Aluya Omofuma and Angelica Wiltz

Final Project Report

Professor Amir Jafari

December 3, 2019

Model Comparison - CNN and VGG16

Introduction

Our project is an examination of a deep learning model that is used for detecting the age of individuals. We have a dataset containing images which have the age, sex, race and time and date of the picture embedded in the file name. We wanted to compare the accuracies of two models to evaluate the performance of the models on our dataset. The two models we chose were a CNN model that was constructed manually and a pre-trained VGG16 model. Our reference for our CNN model came from multiple kernels on Kaggle and our pretrained VGG16 model was downloaded from the Keras library in Python. In addition to comparing each of the models, our goal was also to improve the reference models to see if we could increase the accuracy of the model. The report below gives an analysis of the dataset and why we chose it for our project, the construction of our CNN and VGG16 models and our decision making process, an analysis of the results of both of the models including the accuracy of the training sets and accuracy of the prediction of the training sets, and our final results and conclusions.

Dataset Description

We chose our dataset based on the nature of the project and the type of model we wanted to test which was a CNN model. Initially we were examining a dataset for music analysis and were looking to compare the accuracy of a CNN model versus a CRNN model. However, after doing further research, we decided to change our dataset from music analysis and sound

classification to age detection for image classification. We retrieved our dataset from a GitHub repository created by graduate students from The University of Tennessee, Knoxville. The dataset was a face dataset that consisted of 20,000+ face images which are associated with three labels = age, gender, and ethnicity as well as landmark locations. The images were collected from the internet and the data was split into three separate files, “In-The-Wild” Faces, “Aligned and Cropped” Faces, and landmarks texts that were associated in the images. The labels of age, gender, race, and date and time were embedded and formatted into the file name. They were also checked using the DEX algorithm as well as a human annotator. The “In-the-wild” faces were inclusive of various landmark backgrounds while the “Aligned and Cropped” faces were cropped to solely show the face of an individual in an image. We initially downloaded the “In-the-wild” face dataset images to use to test our models. However, the dataset was split into three separate files and contained corrupted images. We decided to use the “Aligned and Cropped” faces instead which were separated into two files - “crop_part1” and “UTKFace” dataset. The “crop_part1” data contained the same images that were in the “UTKFace” dataset but was only a partial representation of the UTKFace dataset. We decided to just use the UTKFace data and proceed with our project. The UTKFace dataset contained 23,709 images of aligned and cropped faces of individuals. We trained and tested our model using the UTKFace dataset.

Network Description

We created a Convolutional Neural network using the Tensorflow framework. The model is a sequential model that has its base as a convolutional layer with 64 nodes. The kernel size is the size of the filter matrix for our convolution. So a kernel size of 2 means we will have a 2x2 filter matrix. We evenly pad the images and use the relu activation function on resized images

with 32x32x3 = 3072 weights. The output from this layer undergoes a max pooling process which reduces its dimensionality(downsamples) and halved the size of the dimensions. It aids with overfitting. This reduces the number of parameters also reducing the computational cost. The output is then processed for a dropout. This due to the large sample we are dealing with and helps with overfitting as mentioned above. The output is then fed into another convolutional layer with 32 nodes and more max pooling and dropouts to make it easier to process the inputs. The output of the two layers are then flattened since the channel dimension is missing. Flattening adds an extra dimension to the output of those two layers, it serves as a connection between the convolution and Dense layers. Before it goes into the output layer, a larger dropout is conducted on the data and it is fed into the output layer to return the results.
The output layer uses a softmax activation function because it is a classification problem, softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which class has the highest probability.

CNN Model and Pretrained Model

Conducting a model comparison was the initial goal of our project, and we initially wanted to compare a CNN model to a CRNN model. After changing our dataset and receiving guidance from our course instructor, we decided to compare a CNN model to a pre trained model. We chose to use a previously constructed CNN model that we retrieved from a Kaggle kernel that had tested the UTKFace dataset on a CNN model. This particular kernel classified age based on sets rather than age integers and this stood out to us. The code subsetted age based on five different class ranges - Children (1 years old - 4 years old), Youth (age 14 - age 25), Adults (age 25- 40), Middle age (age 40 -60), and Very Old (age 60 and older). The code

transformed the classes into categorical labels for the output and split the data into x training and y training.

Our pretrained model of choice was a VGG16. We chose to use a VGG16 as opposed to ResNet or Inception v3 as a result of its success of achieving 92.7% accuracy when tested on the ImageNet dataset in 2014 during the Large Scale Visual Recognition Challenge in 2014. It made improvements by implementing 3x3 sized kernel filters and used NVIDIA Titan Black GPU with a 224x224x3 input size. It is comprised of twelve convolutional layers, one flattened convolutional layer, and an additional dense layer to predict the output. VGG16 works well with large datasets, so since ours wasn't as large we decided it would be a good option.

Experimental Setup

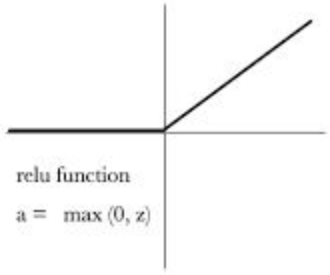
After choosing our models and deciding to use a Keras framework, we selected the "Aligned and Cropped" images to apply to our CNN model and VGG16 model. Initially, for our VGG16 model, we did not construct any additional layers or tamper with the convolutional layers that were provided inside the model. When we first tried to run the model, we received errors due to the input shape/sizes. We first resized the first input layer of the VGG16 model to match the size of the input size that we chose for our X dataset (32,32). However, this returned an error stating that the input array was expected to be 224, 224, 3 but was instead 32,32. After reconsidering our input shapes we decided to allow the first layer of the VGG16 model to remain as an 224x224x3 input size and we reshaped our dataset to fit the model. This eliminated the input array error. Afterwards we continued to follow the format of the initial code that we found that tested the same UTKFace dataset and split the data into training and testing sets with 70% of the data being used for training and 15% used for testing and validation.

In order to judge the performance, we computed the accuracy, loss, and a plot of the age classifications that the model was able to classify based on our models. We did not use mini-batches, but instead used the categorical cross entropy loss function, adam optimizer with a learning rate of 0.001, batch sizes (different sizes for each model) and accuracy as our metrics. We used a batch size of 2 (due to low memory) for our VGG16 model and a batch size of 64 for our CNN model. We decided to use the categorical cross entropy loss function since our target outputs are in categorical format between multiple classes, and the function assigns a single label to each input. We chose the Adam optimizer based on research that showed that the Adam optimizer with a learning rate of 0.001 works better and learns faster in comparison to other optimizers and learning rates such as Adagrad, Momentum, GD, Adadelta, and RMSProp (https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2).

Results

For our pretrained VGG16 model, we first ran the x and y training sets through the model with an input size of 224x224. We did not change or tune any of the hyperparameters, nor did we add any additional layers. As a result, our epochs only returned accuracies between 20% and 40%. In an attempt to increase the accuracy for our model, we added two additional dense layers in order to push more neurons through the model. We set the units to 5000 on each layer, since our training sets were comprised of 15000 units and we wanted to have a good amount of additional inputs to process. We also added a dense layer comprised of 6 units which represented the classes (outputs) that we were training the model to classify. For the first two

dense layers we chose the ReLU activation function which ensures that all negative values return an output of zero.



We applied the softmax function to the final dense layer that featured our class sets in order to normalize the inputs into a sum of up to one .

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

After compiling the model, our model returned a 50% accuracy on the training set and an 18% accuracy on the testing set. Although this was better than what we previously had before, we still needed to make some major improvements. We decided at the last minute to remove the additional dense layers that we had added and to choose another approach. We added “imagenet” weights to our VGG16 model and assigned “x” as the output for the model. We then took “x” and applied it to Global Average Pooling 2D which reduces the dimensions of the data to prepare for the final classification layer. We calculated the predictions to match this layer and fit the inputs from the VGG16 model with our predictions as our output. After running our model

we were able to see a significant increase in our accuracy from 50% to 78% for our training sets and 68% for our testing sets.

Our CNN model was pretty similar to the model we retrieved from Kaggle, with the exception of adding additional class sets to make a better classification for the age ranges contained in the dataset. We also added an adam optimizer with a learning rate of 0.001 which was not included in the original code. The original model in the code achieved an accuracy of 63% on the testing data, while our improved model was able to achieve an accuracy of 74% on the testing data which was a significant increase. As a way to depict our results, we plotted an example of a classification prediction that our model made. For the most part, our model classified the age groups of the individuals in the images correctly, with the exception of the man who is a senior citizen but classified as a young adult.

Summary and Conclusions

We were able to achieve both of our goals for our project which were to improve a CNN model using the UTKFace dataset as well as compare the performance of a CNN model to a

pretrained VGG16 model. We were able to increase the performance of the CNN model by 11%. From analyzing the results for the accuracy on the x and y test sets, we can conclude that the CNN model had a better performance than the pretrained model, although the pretrained model had a significant accuracy on the training set. One thing that we observed was that increasing the class sets turned out to have a significant difference on improving both the CNN and VGG16 models. In the future if we were to spend more time working on this model, we would try to experiment with different optimizers to see what effect optimizers other than Adam would have on the performance of the model.