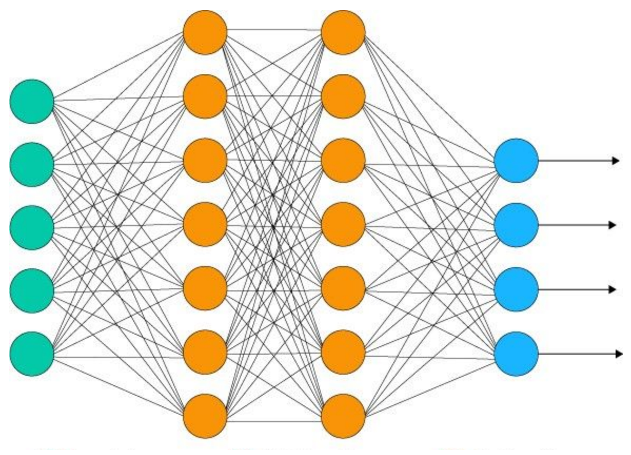
Neural Networks:
Project 3 Report



Kevin Cheong, Haziq Saeed
Professor Gordon, S.
California State University, Sacramento

CSC 180-01 Fall 2017
16 November 2017

Introduction

Convolutional neural network (CNN) development includes the process of teaching a machine through the use of artificial neural networks. CNNs train on a set of sample data for a period of time, and are tested in their accuracy with separate testing data. The advancements in neural networks has had a great impact on society, especially in fields concerning artificial intelligence.

By applying understood concepts surrounding the human brain, engineers have been able to develop neural networks that can perform certain tasks more efficiently or economically. Many of these tasks include image recognition, such as telling the difference between pictures of humans and other animals. The applications of neural networks are endless, having already been applied in many fields (self-driving cars for example). As society discovers new information concerning how the human brain function, these new discoveries could be applied to neural network developments; perhaps neural networks may one day be able to develop a clearly observable sense of self-awareness, or other animal-like characteristics.

Part I: Average and Median Identifying Neural Network

Problem Statement

3 decimal numbers (each between 1 and 50 inclusive) are given to a simple neural network; the average and median are expected as outputs. The input values or provided in unsorted order, which intrinsically makes identifying the median values more difficult for the machine to learn.

Neural Network Architecture Decisions

After several trials, our best results were produced by the following parameters:

Training Criteria:	0.4
Testing Criteria:	0.7
Learning Rate:	0.8
Momentum:	0.16
Bias:	-1.0
Random Weight Max:	0.5
Leaky RelU Amount:	0.1

Column	Number of Nodes	Activation Function
Input Layer	4	N/A
2	128	ReLU
3	64	ReLU
4	16	Leaky ReLU
5	8	Leaky ReLU
Output Layer	2	Sigmoid

The Number of Nodes column includes the bias nodes in the count.

All trials were conducted using 49 training sets, and 23 validation sets. Though we provided a maximum limit of 2,000,000 iterations of training, though our tests seldom required so many.

Training Time

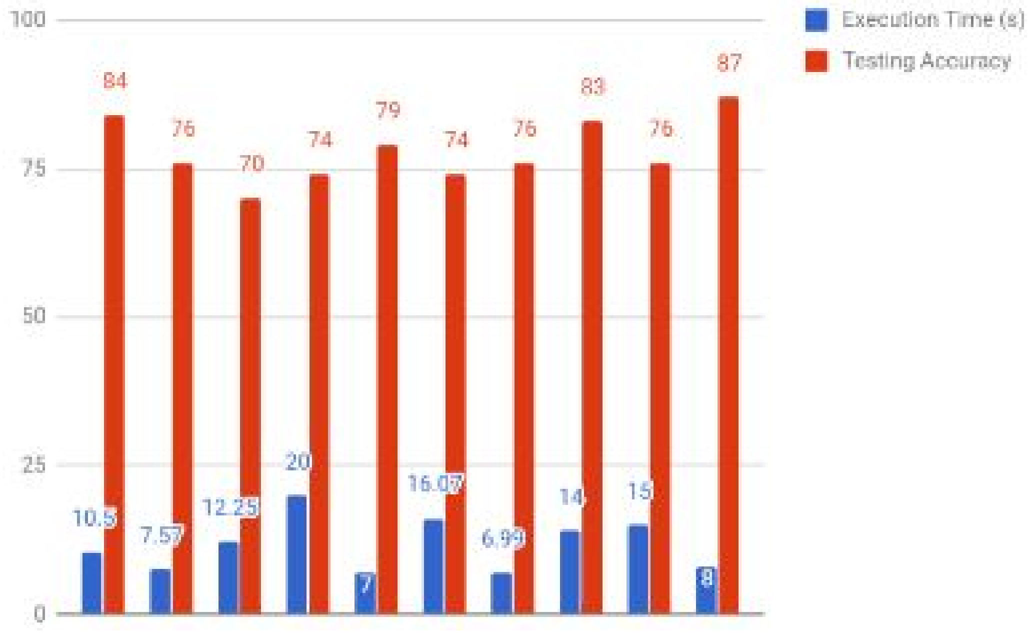
Various training times were produced every single time the same execution was performed on the learning set. The process of capturing the training time was done through the use of a stopwatch, which introduced some minor errors (our reaction speed when starting or stopping the stopwatch).

In order to be sure of our testing results, we opted to run the same neural network multiple times to determine the approximate training time and validation accuracies.

The neural network was able to finish training at a median level of X_{med} = 12.25s, with an average of X_{avg} = 11.80s.

The increase in speed was immediately noticeable after changing activation layers 2 and 3 to ReLU, and activation layers 4 and 5 to Leaky ReLU. The main benefit of ReLU is the linear descent of this activation layer, accelerating the rate of the gradient descent in the formula: $f(x) = max(0,x)$. The issue that is known regarding ReLU is that it can shut down most of the neural network, if the learning rate is set too high. This can be counterbalanced, using Leaky ReLU where it will have a small negative slope, allowing killed neurons a chance to be revived.

Once training began, the neural network would slowly proceed through the forward and backwards passes until more neurons began dying off, allowing training to speed up as it went along. This implies that there may be too many nodes at the start. Accuracy did not seem to correlate with training times in any meaningful way.



The execution times in seconds (blue), and the validation accuracies in a percent (red). Training tended to take between 10-15 seconds, while the accuracy levels hovered around 75-80%.

Generalization

The neural network appeared to generalize fairly well in regards to its fairly high validation accuracy.

The first initial test, did produce a higher accuracy than expected, but gradually declined with each execution towards a settling point. The highest accuracy it produced during 1 execution is 87%.

Learning

In general system, did seem to learn the problem for the most part. Many of the tests fell within the testing criteria, but the neural network has had consistent difficulty with certain validation pairs such as (3, 3, 3, 3, 3) despite there being several training cases similar to it.

Part II: Separating the Average and Median Problems

Problem Statement

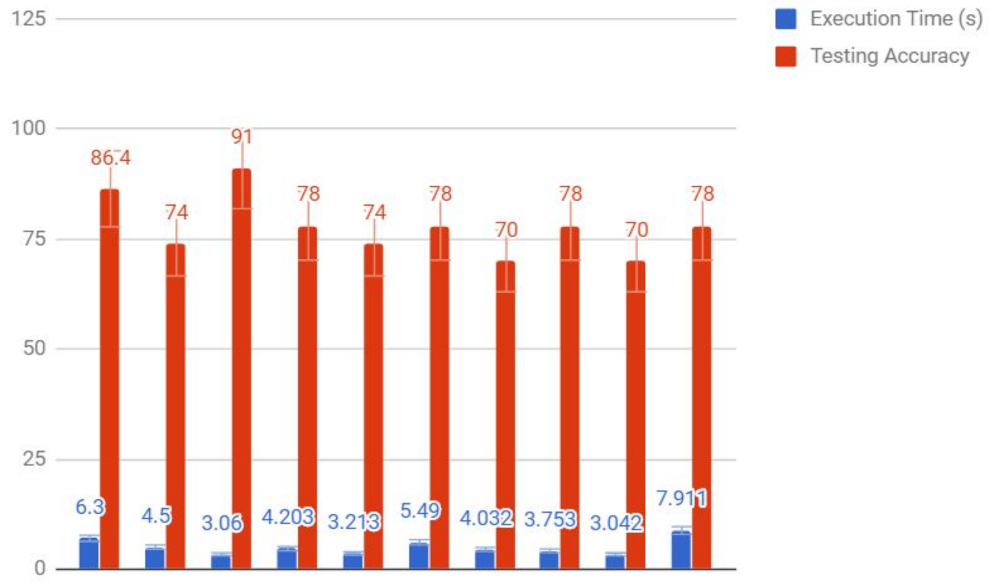
Similarly to part I, 3 decimals are provided as inputs, but the average and median are worked on as independent problems with their own private neural networks.

In order to foster an environment to make logical comparisons to the results generated in part 1, the parameters were kept as similar as possible.

Since averaging only requires an algebraic formula to compute, the neural network should be able to learn how to do it accurately. The median, however, is likely to be much harder to train due to the amount of noise that is introduced with unordered inputs, and is likely to remain with lower accuracy ratings.

Training Times

With problems separated, both training speeds were expected to increase, but not necessarily by the same amounts.



Part III: Convolutional Neural Network (CNN) Image Classifier

Problem Statement

Our CNN attempts to differentiate between images containing four different kinds of fruit: Avocados, lemons, limes, and oranges. These four fruits were chosen due to their high similarities to each other. For example, lemons and limes tend to have the same shape, but have differing colors. Another example is the lighting of the fruits; an orange may appear brightly in one image while it could be darker in another. The CNN will be trained and tested to see if it can recognize the fruits as well as overcome the various sources of noise.

Starting from the top left to bottom right: Whole avocado, arranged lemons, pile of lime slices, and oranges on the vine.

Testing and Training Data Sets

Hypothetically speaking, it would be unreasonable to force users of this CNN to format their images in such a way as to clearly depict the fruit with no noise. The training and test images were selected to involve both clear images as well as images with several sources of noise (e.g. number of fruits, angles, lighting, slicings, tableware, etc.).

16 training and 4 validation images were prepared for each fruit (totaling 80 128x128 images total). With this highly limited data set, the validation images selected from each respective fruit's set of 20 were chosen based on subjective opinion of how well they represented their populations as a whole. This method of validation image selection was done to give the CNN a fair chance of learning the problem with such limited and noisy data.

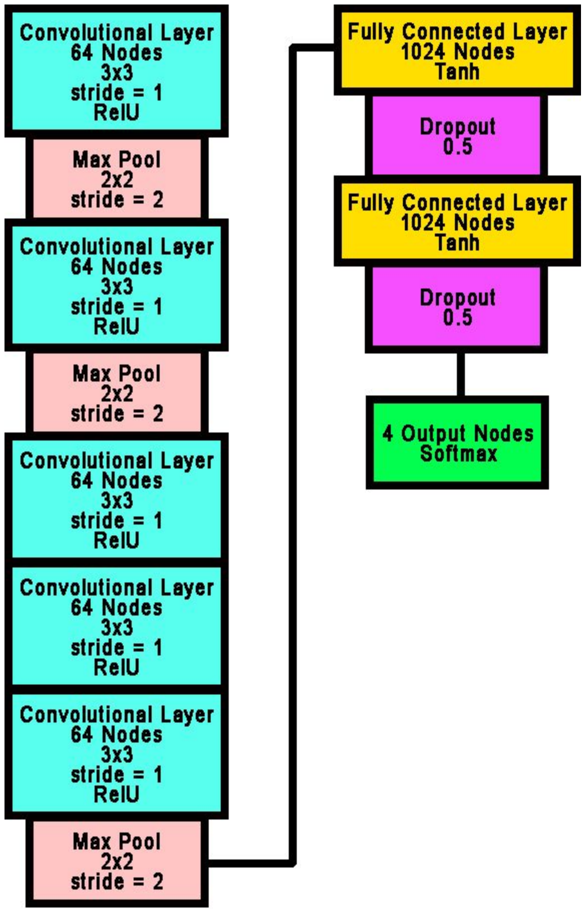
CNN Architecture Decisions

Several iterations through architectures based on AlexNet, VGG-Net, E-Net, and custom designs have been performed, and AlexNet seemed to be the superior choice until E-Net began competing.

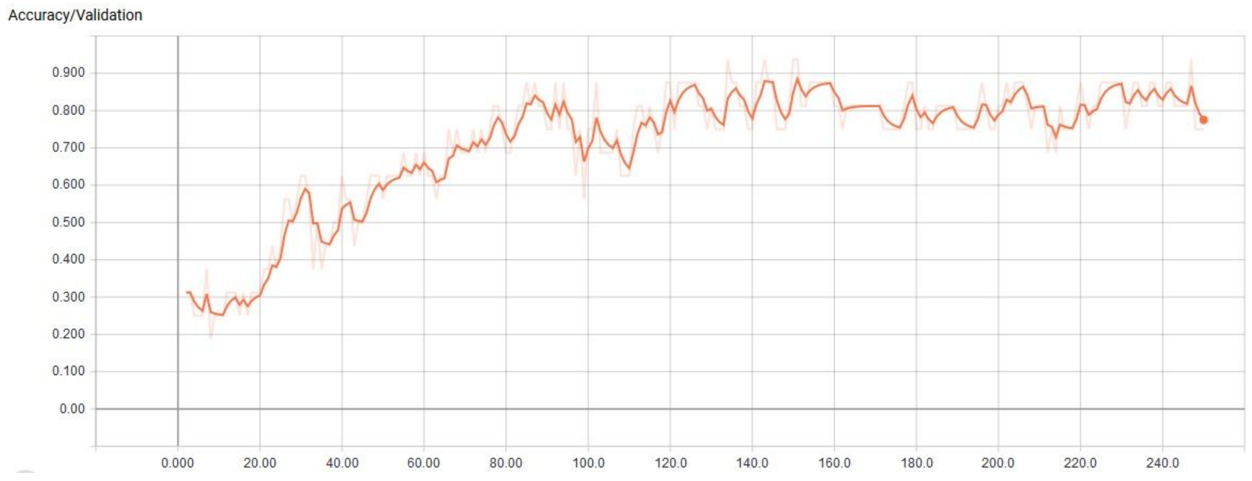
AlexNet Starting Trial

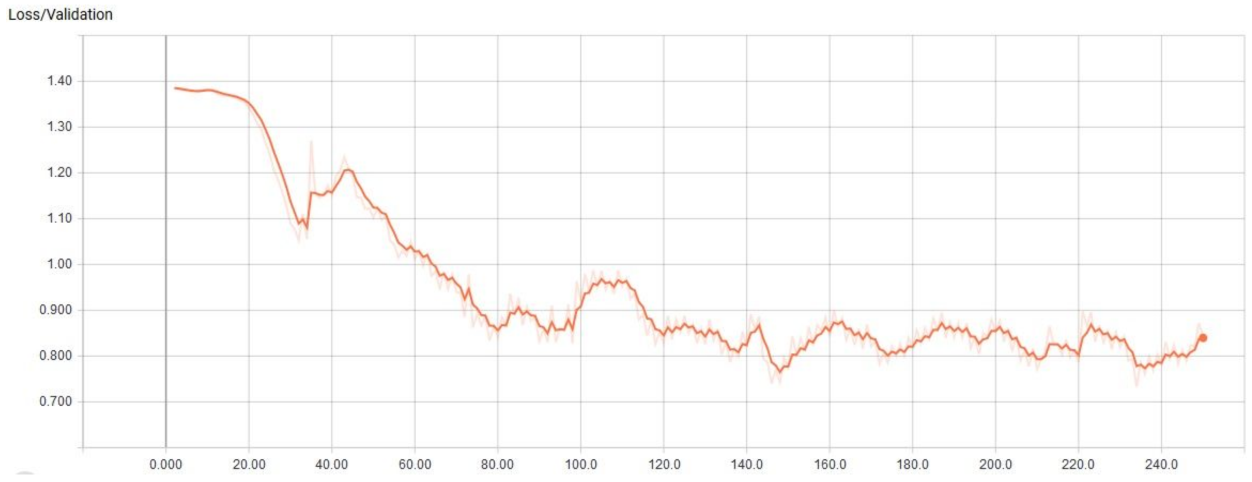
The provided *deepneuralnet.py* file was altered to accommodate 128x128 sized images, and the learning rate was set to 0.01 (as opposed to 0.001).

For this baseline trial, 1 avocado, 1 lemon, and 1 lime were missed. This resulted in 13 out of 16 validation images being correctly identified (81.25% accuracy). The CNN missed the avocado due to a 3-way tie between avocado, lemon, and lime (at roughly 0.3 each); the lemon was missed due to it being mistaken as a lime, and the lime was missed due to it being mistaken as an avocado (though it roughly tied with lime at around 0.6).

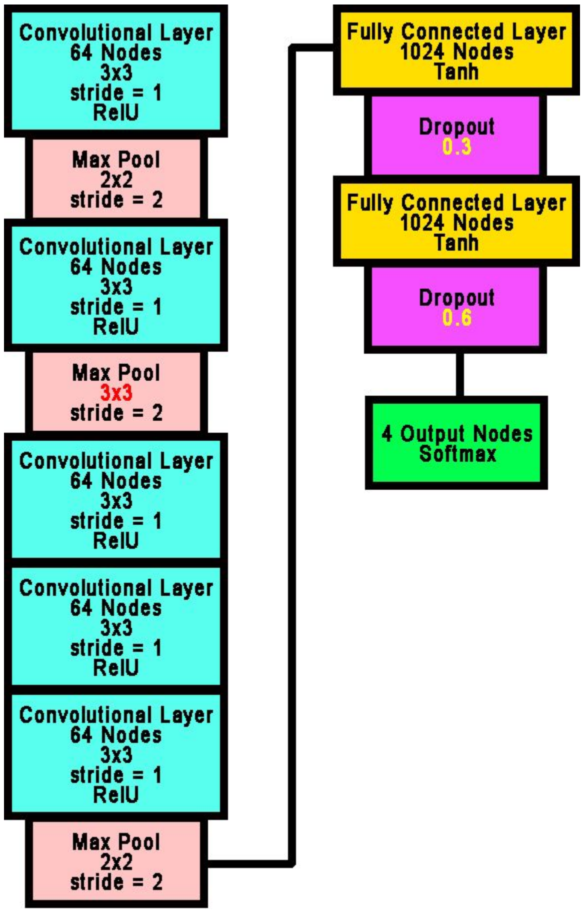

Convolutional Layer 64 Nodes 3x3 stride = 1 RelU
Max Pool 2x2 stride = 2
Convolutional Layer 64 Nodes 3x3 stride = 1 RelU
Max Pool 2x2 stride = 2
Convolutional Layer 64 Nodes 3x3 stride = 1 RelU
Convolutional Layer 64 Nodes 3x3 stride = 1 RelU
Convolutional Layer 64 Nodes 3x3 stride = 1 RelU
Max Pool 2x2 stride = 2
Fully Connected Layer 1024 Nodes Tanh
Dropout 0.5
Fully Connected Layer 1024 Nodes Tanh
Dropout 0.5
4 Output Nodes Softmax

The provided AlexNet-styled architecture.





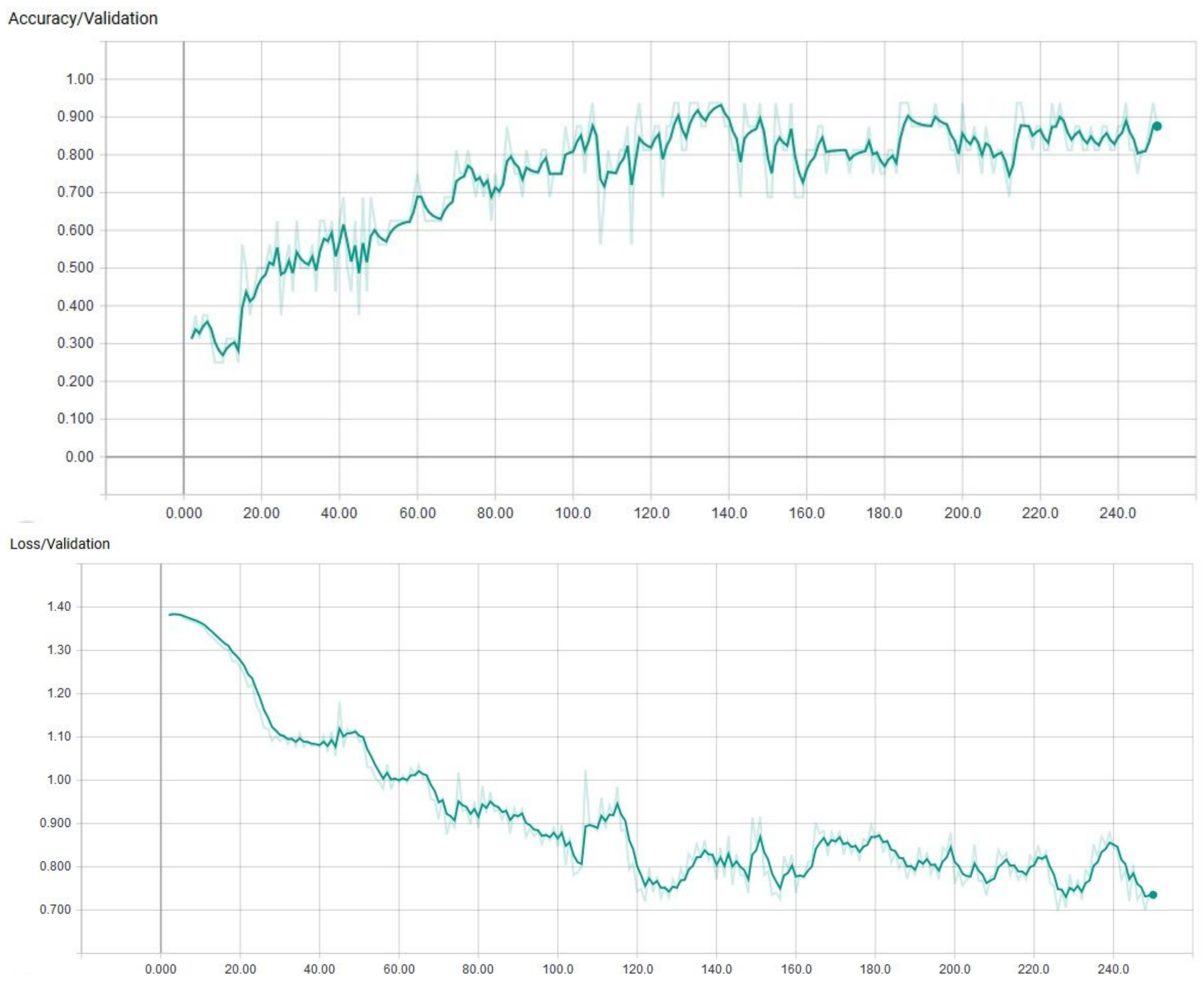
The starting AlexNet CNN was allowed to train for 250 epochs at a learning rate of 0.01, approximately resulting in 85% validation accuracy and 0.8 loss.



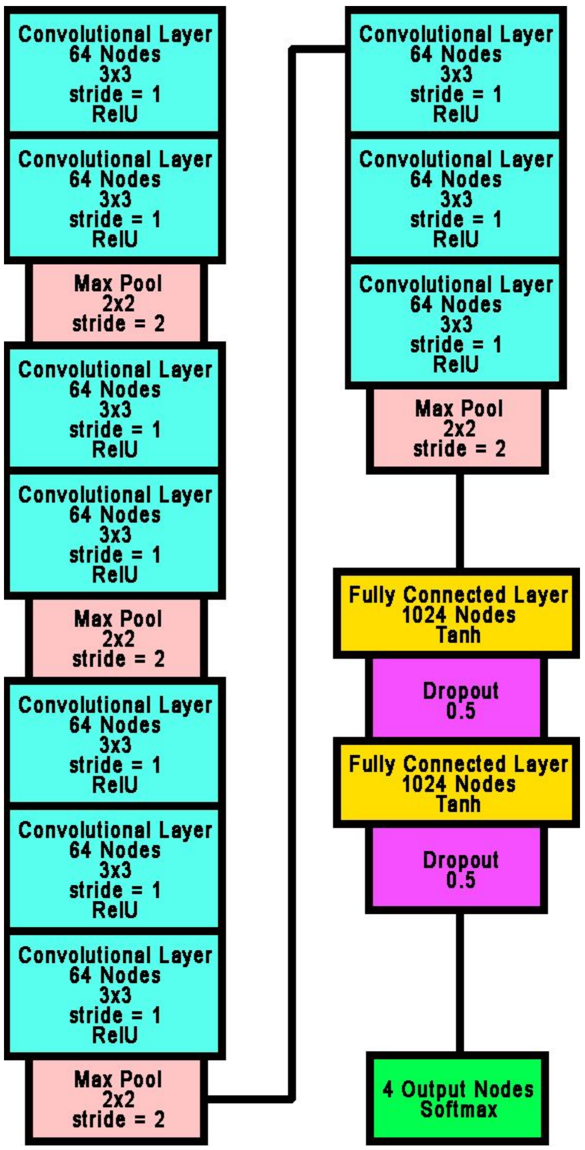
Based on the starting CNN; the primary changes are located in the second pooling layer (changing 2x2 to 3x3), and changing the first and second dropout rates (from 0.5 to 0.3 and 0.6 respectively).

AlexNet Best Trial
The best trial was performed after 3 slight modifications were implemented. In the second pooling layer's grid was changed from 2x2 to 3x3. This was done because the provided notes had a 3x3 grid in the diagram. Also, the first and second dropouts were changed to 0.3 and 0.6. The dropouts were modified to encourage retaining more of the neurons earlier in the pipeline without dropping significantly less neurons overall.

Our best AlexNet trial missed validating 2 lemons, resulting in 14 out of 16 correct identifications (81.25%) accuracy. Though the changes and results appear similar to the provided AlexNet code, it should be noted that at the validation accuracy breached the 90% mark during training (something the starting AlexNet script failed to do). It should also be noted, that the average loss appeared to improve by about 0.05 units. After several tests, we were unable to design AlexNet CNNs that could miss 1 or less images during validation.



The best AlexNet CNN was allowed to train for 250 epochs at a learning rate of 0.01, approximately resulting in 85-90%% validation accuracy and 0.75 loss. This is a 0-5% improvement in accuracy and 0.05 improvement in loss compared to the provided AlexNet CNN.

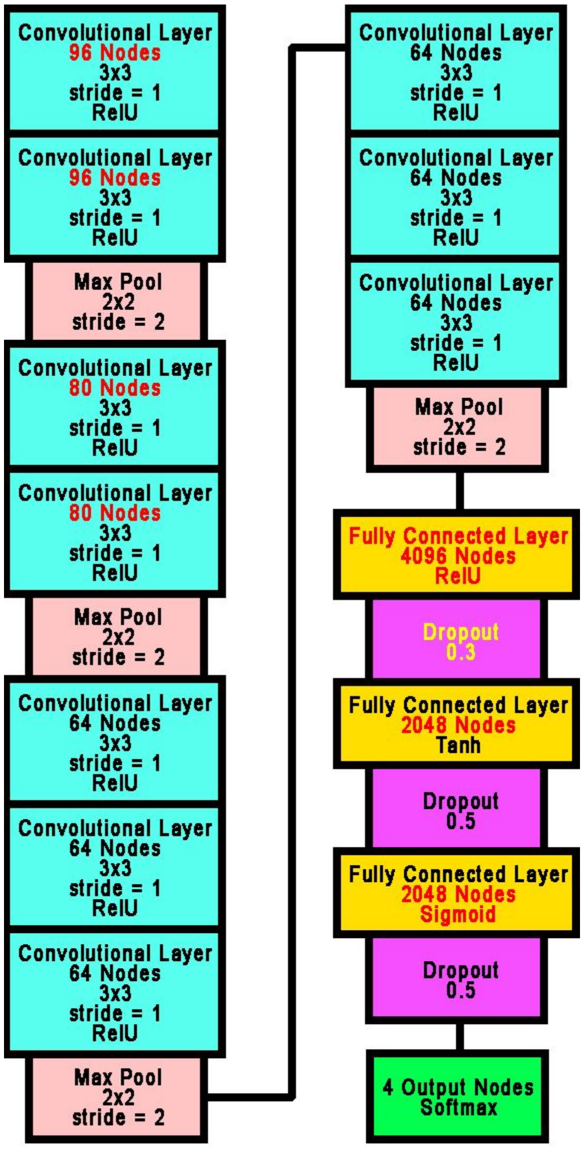


The experimental VGG-Net-styled CNN trained for 300 epochs at a learning rate of 0.013. It ultimately failed to learn the problem, due to it having almost equal scores for all 4 fruits assigned to nearly all of the validation images.

VGG-Net Trials

Two VGG-Net type architectures were explored, but ended as incomplete trials.

The first trial was simply a failure for not proving its ability to learn.
After 3.5 hours, 400 epochs, and a 25% accuracy rating throughout (implying that the CNN was essentially randomly guessing what the validation images were), we discovered that VGG-Net architectures are geared towards big-data ventures. This means they have a tendency to require extensive amounts of time to train, and plenty of training and validation sets to work with (as opposed to our micro-problem's context).



Another experimental VGG-Net-styled CNN trained for 1780 epochs at a learning rate of 0.016. It ultimately failed to learn the problem, due to it having a running validation accuracy of 25% before hanging (never completed training past epoch number 1780).

We certainly do not have the capability of providing hundreds, let alone millions, of images. Nonetheless, we persevered with testing. Our first change was to multiply the training epochs by 10 (totaling 4000 epochs). We also incremented the learning rate from 0.013 to 0.016, and modified the architecture to accommodate more neurons.

Unfortunately, copious amounts of time and resources were ultimately wasted pursuing VGG-Net style coding. The final trial trained for approximately 6 hours before hanging at epoch number 1780, and refused to continue when left alone for the next 8 hours.

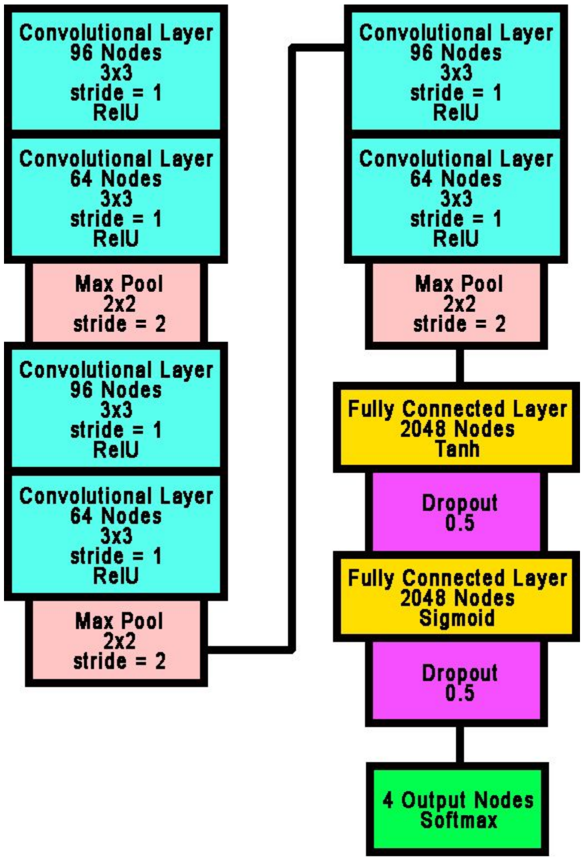
The Tensorflow charts were deemed worthless, as they appeared to be graphically similar to constants. The accuracies seemed to stay at 25%, and the loss stayed at about 1.8 to 2. We decided to cut our losses, and discontinued our research of VGG-Net architectures at this point.

E-Net Starting Trial
After the inconclusive results of VGG-Net, we transferred our focus towards attempting an E-Net-styled architecture.

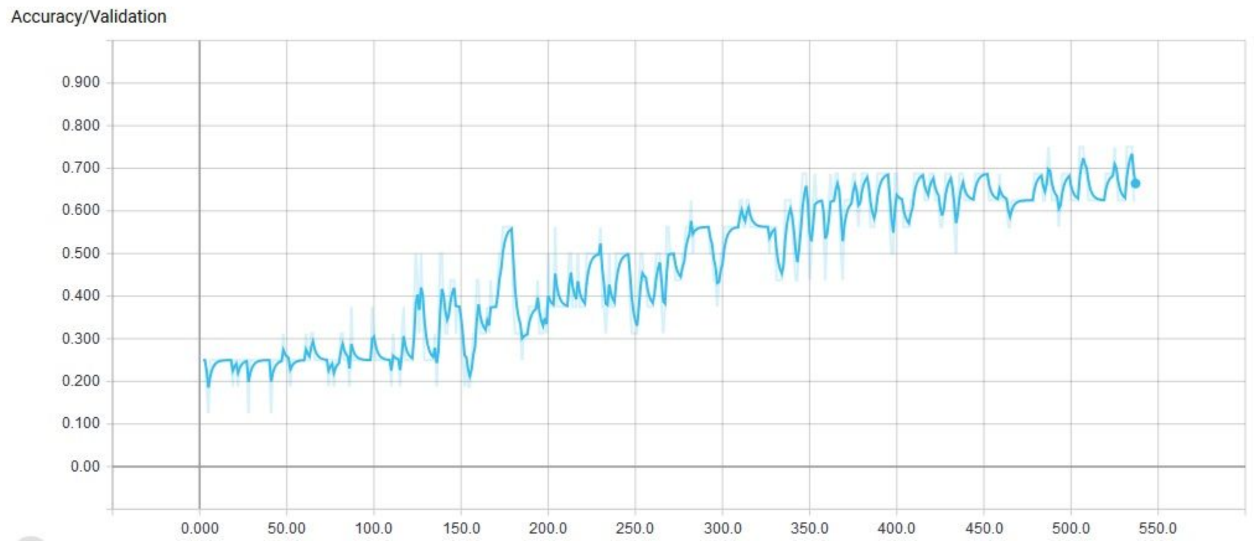
The inspiration to pursue this architecture came from researching an article written by Mariya Yao, who provided us knowledge about 14 well-known architectures. E-Net appeared to complement small data sets with high accuracies and fast training times (relatively speaking), and the components included layers we were familiarized with in class.

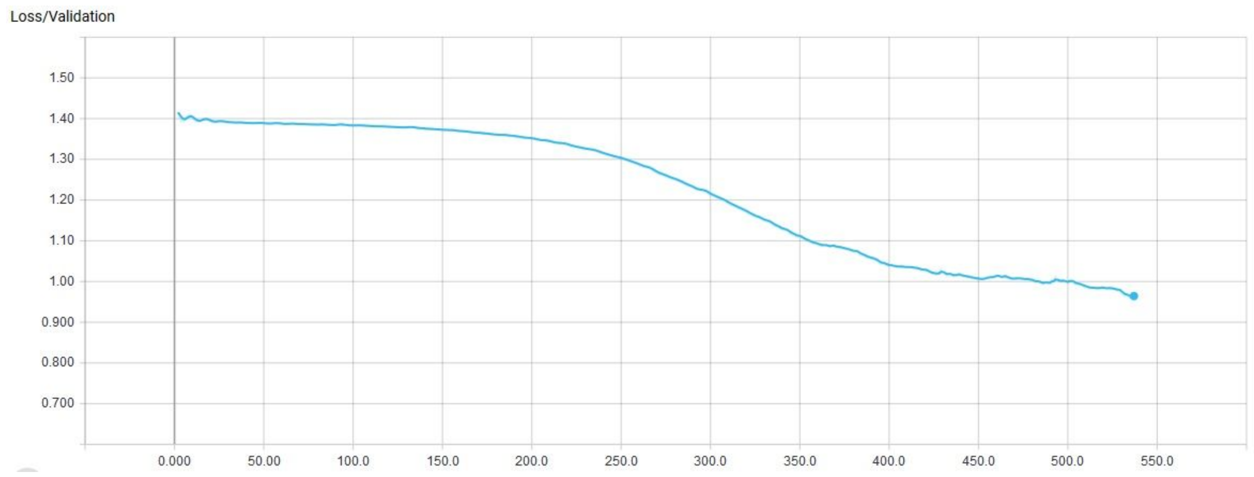
Our E-Net design features simple repetitive layers, and is visually similar to an example CNN displayed in Andrej Karpathy's Stanford Lecture.

The E-Net trial appeared to slowly mimic the efficiency experienced when using AlexNet styled architecture. The slow speed is likely due to the significant change in the learning rate from 0.01 to 0.001, but the trial proved inconclusive due to another unfortunate training glitch.



An E-Net inspired architecture. This architecture trained for 600 epochs at a learning rate of 0.001.



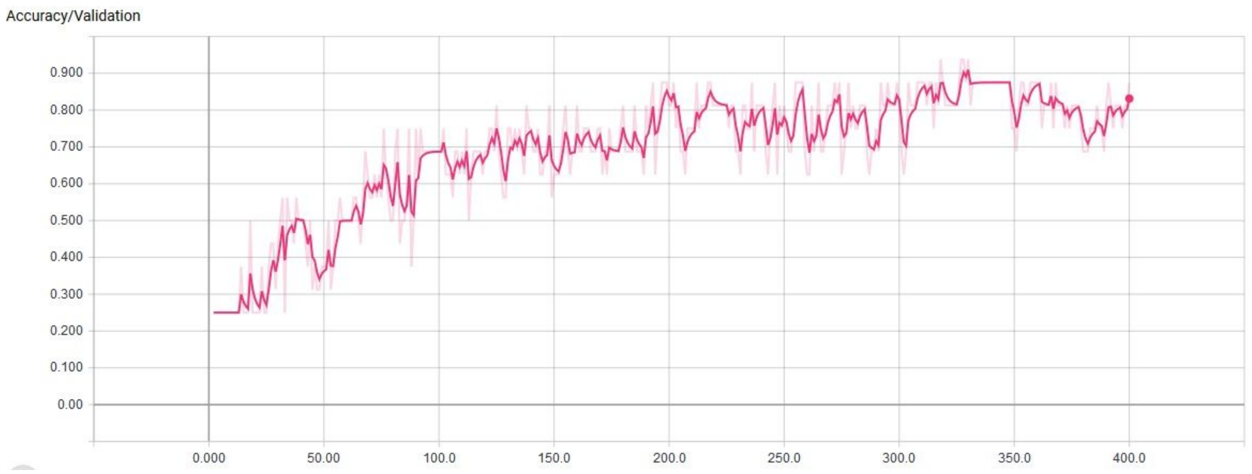


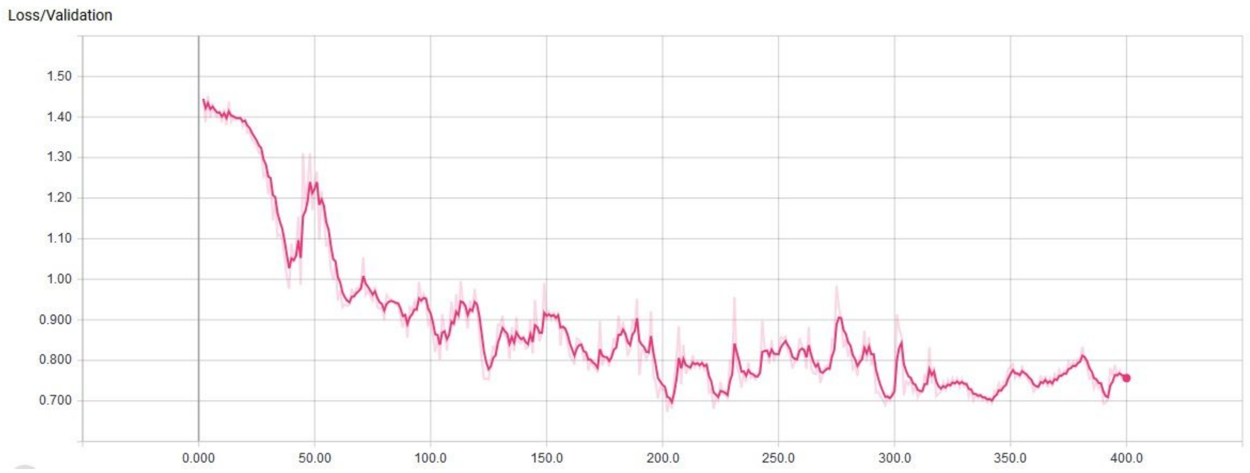
The first E-Net styled CNN was also allowed to train for 600 epochs at a learning rate of 0.001. It hanged at epoch 536 with a validation accuracy of 62.5%, and refused to finish training. The slopes of the graph were taken into consideration whilst preparing for the next E-Net trial (They did not appear to be totally flattening out just yet), motivating us to not give up on it.

E-Net Re-Trial
After analyzing the partial data from our failed E-Net run, some parameters were preemptively altered to (hopefully) secure a better run.

The performed modification was extremely simple: We changed the learning rate to 0.013 to match the AlexNet trials. Since E-Net with a learning rate of 0.001 just seemed like a slower version of AlexNet, we thought simply speeding it up was a reasonable, minimalist, and significant modification to test.

Our E-Net architecture seemed to provide better training results than AlexNet overall. Though it did not strike high validations and low losses as quickly as AlexNet, it managed to maintain positive sloped graphs all they way to its epoch limit. In our previous trials, AlexNet's general learning curve somewhat plateaus over time. This leads us to believe that our E-Net CNN may be superior to our AlexNet CNN, if proper tweaks are implemented.





E-Net trained after 400 epochs at a learning rate of 0.013. Missed 1 avocado (identified as orange) and 1 lemon (identified as lime). The validation accuracy seemed to be between 75-85%, and the loss stayed around 0.75.

Final Results

Training Time

Overall, an E-Net-styled architecture provided the most promising results. Our best run had a running time of 1 hour and 42 minutes.

Validation Accuracies

Validation Images	Percentages			
	Av	Le	Li	Or
Avocado 1	63	17	46	15
Avocado 2	23	23	12	41
Avocado 3	32	20	24	25
Avocado 4	43	23	21	14
Lemon 1	12	18	47	22
Lemon 2	16	47	11	27
Lemon 3	16	57	08	20
Lemon 4	15	59	09	18
Lime 1	13	10	62	15
Lime 2	23	19	38	20
Lime 3	12	11	59	18
Lime 4	13	10	60	16
Orange 1	15	13	11	61
Orange 2	16	24	10	50
Orange 3	10	9	10	71
Orange 4	15	12	11	63

Correct
Near Mistake (Within 10%)
Incorrect

Between E-Net and AlexNet, both have provided cases of identifying 14 out of 16 validation images correctly, but E-Net was able to do so with much more distinct activation percentages (less near-ties). This leads us to believe that our E-Net CNN yields the best results for this particular problem.

Learning

Though 2 fruits were misidentified, it was not due to the result of close scores. The CNN definitively thought each image was a specific fruit, showing that it had tightened its learning criteria (but not with perfect accuracy).

On the other hand, *learning* can be interpreted from multiple perspectives. For one, the CNN was not able to learn the problem in a general sense due to the limited amount of training and validation data. In this specific environment, the CNN was able to solve this particular micro-problem well, and demonstrates further potential to perform better if it could be provided with more time to run experiments.

Best CNN Python Script

The following script *deepneuralnet.py* is responsible for the best CNN trial we conducted.

```
import tflearn

from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression
from tflearn.metrics import Accuracy

acc = Accuracy()
network = input_data(shape=[None, 128, 128, 3])

# Conv layers ------------------------------------
network = conv_2d(network, 96, 3, strides=1, activation='relu')
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 96, 3, strides=1, activation='relu')
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 96, 3, strides=1, activation='relu')
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)

# Fully Connected Layers -------------------------
network = fully_connected(network, 2048, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 2048, activation='sigmoid')
network = dropout(network, 0.5)
network = fully_connected(network, 4, activation='softmax')

network = regression(network, optimizer='momentum',
loss='categorical_crossentropy', learning_rate=0.014, metric=acc)
model = tflearn.DNN(network, tensorboard_verbose=3,
tensorboard_dir="logs")
```

References

Avocado. 2017, https://www.ldoceonline.com/dictionary/avocado. JPEG File.

Karpathy, A. "CS231n Lecture 7 - Convolutional Neural Networks." *YouTube*, uploaded by MachineLearner, 14 June 2016, https://www.youtube.com/watch?v=GYGYnspV230.

Naranjas y Azahar. 2008, https://www.toprural.com/info/turismo-rural/floracion/flor-de-azahar. JPEG File.

Staroseltsev, Alex. *Lime Background*. 2017, https://www.123rf.com/photo_8066584_lime-background.html. JPEG File.

Victoriya89. *Fresh Lemons with Leaves*. 28 Oct. 2015, https://www.istockphoto.com/photo/fresh-lemons-with-leaves-gm530919415-54934992?esource=SEO_GIS_CDN_Redirect. JPEG File.

XenonStack. *Artificial Neural Network*. July 2016, https://hackernoon.com/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7. JPEG File.

Yao, M. "14 Design Patterns to Improve your Convolutional Neural Networks." *TopBots*, 22 Mar. 2017, https://www.topbots.com/14-design-patterns-improve-convolutional-neural-network-cnn-architecture.