Dog Breed Identification using CNN

Abstract:

The data set used is the Stanford dog breed dataset. The data set consist of images of 120 breeds of dogs. Each image has a filename that is its unique id. The goal of the competition is to create a classifier capable of determining a dog's breed from a photo. Using Keras with Tensorflow backend we build a Convolution Neural Network for image classification. I build the CNN by using different network architectures, layers, activation functions, optimizers, network initializers and epochs to obtain the best performing model for our given task of dog breed identification. Apart from the parameters that were required to be used, I also explored the effect of having different learning rate, batch size and image dimensions. The model has been trained on the local machine and using the Google Colab GPU. I have performed several experiments for this problem. I have summarized a few experiments and their architecture and their performance. The metric used to evaluate the models was accuracy and the model loss.

Approach:

Part A:

Since the problem is an image classification problem. The network I used was a Convolutional Neural Network.

Part B:

Activation functions used were RELU, LeakyRELU, and ELU

Part C:

Cross Entropy and Quadratic cost (Mean-square-error)

Part D:

The number of epochs used were 200,150,250,20 etc.

Part E:

Adam, Stochastic Gradient Descent, RMSProp

Part F:

Tried several network architectures by changing the number of layers, type of layers, varying drop-out rates and using different parameters to tune the base model. Explored a pre-trained model called Xception.

Part D:

Networks initializers used were he_normal and constant.

Experiment Summary

Experiment 1:

Number of Layers: 6

Epochs: 200

Activation functions: RELU, Softmax(output)

Cost function: cross- entropy

Optimizer: Rms prop

Accuracy: 7%

Loss: 11.99

Learning rate: 0.001

Batch size: 128

Inference: The model was trained using CPU and it took approximately 8 hours. The model accuracy and the loss were flat through most of the epochs.

Experiment 2:

Number of Layers: 8

Epochs: 250

Activation functions: ELU, Softmax(output)

Cost function: mean squared error

Optimizer: Adam

Accuracy: 6.5%

Loss: 0.009

Learning rate: 0.0001

Batch size: 128

Inference: The model was trained using CPU and it took approximately 8 hours. The changes made did not have any significant impact on the accuracy and loss for the test data.

Experiment 3:

Number of Layers: 8

Epochs: 200

Activation functions: ELU, Softmax(output)

Cost function: mean squared error

Optimizer: rms prop

Accuracy: 0.8 %

Loss: 16.22

Learning rate: 0.001

Batch size: 128

Inference: The model was trained using the google colab GPU. The time for training was approx 5 mins.

Experiment 4:

Number of Layers: 8

Epochs: 200

Activation functions: ELU, Softmax(output)

Cost function: mean squared error

Optimizer: Adam

Accuracy: 1.10

Loss: 15.81

Learning rate: 0.0001

Batch size: 64

Inference: Batch size of 64 did impact the performance. With a smaller number of epochs it is giving more accuracy compared to the previous model.

Experiment 5:

Number of Layers: 8

Epochs: 200

Activation functions: ELU, Softmax(output)

Cost function: mean squared error

Optimizer: Adam

Accuracy: 0.64

Loss: 16.01

Learning rate: 0.0001

Batch size: 32

Inference: Reducing the batch size from 64 to 32 make the networks performance worse. We will keep our batch size 32 and also more epochs are not helping so we reduce our epochs too.

Experiment 6:

Number of Layers: 13

Epochs: 2

Activation functions: RELU, Softmax(output)

Cost function: cross entropy

Optimizer: Adam

Accuracy: 0.81

Loss: 15.98

Learning rate: 0.001

Batch size: 64

Inference: This model did make the accuracy slightly better. However, it did not show any significant changes.

Experiment 7 (Best Model):

Number of Layers: 18

Epochs: 20

Activation functions: RELU, Softmax(output)

Cost function: cross entropy

Optimizer: Adam

Accuracy: 10.79

Loss: 4.43

Learning rate: 0.001

Batch size: 128

Network initialization: he_normal

Inference: This model performed extremely well and gave accuracy of 10%. The new architecture with extra batch normalization layers and kernel initializer

Experiment 8:

Number of Layers: 13

Epochs: 20

Activation functions: RELU, Softmax(output)

Cost function: cross entropy

Optimizer: Adam

Accuracy: 1.36

Loss: 4.78

Learning rate: 0.001

Batch size: 64

Network initialization: Constant

Inference: By changing the network initialization method, the performance of the model deteriorated.

Experiment 9:

Model: Xception Pre-Trained model

Epochs: 150

Accuracy: 1.3

Loss: 4.78

Cost function: cross entropy

Optimizer: SGD

Learning Rate: 0.001

Batch size: 32

Inference: Pretrained model need to be fine tuned before using. Directly using it did on provide significant improvement.

Experiment 10:

Model: Xception Pre-Trained model. Changed the base to improve performance.

Epochs: 50

Accuracy: 0.6

Loss: 4.84

Cost function: cross entropy

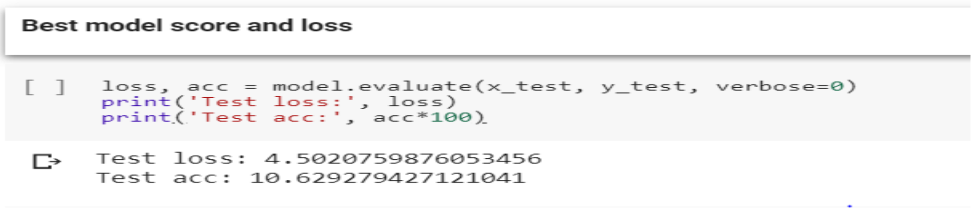
Optimizer: SGD

Learning Rate: 0.001

Batch size: 32

Inference: Changed the base of the Xception model.

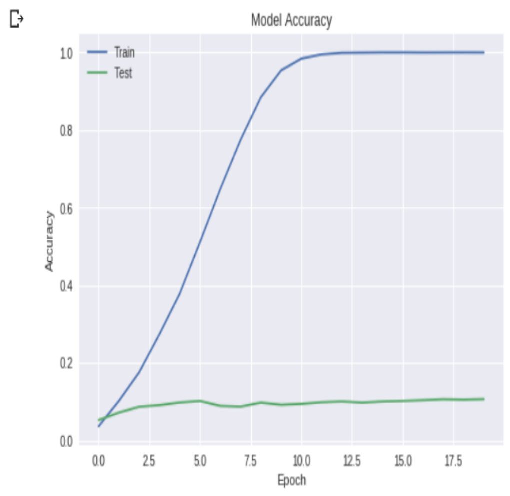
Best Model Accuracy and Loss

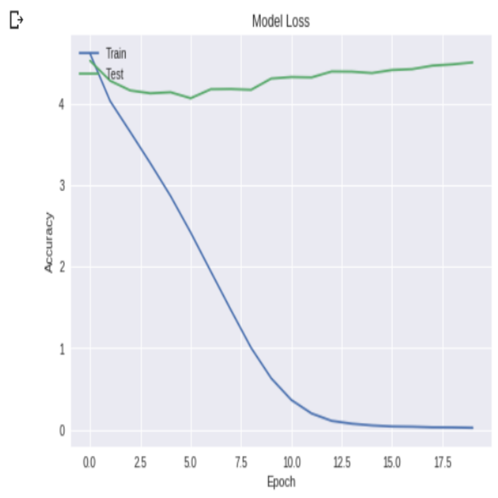


Best model score and loss

```
[ ]   loss, acc = model.evaluate(x_test, y_test, verbose=0)
      print('Test loss:', loss)
      print('Test acc:', acc*100)
```

```
Test loss: 4.5020759876053456
Test acc: 10.629279427121041
```





Conclusion:

Experiment 7 gave the best model with accuracy 10.62% and loss of 4.53. The model consists of 18 layers. One of the main reasons for the performance of this model is the architecture. After each convolutional layer and batch normalization layers is added. This minimized the loss of information in the feed forward network. Also, a bigger batch size had a significant impact. After all the experiments the batch size of 128 was found to be ideal. Kernel initialization method was an important factor as it outperformed other kernel initialization methods. The accuracy score was obtained by running just 20 epochs. I believe that the performance can be improved by using drop-out and by experimenting with different learning rate and optimizers.

Contributions:

1. Ran the models on both GPU and local machines
2. Experimented with different architecture and parameters as per the assignment requirement
3. Used pre-trained model as a base model for the classifier.

References:

1. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
2. https://keras.io/getting-started/sequential-model-guide/
3. https://www.datacamp.com/community/tutorials/deep-learning-python
4. https://arxiv.org/abs/1610.02357

License:

Must have a license such as the MIT License https://opensource.org/licenses/MIT