Part One

The setting that I used for my neural network consisted of a learning rate of 0.30, momentum = 0.90, 2 hidden layers both 10 nodes each with activation function leakyRelu on both hidden layers, and sig on the output layer. Training criteria within 0.01, and testing criteria within 0.02. Bias = -1 and weightInit = 1.00. MaxIterations = 1 million.

Results: The program failed to train the criteria. However, the program still learned, just not well enough to pass. Here are my results;

```
Sum Squared Error for Training cases   = 0.0001
% of Training Cases that meet criteria = 0.6167

Sum Squared Error for Testing cases   = 0.0409
% of Testing Cases that meet criteria = 0.1000
```

The program ran the full 1 million iterations before terminating.

Overall, it appears that it was able to generalize the problem somewhat. About 62% of the training cases met the criteria.

Looking at the test cases we see that 10% of them passed. So, I don't think that it learned as well as it could have, but it still learned. Maybe if I had used more training cases I could have gotten better results. This, considering there are 3 outputs for this program, solving 3 different problems. However, the program did still learn.

I think the hardest problem for the neural network to solve was finding the minimum. The reason being is that in order to compute the minimum, it must take the weighted sums of both inputs and do calculation on them to determine the smaller of the two. This would take a good amount of time to compute. Compare this to the average, which would require subtraction and a single division to compute. As well as the input subtraction, which takes the number of the left and subtracts it from the one on the right. So, the minimum function would take the longest in my opinion due to the extra work needed to compute.

Part Two

1. For the problem (Input1 – Input2), I went with a neural network that consisted of … Learning rate = 0.30, Momentum = 0.72, 2 Hidden Layers 15 Nodes each, LeakyRelu on both hidden layers and sig on output layer. Bias = -1 and weightInit = 1.00. MaxIterations = 1 million.

   Results:

```
Sum Squared Error for Training cases   = 0.0000
% of Training Cases that meet criteria = 1.0000

Sum Squared Error for Testing cases   = 0.0161
% of Testing Cases that meet criteria = 0.0000
```

The program did pass. It converged within 0.01 criteria.

It took 190,971 iterations to converge within the criteria.

The program seemed to generalize well. Taking around 200,000 iterations, and compared to other methods attempted, this seemed to generalize the fastest and within the desired criteria of 0.01.

No, I don't think this was able to learn well. Looking at the percent of test cases that passed we can see this is 0. Although the program successfully generalized, upon testing, none of the test cases passed. This could be due to the computation taking place or possibly the architecture being used here. I had some attempts that would yield 10% pass rate on test cases, however this took more iterations to generalize. This other method I used involved using Relu instead of leakyRelu with all the same settings shown above.

2. For the problem (AVG), I went with a neural network that consisted of …
   Learning rate = 0.30, Momentum = 0.80, 2 Hidden Layers 15 Nodes each, Relu on both hidden layers and sig on output layer. Bias = -1 and weightInit = 1.00. MaxIterations = 1 million.

   Results:

   ```
   Sum Squared Error for Training cases   = 0.0000
   % of Training Cases that meet criteria = 01.0000

   Sum Squared Error for Testing cases   = 0.0007
   % of Testing Cases that meet criteria = 0.3000
   ```

   The program did pass. It converged within 0.01 criteria.

   It took 64,862 iterations to converge within the criteria.

   I feel like it generalized well. Compared to the other problems I tested, this was the only one to converge in less than 100,000 iterations.

   With Relu applied on two hidden layer, 15 nodes each, I think the program was able to learn alright. Looking at the percent of test cases passed, we get 30% success. Although it is not the best, it is still learning within the required criteria, within 65,00 iterations, and successfully guessing 30% of the test cases.

3. For the problem (MIN), I went with a neural network that consisted of …
   Learning rate = 0.30, Momentum = 0.71, 2 Hidden Layers 15 Nodes each, LeakyRelu on both hidden layers and sig on output layer. Bias = -1 and weightInit = 1.00. MaxIterations = 1 million.

   Results:

   Sum Squared Error for Training cases = 0.0179
   % of Training Cases that meet criteria = 1.0000

   Sum Squared Error for Testing cases = 0.0055
   % of Testing Cases that meet criteria = 0.3000

   The program did pass. It converged within 0.01 criteria.

   It took 251,809 iterations to converge within the criteria.

   I feel like it generalized well. The program was able to converge in around 250,000 iterations. Which compared to other attempts made, I feel like the performance is fairly good for this problem.

   Yes, I think the program was able to learn alright. Looking at the percent of test cases passed, we get 30% success. Although it is not the best, it is still learning within the required criteria and successfully guessing 30% of the test cases.

Part 3

Description:

The image classification problem that I have created will tell whether the image being tested is a ‘cat’ or a ‘chicken’. It uses pictures of size 100x100 pixels, with file type of jpg and color type RGB.

The way I developed my training and testing sets was to go through both Bing and Google image search and look for random images that included either a cat or a chicken, but not both. When I looked for my pictures, I looked for images that contained the full body of the animal. This was so the image recognition could pick up on the distinct aspects of the animal shown in each test case. I also tried to not take any images that contained different animals as to eliminate the possibility of learning improper traits of what it’s trying to classify. There is a reference list at the end of this report. It lists all the websites I obtained my training and testing images from.

When it comes to formatting the images, I used Photoshop and ran a script to resize my images to 100x100.

Summary:

The CNN architecture that worked best for me was AlexNET, the provided CNN Architecture style. I tried using VGGNet, which consists of 16 convolutional layers with a uniform architecture. I became intrigued with VGGNet while researching different CNN architecture styles. This architecture style was developed in 2014 and is a lot like AlexNet, but with lots of filters. According to, https://ww.medium.com, "It is currently the most preferred choice in the community for extracting features from images." Knowing this information, I was pushed towards this architecture style because of it. Unfortunately though, when I tried running this architecture style I ended up with results that were not as good as AlexNET. As well, the time taken between iterations took much longer due to the added convolutional layers. The reason why VGGNet performed worse than AlexNET could be due to my implementation of it? However, it could be possible that AlexNet worked better in the scenario I have.

Results:

Overall my program ran for 300 iterations, (epoch = 300), for my training.

I feel like my program was able to generalize fairly well. It was able to correctly classify 68% of my images, which when compared to my tensorflow graphs and the accuracy maintained and achieved from them, the information makes sense. While training, I can see from the distribution on the accuracy graph, that my accuracy began at around 49% and ended up towards the end with a peak of 86.12%. And looking at the average from this graph and comparing to my results from 'predict.py', the precision of 68% seems reasonable.

From the results of predict.py, it was able to correctly classify 11 out of the 16 test images I used. It incorrectly identified image 3, 8, 9, 10, and 13. Two of those incorrectly classified images where of cats while the other three were of chickens. The actual results are listed below.

When it comes to how well I think my program learned, I feel like it was able to learn well. Being able to classify 68% of the test images correctly proves to me that it learned to recognize when there is a chicken or a cat within the picture. Having my program obtaining a 68% pass of its tests, shows that improvements could still be implemented. The best way to increase the accuracy would be to add more training cases along with changing my CNN architecture style.

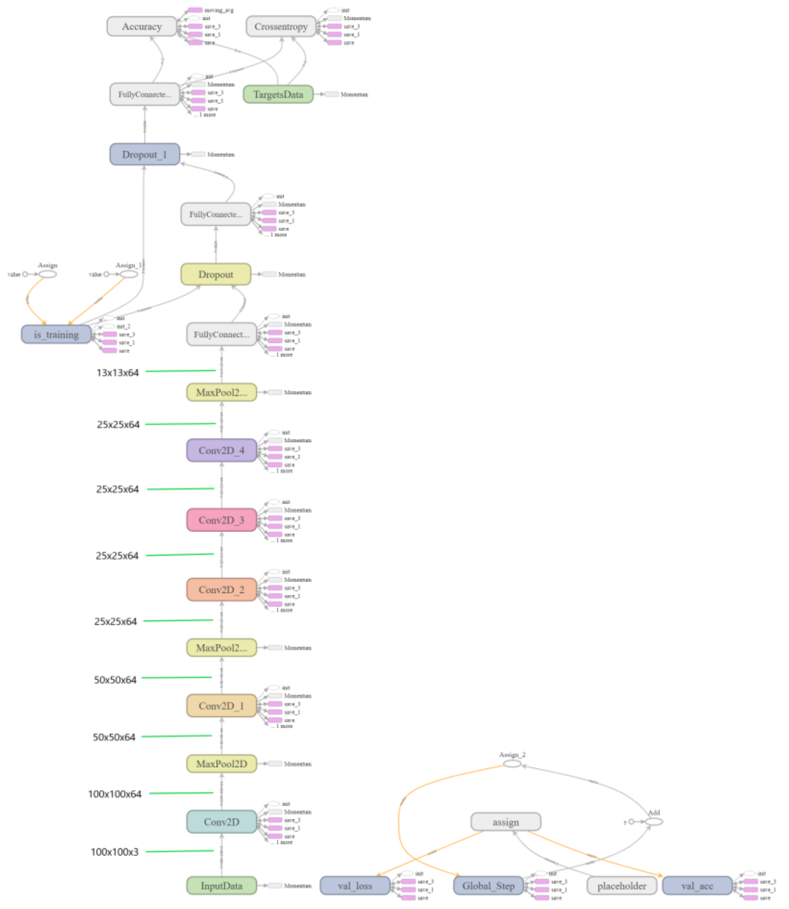
'predict.py' results:

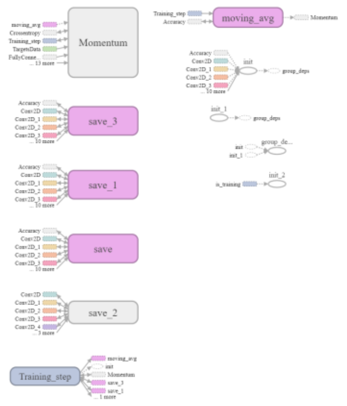
```
For CNN architecture type AlexNET

image 0 CORRECT  [0.5799549  0.42004517]

image 1 CORRECT  [0.8147987  0.18520126]

image 2 CORRECT  [0.7673305  0.23266941]

image 3 WRONG    [0.40559658 0.5944034 ]

image 4 CORRECT  [0.77833724 0.22166279]

image 5 CORRECT  [0.5517245 0.4482755]

image 6 CORRECT  [0.7249949  0.27500507]

image 7 CORRECT  [0.52121186 0.47878817]

image 8 WRONG    [0.797752   0.20224796]

image 9 WRONG    [0.7694563  0.23054375]

image 10 WRONG    [0.71445453 0.2855455 ]

image 11 CORRECT  [0.17624567 0.8237543 ]

image 12 CORRECT  [0.1516784 0.8483216]

image 13 WRONG    [0.76951057 0.23048942]

image 14 CORRECT  [0.32867536 0.6713246 ]

image 15 CORRECT  [0.14058064 0.8594193 ]
```

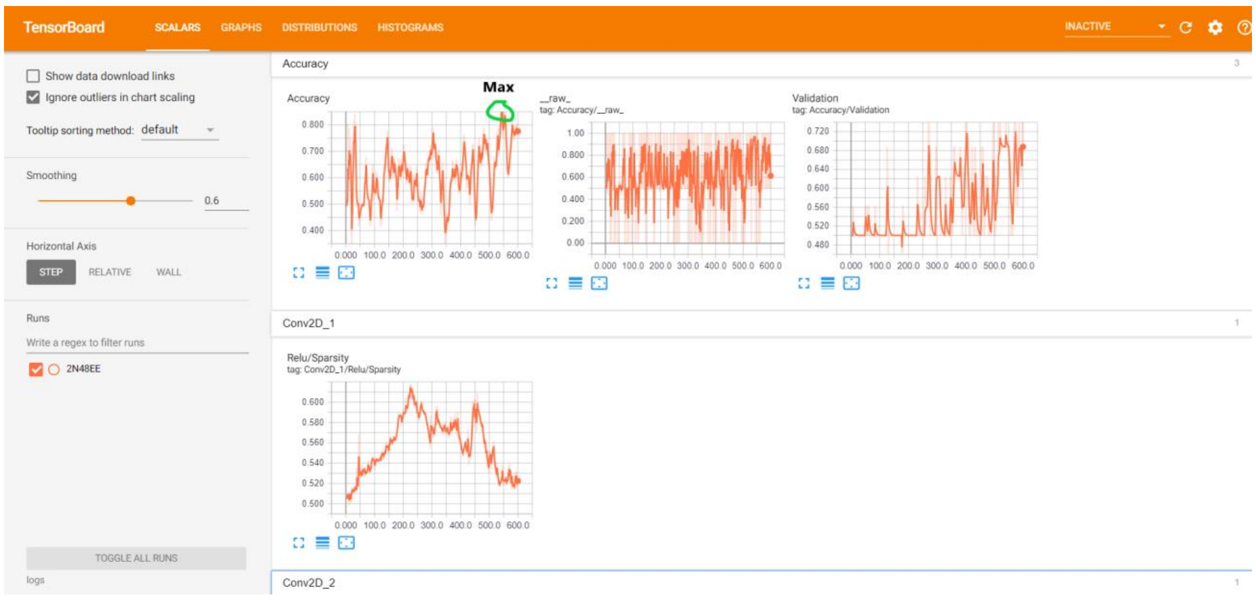
When looking at the output shown above, we can see that several of the images that were labeled CORRECT, had fairly good accuracy. For example, with image 15, we see that it was about 86% sure that the image was a chicken while only 14% that it might be a cat. Having this distinct difference helps validate that the program did learn to recognize the image.

Craig Hulsebus Project 3 11/26/2018

TensorBoard Architecture Graphs:







☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs
Write a regex to filter runs
☑ 2N48EE

Accuracy
tag: Accuracy/__raw__
tag: Accuracy/Validation
tag: Conv2D_1/Relu/Sparsity

Relevant Dimensionalities:

On the graph above, we can see the relevant dimensionality on the ARCS shown between convolutional layers and fully connected layers. From input to the first Conv2D (our 2D convolutional layer) we have 100x100x3. This makes sense consider the image is 100x100 pixels and is x3 for RGB. Next, going from Conv2D to MaxPool2D we have dimensionality of 100x100x64. Our third dimension value 3 in now 64. This is because the MaxPool2D is splitting up the image into different filters it will use to identify the image. From here we go to our next Conv2D where dimensionality is now 50x50x64. The reason for the value change is due to reduction within the architecture. The convolutional layer is extracting features from the 100x100 image into a reduced form of 50x50. This dimensionality continues on to the second MaxPool2D layer. Feeding back into another Conv2D layer the image is further reduced from 50x50x64 to 25x25x64. This again due to the convolutional layer extracting features from the image in order to reduce it. After going through 2 more Conv2D layer, we feed back into a MaxPool2D. This finally feeds in to our FullyConnected layer where further reduction occurs from 25x25x64 to 13x13x64. This entire series correlates correctly to the relevant dimensionality on the graphs ARCS.

TFLearn Script Code: AlexNet

```
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, 100, 100, 3])

# Conv layers ------------------------------------
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu')
network = conv_2d(network, 64, 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, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 2, activation='softmax')
# Final network
network = regression(network, optimizer='momentum',
                loss='categorical_crossentropy',
                learning_rate=0.001, metric=acc)
model = tflearn.DNN(network, tensorboard_verbose=3, tensorboard_dir="logs")
#model = tflearn.DNN(network)
```

Image References

The following is a list of images used during training and testing. It's ordered by the labeled picture followed by the URL it was found from.

Note: Images 101 – 130 are found from Bing Images, Images 130-150 and 1-50 are found from Google Images. Some images are missing a url link. This is due to not being able to relocate that particular image again within google or bing search. If there is an image without a url that needs to be located, the images were grabbed from the first few pages of pictures displayed when searching for either 'live chicken' or 'cat'. The image should be located somewhere within that google or bing search.

1.jpg = https://news.nationalgeographic.com/2018/05/animals-cats-training-pets/

2.jpg = https://www.catster.com/cats-101/savannah-cat-about-this-breed

4.jpg = https://www.petmd.com/cat/behavior/evr_ct_what-does-it-mean-when-a-cat-wags-tail

5.jpg = https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjBjdD9l_jeAhVSFzQIHcJCD_QQjhx6BAgBEAM&url=https%3A%2F%2Fhackernoon.com%2Fa-guide-to-giving-your-cats-their-annual-performance-review-fbf14610305&psig=AOvVaw3NciCQitD69hrLHqEhCwhi&ust=1543532150922355

8.jpg = https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjonYGRmPjeAhX-IjQIHfFcC3sQjhx6BAgBEAM&url=https%3A%2F%2Fcatscornervet.com%2F&psig=AOvVaw3NciCQitD69hrLHqEhCwhi&ust=1543532150922355

9.jpg = https://unsplash.com/search/photos/cats

11.jpg = http://time.com/5382551/new-zealand-cat-ban/

12.jpg = https://www.purina.co.uk/cats/key-life-stages/ageing/cats-age-in-human-years

13.jpg = https://abcnews.go.com/Lifestyle/scaredy-cats-absolutely-terrified-cucumbers/story?id=35953032

14.jpg = http://www.petsexperience.com/cat/what-are-hybrid-cat-breeds.html

15.jpg = https://www.mnn.com/family/pets/stories/toby-cat-walked-12-miles-get-back-family-didnt-want-him

16.jpg = https://commons.wikimedia.org/wiki/File:Felis_catus-cat_on_snow.jpg

18.jpg = http://www.royalcanin.ca/products/cat-nutrition/adult-cat

19.jpg = https://timesofindia.indiatimes.com/life-style/relationships/pets/popular-cat-breeds/articleshow/60827996.cms

20.jpg = https://wagznwhiskerz.com/11-reasons-cat-needs-daily-visits-away/

23.jpg = https://twistedsifter.com/2014/10/this-cats-chin-fur-makes-him-look-forever-surprised/

25.jpg = http://www.vetstreet.com/cats/

27.jpg = https://www.prestigeanimalhospital.com/services/cats/blog/4-types-cat-cancer-and-their-common-symptoms

28.jpg = https://news.nationalgeographic.com/2017/06/domesticated-cats-dna-genetics-pets-science/

30.jpg = https://www.msah.com/services/cats/blog/playing-your-cats-%E2%80%93-every-day

31.jpg = https://animals.howstuffworks.com/pets/is-it-ok-for-cats-to-drink-milk.htm

34.jpg = http://www.petmania.ie/cat/choosing-a-cat-as-a-pet/kitten-adult-or-senior-cat-

36.jpg = https://www.royalcanin.com/products/cat/senior

39.jpg = https://people.com/pets/iambronsoncat-33-pound-cat-weight-loss-journey/

41.jpg = https://www.alleycat.org/resources/kitten-progression/

42.jpg = https://www.care.com/c/stories/6375/how-to-tell-if-your-cat-is-pregnant-5-tell-t/en-gb/

49.jpg = http://oakpark.outugo.com/services/cat-sitting/

102.jpg = https://www.sowetanlive.co.za/news/2017-06-29-avian-flu-government-imposes-conditions-on-live-chicken-trade/

103.jpg = https://upload.wikimedia.org/wikipedia/commons/thumb/d/d3/Partridge_Cochin_cockerel.jpg/1200px-Partridge_Cochin_cockerel.jpg

104.jpg = https://3.bp.blogspot.com/-2jOAxkD-HQg/WSGfqVo6axI/AAAAAAAAjBY/0sk3aZFv6WgBnPVnf25W4qUnUXbZQ8jVgCLcB/s1600/started-buff-orpington-chickens.jpg

106.jpg = http://hdwallpaperg.com/wp-content/uploads/2014/02/Chicken-Picture.jpg

107.jpg = http://www.corvallisadvocate.com/wp-content/uploads/2015/05/DSC_0133.jpg

110.jpg = http://4.bp.blogspot.com/-FckC9pP-y7Y/T2bh_e5Ph-I/AAAAAAAAGsg/oIV2mgvuWE4/s1600/Rooster-pictures-3.jpg

111.jpg = http://3.bp.blogspot.com/-kOCNkjPph1c/UiENwuQGOtI/AAAAAAAAAYg/XKhbGC2ddmE/s1600/Chicken-Images.jpg

113.jpg = https://i2.wp.com/www.typesofchicken.com/wp-content/uploads/2015/12/Booted-Bantams-Chickens.jpg

116.jpg = https://www.indiamart.com/proddetail/broiler-chicken-16341318962.html

117.jpg = http://awallpapersimages.com/wp-content/uploads/2016/07/chicken-nature-hd-wallpaper-wallpapers.jpg

122.jpg = http://thistlehillplantation.com/IMAGES/New%20Folder/chicken%20(7).jpg

127.jpg = https://www.cacklehatchery.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/s/h/shutterstock_76802203_1_1.jpg

132.jpg = https://www.exportersindia.com/gp-orgo/live-chicken-trichy-india-1527262.htm

137.jpg = http://www.novagriproducts.com/ar/livestock/live-chicken-broiler-1kg-detail