Data Science Project Report

Principal Investigator

Michael Curry – currymike@gmaillcom or currym6@newpaltz.edu

Title of Project

Convolution Neutral Network design for the Stanford Cars 196 dataset.

Code

https://github.com/currymike123/DataScience_CNN

Mentoring

Professor Min Chen, Department of Computer Science, SUNY-New Paltz
chenm@newpaltz.edu

Introduction

My motivation for this project was to begin a study of computer vision and more specifically Convolutional Neural Networks. I have a deep background in manually manipulating digital image data from my work in image processing. Removing the human hand and finding meaning and representation through algorithmic techniques has sparked my interest for many years.

My goal was to construct a Convolutional Neural Network for image classification on a medium sized dataset, which has not been pre-processed, and does not have a large background of study. This would allow me to experiment with the full pipeline of a deep learning workflow and limit the previous solutions available for search.

Once I construct my deep learning pipeline the main objective of my project would be to experiment with Convolutional Neural Network design. The explicit goal is to construct a design that would maximize the accuracy of my model. The implicit goal is to fully grasp the top to bottom workflow of deep learning and use this knowledge to design real world computer vision tools.

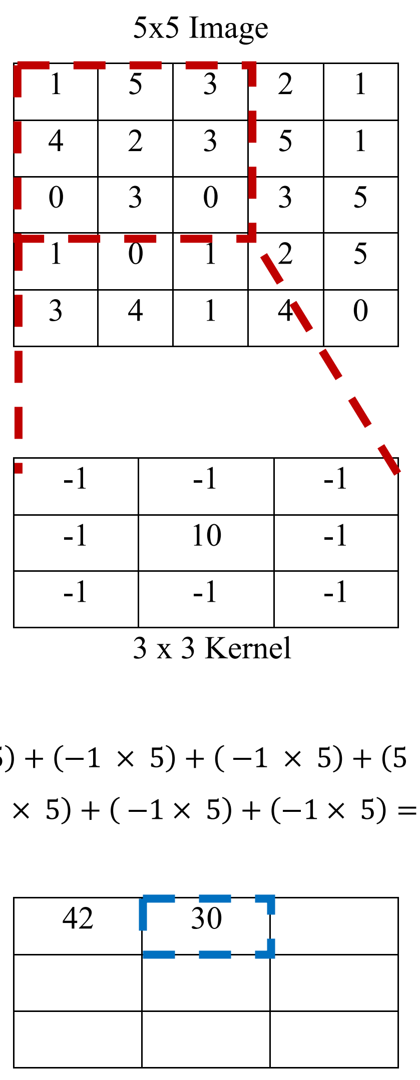
Background/History of the Study

Convolutional Neural Networks function similarly and are influenced by the design of the human visual system. Each neuron layer in our visual system filters certain features within its receptive field. The receptive field being the range of stimuli that provoke a sensory response. [1]

The study of receptive fields in the visual cortex began in the 1950's and 1960's by David Hubel and Torsten Wiesel. They identified the different neuron filters and their arrangement in the eye. Their work was used by Kunihiko Fukushima in 1980 to form Neocognitron architecture. The computational inspiration for the Convolution Neutral Network.[1]

Neurons in the eye are stacked and overlap so the sensory response from the higher layers are fed as input to the layers below. Since neurons each filter different sensory input the combination of these inputs produce the complete receptive field. The complete receptive field (feature map in computer vison) [3] is fed to the brain which processes and forms representations to form what we see.

Convolution Neural Networks or CNN use a similar construction. A filter called a kernel is moved across the input image data, emphasizing and deemphasizing features.[3]. The kernel is a matrix of numbers which slides over the pixel data emphasizing and deemphasizing features contained in the data. Figure 1 (below) illustrates the matrix operation on a 5 x5 image with a 3x 3 edge detection filer.



1	5	3	2	1
4	2	3	5	1
0	3	0	3	5
1	0	1	2	5
3	4	1	4	0

-1	-1	-1
-1	10	-1
-1	-1	-1

$$(-1 \times 5) + (-1 \times 5) + (-1 \times 5) + (-1 \times 5) + (5 \times 10) + (-1 \times 5) + (-1 \times 5) + (-1 \times 5) + (-1 \times 5) = 30$$

42	30	

Figure 1

A CNN stacks layers of these filters which encode features within the image input data. The top layers of the network encode simple features such as shapes, textures, and colour combinations. As you move deeper into the network these simple features combine to form more complex arrangements. For example, if you were encoding an image of a car. In the top layers you would expect to see simple shapes such as circles or horizontal/vertical lines. As you move deeper into the network the shapes will combine to form complex shapes like wheels or the grill. The ability to pull from a simple set of low level features and use them as building blocks to complex shapes is what forms a CNN's ability to efficiently encode image information.

On top of the convolutional network a standard neutral network is placed for the purpose of classifying the feature output. A neural network is composed of layers of connected neurons which all perform a simple computation. (Figure 2)

$$y \ = wx + b$$

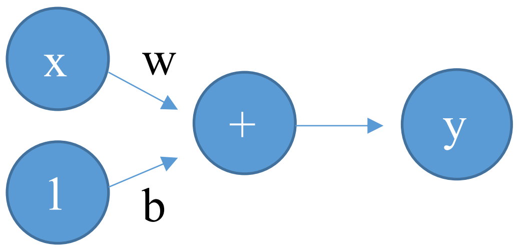
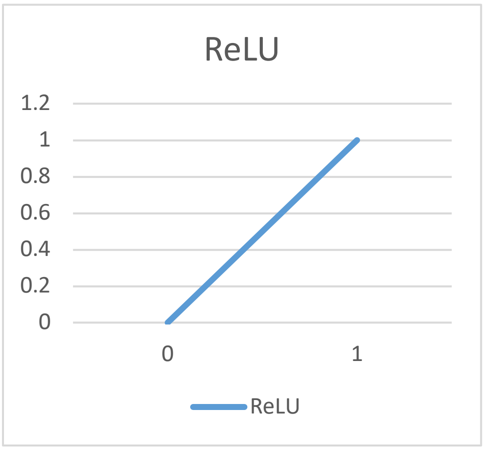


Figure 2

This formula is familiar to all since it's the same as the formula for calculating the slope of a line. This is essentially what a neutral network is doing at each neuron. X is the input from the network, in our case image pixel data, and it is multiplied by W the weight of the neuron. B is the bias of the neuron in this example it is kept a 1 which is standard for a CNN.

Intuitively, the larger the value of X & W the larger the sum of the equation and the value contained in the neuron. This simple relationship and the ability of the network to adjust its weights (which I'll soon) gives a neural network the power to encode complex relationships within data.

The output is then sent through an activation function such as ReLU (Figure 3), Sigmond, or Softmax.



$$f(x) = \max(0, x)$$

Figure 3

The activation function maps the input between a set of values indicating the amount of activation in the neuron. For example in a binary classification, the input can be mapped between 0 to 1 with 0.5 being the dividing point between classes. Activation functions can be broken down into two types: Linear Activation Functions & Non-Linear Activation functions. ReLu is an example of a linear activation function.

The weight is the network's control mechanism and by adjusting their values it can control the data's path through the neurons to the output classes.

The following sudo code is executed to adjust the weights. [8]

1) The weights are randomly selected at the creation of the network.
2) Take a batch of training samples and feed them through the network.
3) Run the model on the training samples to calculate the predictions.
4) Calculate the loss of the model by measuring the distance between the training prediction and the ground truth.
5) Compute the gradient of the loss measured by the model's parameters
6) Move the parameters in the opposite direction of the gradient.

The procedure is performed over and over until training is complete.

Approach and Implementation

I began my search for a dataset that met my previous specified requirements (medium sized dataset, not pre-processed, and no large studies) and through the magic of Google found the Stanford Cars 196 dataset. The dataset contains 16,185 images of cars split between 196 classes at the level of *Make, Model, Year* (2012 Tesla Model S). The data is split between 8,144 training images and 8,041 test images. [7]

I started by investigating the format and data types of the files in the dataset. The images are all in the Jpeg format and are not consistently sized. The bounding box and label files were both in the MAT (MatLab) format.

I choose Python, Tensorflow, and Visual Studio Code as my coding language, ml library, and development IDE.

I pre-processed the data using the following sudo code:

```
1) Read & store image files, bounding boxes, and labels.
2) Crop image files to bounding boxes. (x1, y1,x2,y2 format)
3) Combine cropped images and labels into a TFDataset (TensorFlow Dataset).
    a. Labels[0] corresponds to Images[0]
```

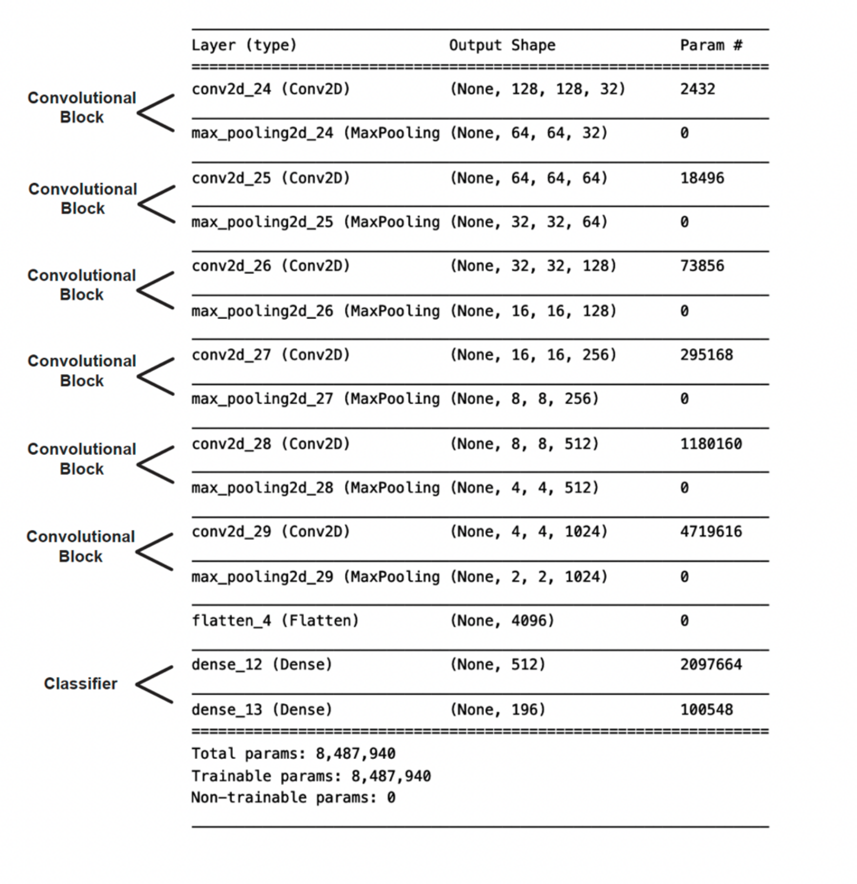
The TFDataset was then split into 80% training, 10% validation, and 10% testing each a separate TFDataset.

For initial testing I constructed a CNN following the steps on Kaggle's Computer Vision tutorial. [3] This tutorial uses a simplified version of the Stanford dataset which only contain two classes, car or truck. I quickly had the network with 80% accuracy.

My next step was to bootstrap a classifier with 196 classes on top of the CNN. My network contained the following structure:

- Six Convolutional blocks starting with 32 filters and doubling up to 1024 (32, 64, 128, 256, 512, 1024).
- Each block contains a Max Pooling layer which performs a down sampling of the data by taking the maximum value in a set of numbers. The layer has a default (2,2) input frame with a stride of 2.

  MaxPooling([5,10,2,3]) outputs 10
- The final block is then flattened into a 1D array.
- One Dense layer with 512 neurons and a ReLu activation.
- One Dense layer with 196 neurons with a SoftMax activation.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 128, 128, 32)      2432
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 512)         1180160
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 4, 4, 1024)        4719616
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 2, 2, 1024)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_12 (Dense)             (None, 512)               2097664
_________________________________________________________________
dense_13 (Dense)             (None, 196)               100548
=================================================================
Total params: 8,487,940
Trainable params: 8,487,940
Non-trainable params: 0
_________________________________________________________________
```

Figure 4 (TensorFlow Summary)

This network had an accuracy of 30% accuracy 4+ hours to train on my laptop.

At this point I moved all my training to the cloud. I selected AWS Sagemaker as my cloud development platform. This gave me access to instances containing multi CPU's and GPU's which greatly improved training, from 4+ hours to under 30 minutes. I mainly used the m1.p3.8xlarge instance which contains 32 vCPUs and 4 GPUs with 244 GB of memory.

For Distrusted computing I choose the Horovod framework. [9]. Developed by Uber Horovod allows you to spread training over multiple GPU's in parallel. It uses the All-Reduce framework.

"All-Reduce framework assumes communication between mappers as well, while the reducers are not used. In particular, each mapper maintains a copy of the global model. Then, when computing the update step for the current model, partial update step computed on one mapper is communicated to all other mappers. Once every mapper receives a message from all other mappers, the aggregated update step is performed on all computation nodes simultaneously, each updating their local model copy with exactly the same update step." [9]

All experimentation and testing from this point used this setup.

To book strap feature extraction a pretrained base was added to my network in place of my custom convolution network. I tested Inception, VGG16, and VGG19. VGG19 had the highest initial accuracy and was chosen for all further testing.

Experiment Results and Discussion

All experimental results will have the following format.

1) Description of network
2) Results
3) Discussion and improvements to be implemented.

Base:

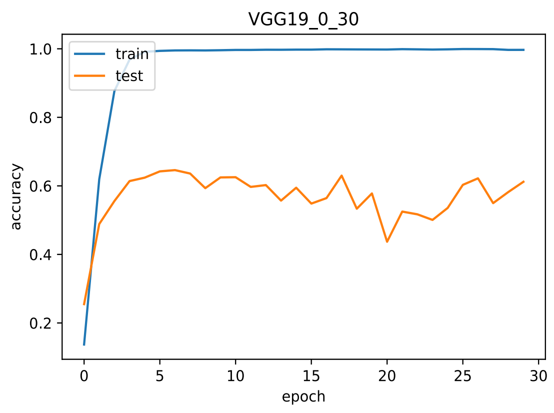
VGG16 Pretrained Base

Head:

512 Dense Unit with a ReLu activation.

196 Dense Unit with a SofMax activation.

Results: 61.2% accuracy



Base: VGG19
Train: 80%
Test: 20%
Head: 512

The Model is overfitting to the training data. It needs to by generalized to learn more of the signal and less of the noise. The next model should add Dropout to randomly remove neural connections and increase noise within training. This noise filters out features that are only prominent in the training data. Also add Batch Normalization within the network to reduce the impact of any features with an outsized activation. The combination of these two layers performs a filtering of both the low activation and high activation features leaving a more generalized model. Also, increase the size of the classification network.

Base:

VGG16

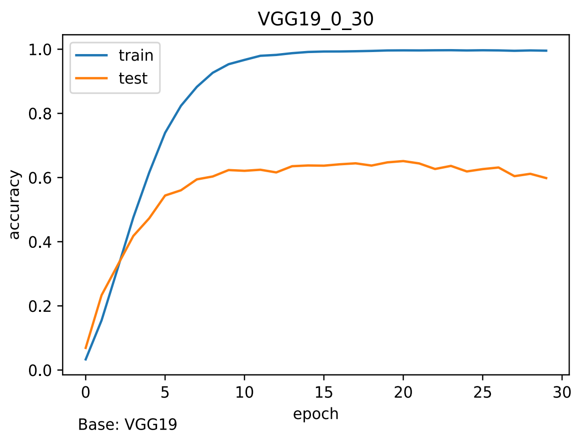
Head:

1024 – Dropout 30% - Batch Norm - ReLu

512 – Dropout 30% - Batch Norm - ReLu

256 – Dropout 30% - Batch Norm – ReLu

196 – Dropout 30% - Batch Norm - Softmax



Base: VGG19
Train: 80%
Test: 20%
Head: 1024 -> 256, Batch Normalization, Dropout 0.3

Results: 65.1% accuracy

The model is still overfitting to the training data but it took over 10 epochs to reach ~100% train accuracy. The batch normalization and dropout are helping to generalize the model. Accuracy has not increased very much though. The network needs more training data and the structure needs to be improved.

The dataset has be finite amount of images so I used a technique called Data Augmentation to transform the image files to create more versions of each car. I selected horizontal flip, random rotation from -10% to 10%, and random contrast from -20% to 20%. I also will increase the depth of the classifier head.

Base:

VGG16

Head:

4096 – Dropout 30% - Batch Norm - ReLu

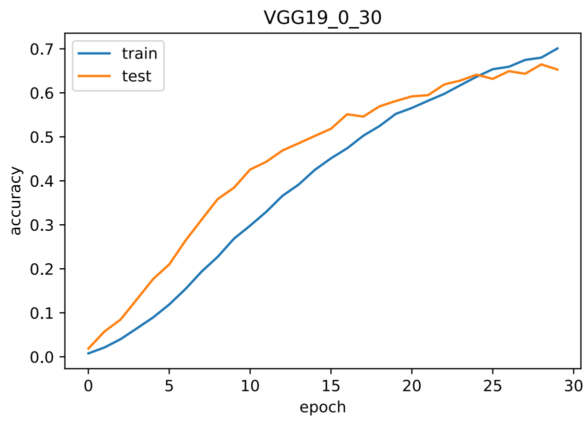
2048 – Dropout 30% - Batch Norm - ReLu

1024 – Dropout 30% - Batch Norm - ReLu

512 – Dropout 30% - Batch Norm - ReLu

256 – Dropout 30% - Batch Norm – ReLu

196 – Dropout 30% - Batch Norm – Softmax



Base: VGG19
Train: 80%
Test: 20%
Head: 4096 -> 256, Batch Normalization, Dropout 0.3
Data Aug: Flip Horizontal, Rotation 0.1, Contrast: 0.2

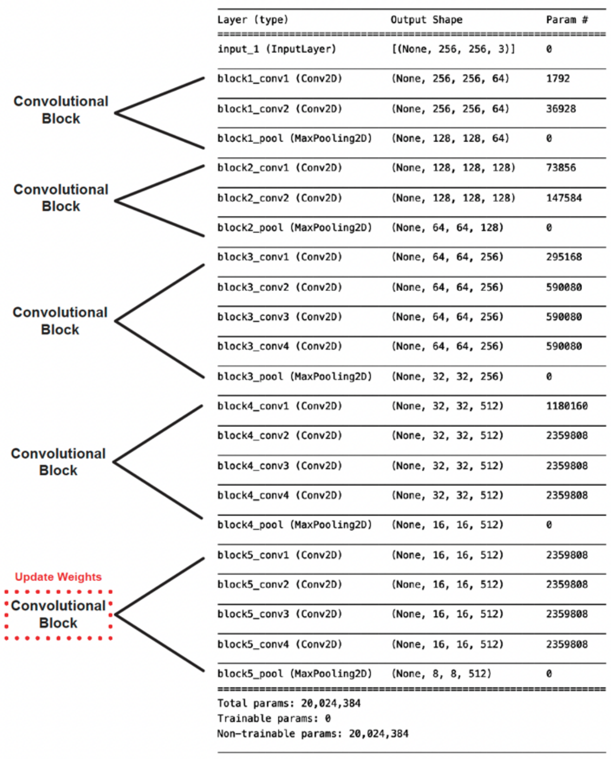
Data Augmentation: Flip Horizontal, Rotation, Contrast

Results: 70.2% accuracy

The overfitting has been greatly reduced. The model actually started to under fit until around the 30-epoch mark. At this point the accuracy leveled off around 70% and continued at this level while the train accuracy went to 100%. The combination of the previous adjusts are allowing my network to learn a more generalized version of the input data.

Up to this point I have not touched the base of my network. Its weights have been locked and are not adjusted. In the next model I will unlock the 5th convolutional block of the VGG19 base and allow network to train these features. The 5th convolutional block is the deepest layer which contains the complex arrangements of the lower-level layer's features. This will allow the network to use those features as building blocks and train to the more complex shapes, textures, and colors or the car images.

A summary of the VGG19 network:


Convolutional Block
Convolutional Block
Convolutional Block
Convolutional Block
Update Weights
Convolutional Block

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 3)]     0
block1_conv1 (Conv2D)        (None, 256, 256, 64)      1792
block1_conv2 (Conv2D)        (None, 256, 256, 64)      36928
block1_pool (MaxPooling2D)   (None, 128, 128, 64)      0
block2_conv1 (Conv2D)        (None, 128, 128, 128)     73856
block2_conv2 (Conv2D)        (None, 128, 128, 128)     147584
block2_pool (MaxPooling2D)   (None, 64, 64, 128)       0
block3_conv1 (Conv2D)        (None, 64, 64, 256)       295168
block3_conv2 (Conv2D)        (None, 64, 64, 256)       590080
block3_conv3 (Conv2D)        (None, 64, 64, 256)       590080
block3_conv4 (Conv2D)        (None, 64, 64, 256)       590080
block3_pool (MaxPooling2D)   (None, 32, 32, 256)       0
block4_conv1 (Conv2D)        (None, 32, 32, 512)       1180160
block4_conv2 (Conv2D)        (None, 32, 32, 512)       2359808
block4_conv3 (Conv2D)        (None, 32, 32, 512)       2359808
block4_conv4 (Conv2D)        (None, 32, 32, 512)       2359808
block4_pool (MaxPooling2D)   (None, 16, 16, 512)       0
block5_conv1 (Conv2D)        (None, 16, 16, 512)       2359808
block5_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
block5_conv4 (Conv2D)        (None, 16, 16, 512)       2359808
block5_pool (MaxPooling2D)   (None, 8, 8, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
```

Figure 5

Base:

VGG16

5th Convolutional Block is now trainable.

Head:

4096 – Dropout 30% - Batch Norm - ReLu

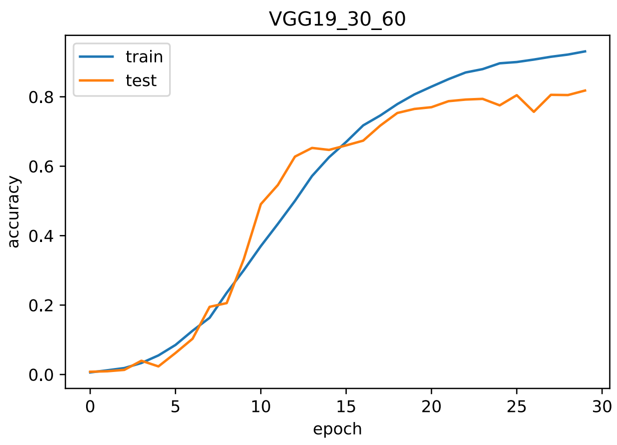
2048 – Dropout 30% - Batch Norm - ReLu

1024 – Dropout 30% - Batch Norm - ReLu

512 – Dropout 30% - Batch Norm - ReLu

256 – Dropout 30% - Batch Norm – ReLu

196 – Dropout 30% - Batch Norm – Softmax



Base: VGG19
Convd 5 trainable
Train: 80%
Test: 20%
Head: 4096 -> 256, Batch Normalization, Dropout: 0.25
Data Aug: Flip Horizontal, Rotation 0.1, Contrast: 0.2

Data Augmentation: Flip Horizontal, Rotation, Contrast

Results: 82.4% accuracy

Training the 5th convolutional block increased the training accuracy 12.2%. The network overfits to the training data after the 15th epoch which can be reduced with further tuning.

8. Conclusion

Reducing overfitting was the greatest challenge to the project. A combination of a larger classifier head, batch normalization, dropout, data augmentation, and training the 5th block of the base layer increased accuracy 21.2%. These parameters can be fine-tuned to further reduce overfitting and drive accuracy even higher.

The pipeline I created for this project proved itself to be stable and fast in development and training. The use of AWS Sagemaker with large training instances was extremely important in reducing the training time of my models. In the future I will start training in the cloud and circumvent local training all together.

I have begun further research into the computer vision literature for future convolutional neural network designs. This project has been a vital first step in my exploration of the subject and will be my base for all further designs.

8. References

[1] Wikimedia Foundation. (2021, September 17). *Convolutional neural network*. Wikipedia. Retrieved September 22, 2021, from https://en.wikipedia.org/wiki/Convolutional_neural_network#History.

[2] *Convolutional neural NETWORK (CNN) : TensorFlow Core*. TensorFlow. (n.d.). Retrieved September 22, 2021, from https://www.tensorflow.org/tutorials/images/cnn.

[3] Kaggle. (n.d.). *Learn computer Vision Tutorials*. Kaggle. Retrieved September 22, 2021, from https://www.kaggle.com/learn/computer-vision.

[4] Kaggle. (n.d.). *Learn intro to deep learning tutorials*. Kaggle. Retrieved September 22, 2021, from https://www.kaggle.com/learn/intro-to-deep-learning.

[5] Leung, J., & Chen, M. (2018). Image Recognition with MapReduce Based Convolutional Neural Networks. Retrieved from https://bbnewpaltz.sln.suny.edu/bbcswebdav/pid-3586884-dt-content-rid-16482488_2/courses/fall21_CPS593_04/CNN_MinChen_Final.pdf.

[6] Sun, Z., Ozay, M., & Okatani, T. (2016). Design of Kernels in Convolutional Neural Networks for Image Classification. Retrieved from https://sci-hub.ru/https://link.springer.com/chapter/10.1007/978-3-319-46478-7_4.

[7] Jonathan Krause, Michael Stark, Jia Deng, Li Fei-Fei(2013 3D Object Representations for Fine-Grained Categorization. *4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013* **(3dRR-13)**.

[8] Chollet, F. (2021). *Deep Learning with Python, Second Edition* (2nd ed.). 8

[9] Sergeev, A., Del Balso, M. (2018) *Horovod: fast and easy distributed deep l earning with Tensorflow. Frome* arXiv:1802.05799