Stephen O'Neil

CS6140

Final Project

Abstract: **This project focuses on the Stanford Dogs Dataset. A convolutional neural network and a fully connected network were created to correctly classify dog breeds based off of RGB images. The results between each neural network were compared and it was determined that the convolutional neural network outperforms the fully connected network but is still very prone to overfitting of the dataset.**

Overview:

The main problem being solved in this project is image classification. There is a set of images containing 120 dog breeds, and the problem is that there is no neural network to classify them into groups. The motivation behind this problem is that if a neural network is created that can accurately classify dog breeds, then it could likely be easily generalized to classify other things as well. Possible expansions would be to classify multiple different subspecies of animals including dogs. This is an interesting problem because the machine learning task of image classification is a very common problem in industry today. Very large and robust models are being created to be able to classify a large range of items. One major task that these models are undertaking is their usage in self-driving car systems. These classification models can be used to classify objects detected in front of cars, determining what actions need to be taken.

Using a neural network to solve this problem was a clear approach. Neural networks are very useful for processing images and making predictions. In addition to this, convolutional neural networks are even better for image classification, making for an initial hypothesis that the CNN will outperform any fully connected network. This problem of image classification is a very common one, especially with the stanford dogs datasets. It appears that almost all approaches use some kind of neural network for classifying the dog breeds. The key components of this approach is the neural network architecture. The different layering of dense and convolutional layers was very important in creating a usable model. The key result is the training and test accuracy on the dataset. With a properly performing model, the training and test accuracy should be fairly high, and the neural network should be able to classify images that it has never seen before. One major limitation found during the project was computation time. Google Colab was utilized for GPU acceleration, however even with this acceleration the training time for the models was still rather high. The cause of this limitation was likely due to the very large dataset being used, where the train/test data split was around 700 MB and 100 MB respectively.

Experiment Setup

The data set being used is the Stanford dogs dataset. This dataset consists of pictures of 120 possible dog breeds, having 20,580 total pictures. The total size of the dataset is 787 MB. The images obtained from the dataset are typical RGB jpeg files, having dimensions of 256 x 256 x 3. One interesting aspect of the pictures is that there are other artifacts in some files such as people standing next to a dog or multiple dogs in the image. In the dataset the split of images

for each dog breed is not consistent. Popular dog breeds such as German Shepards or Golden Retrievers have around 200 images in total, where more unpopular breeds can have under 100 images. This may lead to popular dog breeds being more likely to be classified correctly rather than less prominent breeds. This dataset was obtained directly from Kaggle and is a rather popular dataset to be used for image classification.

To classify the images obtained from the dataset multiple neural networks were created. The first network was designed to be a baseline, consisting of a simple fully connected neural network using dense layers of the TensorFlow library. This first network only had a flattening layer, a dense layer, and an output layer. The second network created was a baseline for the convolutional neural network. This network consisted of an input layer, a hidden convolutional 2-dimension layer followed by a flattening layer and a dense fully connected layer. All networks created had an output layer of 120 neurons, one for each possible dog breed. For each layer ReLU activation was used, with a softmax output activation.

The main classification model was created using a convolutional neural network. It was based off of the architecture of AlexNet. This means that multiple convolution layers were stacked on top of one another before pooling layers, allowing for more parameter training. The final network architecture began with a convolution input layer using a kernel size of 4x4 and an input shape of 256x256x3. It was followed by a pooling layer using a 2x2 kernel. This initial layer was followed by multiple 2-dimensional convolution and max pooling layers. These convolution layers had a kernel size of 3x3 and ReLU activation. Following these layers were the stacked convolution layers. 3 convolution layers were used, each using 200 filters and a kernel size of 1x1 allowing for maximum parameter training. Each of these layers use ReLU activation, and the stack was followed by a max pooling layer. This stack was repeated 1 more time in the network. After the stacked convolution layers, a flattening layer was used to convert from convolution layers and dense layers. There were 2 dense layers used, consisting of 2056 and 1024 neurons, both using ReLU activation. These two dense layers were followed by the output layer with 120 neurons, using softmax to determine the classification of the dog breed. In addition to the architecture many overfitting mitigation was added. Each layer was followed by a dropout of 0.5 alongside batch normalization. The initial data was also normalized, and random image flipping was added. The SGD optimizer was used with a learning rate of 0.01, and SparseCategoricalCrossEntropy was used for both loss and accuracy measurements. All of the networks were implemented using Google Colab Pro and generic GPU acceleration provided by the service. The TensorFlow library was used for the implementation of the neural networks.

Experiment Results

The baseline models performed as expected. The fully connected network was very quick to converge on the training data only taking approximately 5 epochs, however it was very prone to overfitting. When evaluating on the testing data it was only able to perform at about 10% accuracy on images it had never seen before. The baseline CNN performed very similarly. It was again quick to converge but did take longer to train on the dataset. It provided similar accuracy results, reaching about 98% accuracy on the training set and only 12% on the test set. The AlexNet based network performed much better in terms of accuracy. The larger network took significantly longer to train, likely due to the overfitting mitigation, taking 200 epochs to reach 92% accuracy on the training data. The final network was able to reach around 40% accuracy on the testing data, outperforming both baseline models.

One of the most interesting results was the effect that overfitting factors had on the convergence of the model. When using kernel regularization, the model would always converge at the same local minimum value of 1.24% accuracy regardless of L1 or L2 regularization. In addition, adding too many convolution layers would result in reaching the same local minimum. The dropout rates seem to provide the most benefit at 0.5, where a lower value does not mitigate overfitting enough but a higher value prevents the model from properly training on the data. Similar results were obtained when using too many image modification layers on the input. If the input data was modified too much it prevented the model from actually learning on the training data, only reaching about 50% accuracy on the training data. The size of the dense layers also played a large role in the design of the model. The dense layers provided significantly more parameters to the model than the convolution layers, but were very prone to overfitting on the dataset. If the dense layers had too many neurons then the model would train very quickly and reach a higher accuracy on the training set, but perform much more poorly on the test set. The inverse was true when using too few neurons where if the layers were too small then the model had more trouble performing well on the training set, but performed better on the test set.

Discussion

Overall the neural network did not perform as good as hoped for. A testing accuracy of 40% is generally not a very good result and would not be practical for any applications of this network on other images. The main limiting factor on the performance of the model was the constant problem of overfitting. Many different overfitting mitigation techniques were used in the development of the model, but very few seemed to make much impact on the overall performance. The most useful techniques used were using dropout layers and image modification layers, but these had their downsides as well. Many of the techniques such as kernel regularization, image rotation, and lowering the ratio of convolution and dense layer parameters led to the model being unable to properly train on the provided data. Removing or lowering their prevalence in the model led to better training results, but again worse testing results. Testing these factors effects on the model was not necessarily difficult, but very time consuming. Some cases the overall results of adding or removing a factor couldn't be evaluated until the model was finished training completely which sometimes took as long as 20 minutes. This process of tuning the parameters of the model was very time consuming, but very useful in visualizing what effects each factor of the model had on its overall performance.

If the project were to be recreated starting from a more bottom-up approach would likely be beneficial. Basing the model architecture off of the preexisting AlexNet provided a large leap in performance from the baseline models, but it was difficult to determine how much of an effect the architecture of the network had on its performance. Starting from the baseline model and slowly adding layers and overfitting mitigation factors would likely have been a better approach. It would be easier to identify which aspects of the network were causing the problems, and which were aiding the model to learn the parameters better. A continuation of this project is very possible. Future improvements would include improving the models accuracy and potentially lowering the training time of the model while preserving its performance.

Conclusion

Overall while the model did not perform as well as hoped for, much was learned during this project. The effects of adding overfitting mitigation were examined and it was determined that there is a delicate balance between preventing overfitting and allowing the model to properly

train on the data its provided. In addition, the effects of L1 and L2 regularization were examined particularly how they caused the model to reach a consistent local minimum value regardless of how they were used. If the project were to be recreated a more bottom-up approach would likely be useful, and using better hardware for faster training times would be recommended. This project did not provide a very robust model for dog breed classification, but did provide a strong understanding on the importance of neural network architecture and how to improve a models performance.

References

Dataset:
http://vision.stanford.edu/aditya86/ImageNetDogs/
https://www.kaggle.com/datasets/jessicali9530/stanford-dogs-dataset

Initial setup:
https://www.kaggle.com/code/atuflin/starter-stanford-dogs-dataset-8250c659-1
https://www.analyticsvidhya.com/blog/2021/06/how-to-load-kaggle-datasets-directly-into-google-colab/

Comparison:
https://github.com/ayushdabra/stanford-dogs-dataset-classification