateau at an accuracy of 0.3. In either case, the problem-model combination here is not very feasible as it stands. This could be due to having too many categories of thumbnail success with not enough distinguishing features for the model to pick up. The high training accuracy at first in 2 seemed like themodel may have been merely overfitting and in need of regularization, but it seems that that was not so simply the case.

5 Next Steps

My first next step will be to reduce the number of classes from 5 to 3. This way the model should be able to pick up on more features for each class to distinguish them from eachother more easily. Another effect of the lower class count is that there would be more data to train with for each class, but that is just a side-effect as I already have plenty of data to train with (over 25k images in total).

References

[1] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[2] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.