Computer Vision Final Project

Joel Yuhas : jxy8307

April 2019

1 Introduction

There are many different types of fruits and vegetables in the world. The objective of this project was to be able to identify images of some of these fruits using a deep neural network. Two different data sets were used to test the neural network One data set contained well shot photos of fruit with white backgrounds, while another data-set contained fruit "in the wild", with fruit in a variety of situations. A pre-trained version of Alexnet was used, which used transfer learning to train the network further. Several parameters of the network were modified in order to get the most effective results possible. This also served as a way to measure the effectiveness of Alexnet compared to related work done in similar situations. The initial objective was to have at least 10 different classes of fruit. However, the model ended up performing well above expected, so the class size was increased to 101 and 30 for each respective data set. Overall, the project ended up being a great success, and was able to far exceed expectations at recognizing fruit both out and in the wild.

2 Related Work

Fruit recognition is actually a very widly studied topic. There are many fields within agriculture that study fruit, vegtables, and other vegetation using machine learning. The following are some pieces of related work that has been done by others.

DeepFruits: A Fruit Detection System Using a Deep Neural Networks.

This project created an "accurate, fast, and reliable fruit detection system" for an autonomous agricultural robotic platform. It used a Faster Region-based CNN (Faster R-CNN) architecture for object detection to separate the fruit from the background, and a 16 layer VGG network to then recognize the fruits. This project worked in both the RGB and Near-Infrared fields of vision. This project also mainly recognized fruit in their "natural environment" and was successfully at being able to accurately pick out and reconize the subjects.

Automatic Fruit Recognition from Natural Images Using Color and Texture Features.

This project was created for a robot-assisted harvesting device. This one focused more on finding new approaches for classifying fruits in order to separate them from the background. Texture features form Gray-level, co-occurred Matrix (GLCM) and statistical color features were used together to try to identify the fruit. This project was also geared toward embedded systems and streamlined devices, which focused on real-time computing. A SVM network was utilized to reconize the fruit that was seperated/detected by the GLCM.

3 Methods

Alexnet was the primary architecture that was used for this project. Alexnet was loaded and manipulated using pytorch, which was also responsible for loading the pre-trained layers. The network took in images that were 256 x 256, which meant all images had to be resized before they could be used. In total, 5 convolutions layers were utalized with 3 fully connected layers. The only layer that was modified from the initial initialization was the final layer, which was adjusted relative to the number of classes. The figure below shows a diagram of the layers within Alexnet.

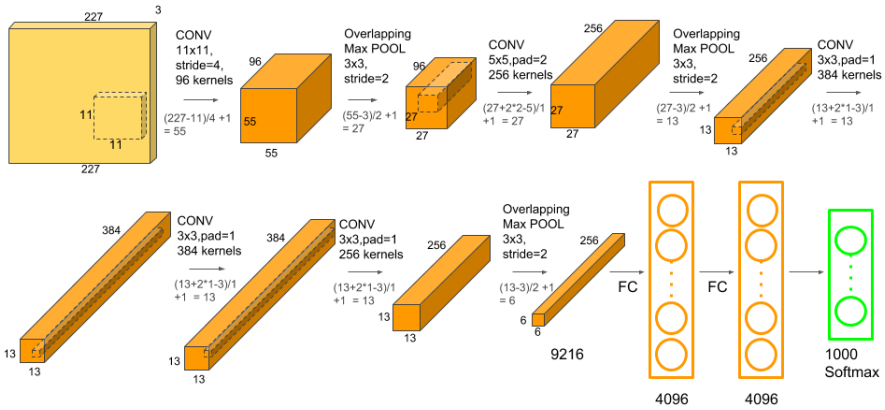


Figure 1: Alex Net Layout Diagram

Some of the network parameters were modified to better execute each data set. Specifically, the learning rate and the momentum were typically altered to observe their effects on accuracy. The code written also had a mechanism that would decay the learning rate by a factor of 0.1 every 7 epochs. Each official experiment was ran with a minimum of 20 epochs, and a maximum of 60. The network was also retrained for each data set repspectivly.

It should be noted that the main framework of the code was heavily based

off of the tutorial linked in homework 5. This tutorial was modified in order to get the desired results and is included in the reference section.

There was also code written that automatically saved all loss and accuracy information for both the training and test set experiments to an excel file, to make data analysis easier.

The initial tests were ran on a laptop CPU. However, later test were ran on a dedicated desktop GPU, which significantly decreased training times.

Lastly, it should be noted that all of the code was written using Jupyter Notebook. The code has also been into a .py file for ease of reading and the notebook that was used has been included in the submission as well. Commented lines in the .py file delineate where different task were created in the notebook.

4 Experiments

Two main data sets were used in this project. One was the "Fruits 360" dataset, which contained over 65000 images with over 101 classes. These images were very high quality and included a single fruit of the specified class with multiple photos shot at multiple angles with white backgrounds. The figures below show some example images and their corresponding classes.

Figure 2: Banana

Figure 3: Cactus Fruit

Figure 4: Pineapple

Figure 5: Lemon

Figure 6: Apple

Because the data set was rather large, training times on the laptop CPU took exceptionally long. At one point the testing was stopped so that the project could be transferred over to a dedicated GPU, which drastically sped up training times. The initial plan was to use only 10 of the 101 classes, both to simplify the training but also cut back on training duration. However, after even only a few epochs of training, the network was able to identify all test images with 100 percent accuracy. This was deemed to easy, so the decision was made to use all 101 classes in order to observe the networks effect on a larger scale. This increased training time but also made the system more robust.

The dataset was also divided into a test and training set, with the test sets being completely new images not included in the training sets. There were roughly 10,000 test images and 52,000 training images.

The "FIDS30" dataset was the second database that was used to test the network. This data set included photos of fruits "in the wild", meaning that the images were purposefully more abstract with less consistency in order to replicate real world conditions. There were only about 30 classes, each with close to 50 images for around 1500 photos in total.

Initially, a few images were separated out to act as the test batch while the rest operated as the training batch. However, This wasn't desired since it took away images from the training data set, so instead all new images were gathered from google to serve as the training set. This put the training images at around 50 per class with 8-10 for the test set. The figures below show some of the images that were used in the data set.

Figure 7: Banana

Figure 8: Apples

Figure 9: Grapes

Figure 10: Lemon

Figure 11: Passion fruit

This data-set had more manipulation of network parameters than the first, as the second one proved to be more difficult when attempting to get higher accuracy's. Since the size of the data set was smaller, training times were significantly quicker. This let the tests be ran for up to 60 epochs each. The figures below show some of the loaded data being displayed by the program with their corresponding classes.

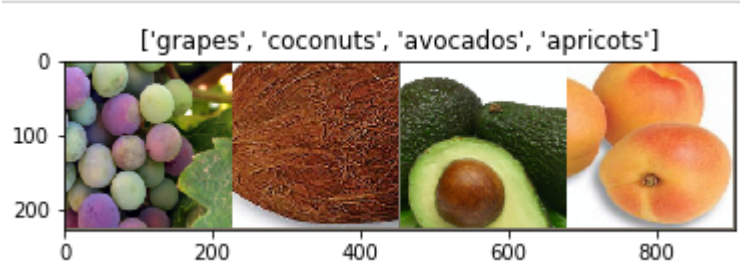


Figure 12: FIDS30 Images Displayed in Program

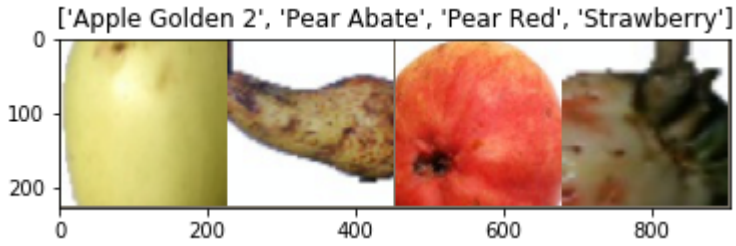


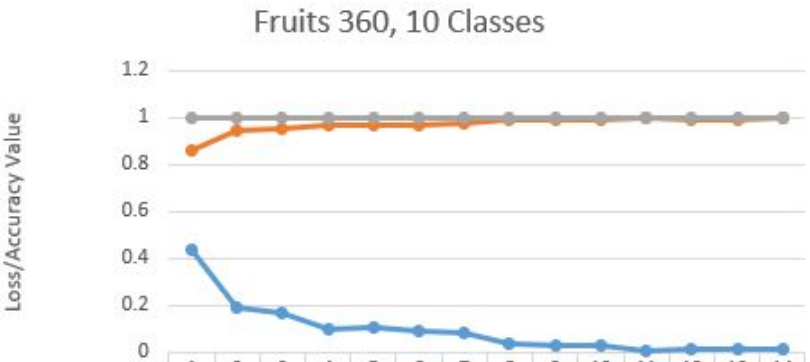
Figure 13: Fruits 360 Displayed in Program

5 Results

As mentioned in the experiment sections, the Fruits 360 data set was originally ran with only 10 classes. The initial training far exceeded performance expectations even after only a few iterations, so the decision was made to use all 101 classes. The figure below shows what some of the predicted outputs looked like when printed from the program.

Figure 14: Fruits 360 Output Display Images

The figure below shows the numerical results from the exploratory testing that was done. Notice how even in the beginning, the test accuracy was 100 percent.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Loss (train)	0.44	0.19	0.17	0.1	0.11	0.09	0.08	0.03	0.03	0.03	0.01	0.02	0.01	0.01
Acc (train)	0.86	0.94	0.95	0.97	0.97	0.97	0.97	0.99	0.99	0.99	1	0.99	1	1
Acc (val)	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 15: Fruits 360 Initial 10 Classes Run

After all 101 classes were Incorporated, the network was trained for 20 epochs using a learning rate of 0.001, which decremented by a factor of 0.1 every 7 epochs, and a momentum of 0.9. The figures below shows the loss and accuracy results for the training in both table and figure form.

epoch	Loss (train)	Acc (train)	Acc (val)
1	0.8778	0.7451	0.9582
2	0.3866	0.8819	0.9761
3	0.3163	0.9046	0.968
4	0.2713	0.9179	0.985
5	0.2464	0.9251	0.9814
6	0.2252	0.9327	0.9784
7	0.2208	0.9336	0.984
8	0.0747	0.9757	0.9983
9	0.0573	0.9812	0.9994
10	0.0497	0.9833	0.9989
11	0.0438	0.9851	0.9997
12	0.0417	0.986	0.9995
13	0.0395	0.9865	0.9994
14	0.0369	0.9879	0.9996
15	0.032	0.9898	0.9997
16	0.0331	0.9889	0.9996
17	0.0319	0.9898	0.9997
18	0.0305	0.9894	0.9995
19	0.033	0.9891	0.9995
20	0.0306	0.9894	0.9992

Figure 16: Fruits 360 101 Classes Results

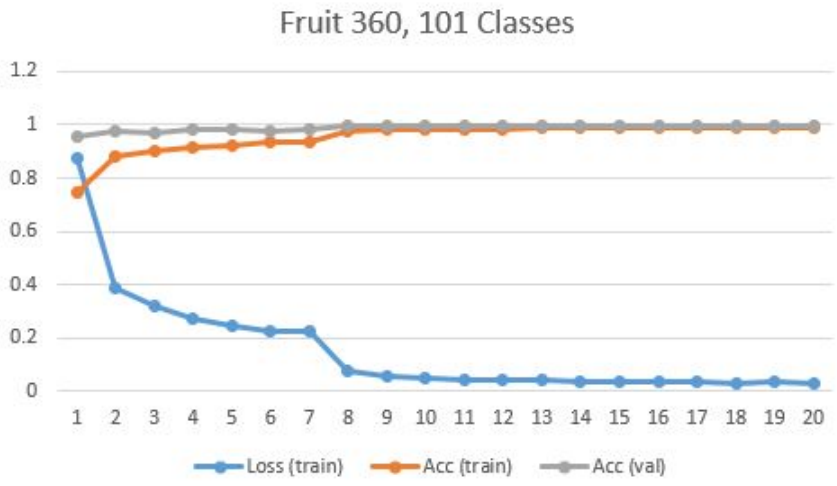


Figure 17: Fruits 360 101 Classes Results

Notice how the accuracy values for both the training and the testing were able to get above 98 percent. The testing values even reached up to 0.9997 at one point, nearly perfect. This was rather impressive considering the system was testing and training on over 65,000 images. Because the results were so successful, no changes to the network were made, and it was only trained up to 20 epochs. The decision to stop at 20 epochs was also due to the extremely long amount of time it took to train on the Fruits 360 data set.

The data shows that around iteration 8 is when the test values exceeded 0.99 and began to plateau. There was also a significant loss drop, from 0.2208 to 0.0747 at this time.

Next, the FIDS30 data set was tested. As mentioned previously, the original test had a few images taken from the training pool to use as the test pool. The figures below show the initial test that was done with around 3 photos for each class in the test sets.

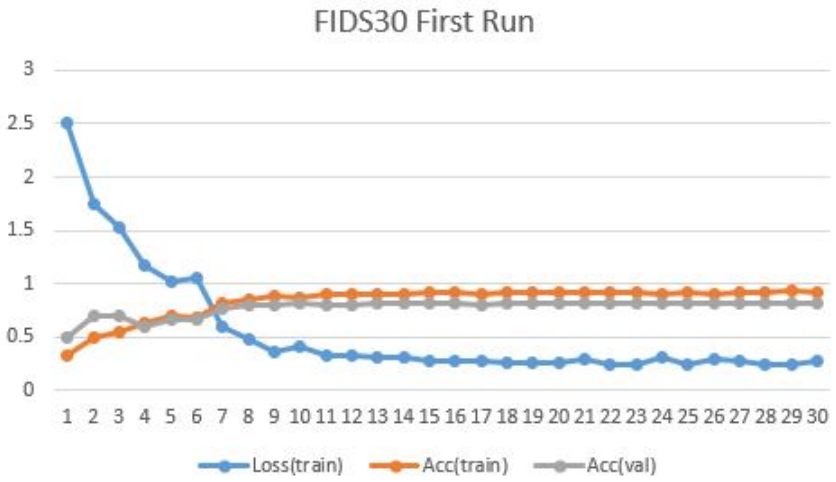


Figure 18: FIDS30 Initial Results

Figure 19: FIDS30 Output Images

epoch	Loss(train)	Acc(train)	Acc(val)
1	2.5117	0.3178	0.5
2	1.7407	0.4892	0.6889
3	1.5234	0.5414	0.6889
4	1.1778	0.6345	0.5889
5	1.0208	0.6901	0.667
6	1.055	0.6754	0.6667
7	0.5971	0.8116	0.7556
8	0.4701	0.8502	0.8
9	0.355	0.8899	0.7889
10	0.4079	0.8627	0.8222
11	0.3268	0.8922	0.8
12	0.3179	0.8956	0.8
13	0.3062	0.9047	0.8111
14	0.3011	0.9047	0.8111
15	0.2771	0.9171	0.8111
16	0.2643	0.9194	0.8111
17	0.2759	0.9047	0.8
18	0.2542	0.9205	0.8111
19	0.2585	0.9205	0.8111
20	0.2619	0.9103	0.8222
21	0.2823	0.9092	0.8222
22	0.2369	0.9183	0.8222
23	0.2393	0.924	0.8222
24	0.2988	0.9035	0.8222
25	0.2373	0.9228	0.8222
26	0.2865	0.9001	0.8222
27	0.2771	0.9103	0.8222
28	0.2464	0.916	0.8222
29	0.2408	0.9251	0.8222
30	0.2648	0.9171	0.8222

Figure 20: FIDS30 Initial Results

However, as shown by the data, the test pool was much to small to be effective. Certain numbers, such as 0.8111 and 0.8222 kept repeating, making the results not as detailed. Rather than take away more images from the training set it was decided to gather new test images of similar style from google. The figure below show the first major test done with the updated, data set. 60 epochs were done with a learning rate of 0.001 and a momentum of 0.9. Since a significant amount of epochs were executed, only the final numerical results and the graph were reported, rather than trying to display the whole table.

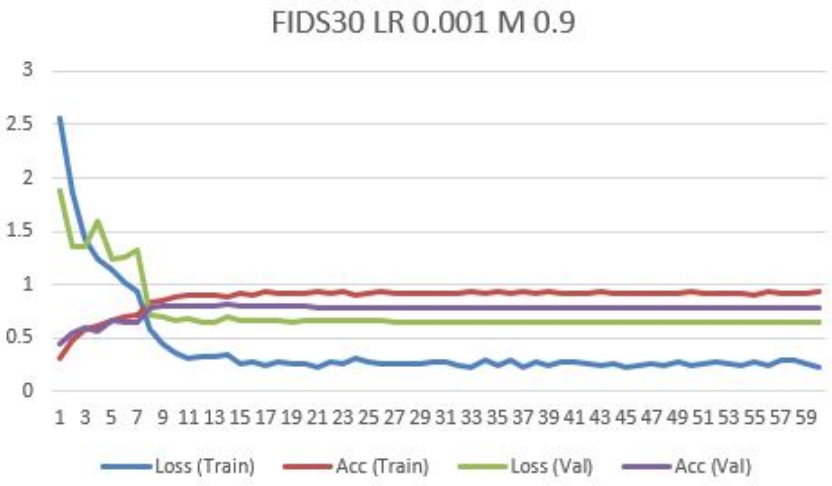


Figure 21: FID LR 001 M 09 Graph

Loss (Train)	Acc (Train)	Loss (Val)	Acc (Val)
0.228397585	0.936148301	0.65276811	0.78813559

Figure 22: FID LR 001 M 09 Final Results

This initial test proved to be pretty promising, but with only a final test accuracy of 0.788, there was definitely room for improvement. Additionally, the training did not see any major changes after 13 epochs, so it was decided to decrease the learning rate from 0.001 to 0.0001 and magnitude form 0.9 to 0.7. The figures below show the results for the following test.

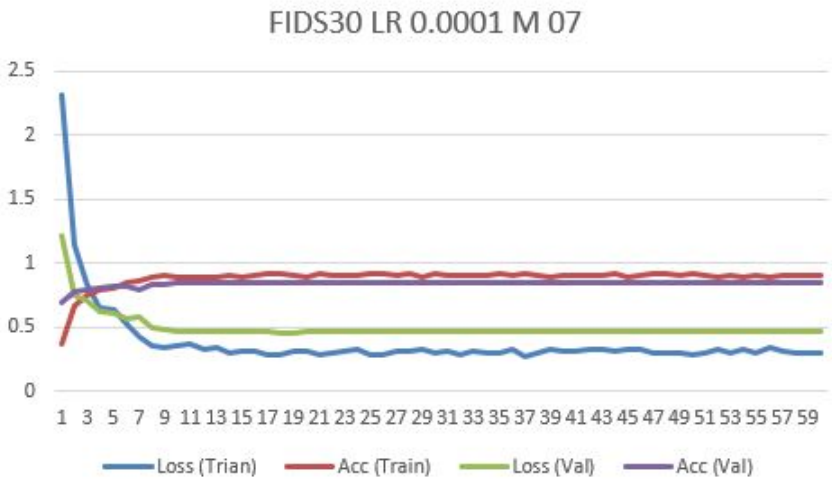


Figure 23: FID LR 0001 M 07 Graph

Loss (Trian)	Acc (Train)	Loss (Val)	Acc (Val)
0.296511875	0.907312049	0.46032241	0.85169492

Figure 24: FID LR 0001 M 07 Final Results

These changes in parameters produced much better test results, but worse training results. Training accuracy decreased from a final 0.93 to 0.90, but the test accuracy jumped from significantly to 0.78 to 0.85. It took longer for the loss values to even out, while surprisingly the accuracy's plateaued quicker than the previous test, evening out around 9 epochs. Another test was ran, this time with a standard learning rate at 0.001, but a lowered magnitude of 0.2. The figures below show those results.

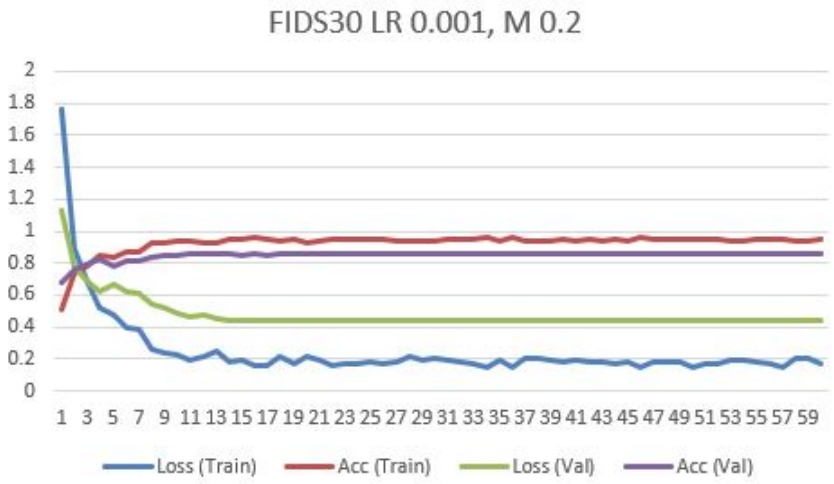


Figure 25: FID LR 001 M 02 Graph

Loss (Train)	Acc (Train)	Loss (Val)	Acc (Val)
0.168873664	0.950566426	0.44405857	0.8559322

Figure 26: FID LR 001 M 02 Final Results

As seen by the results, both the training accuracy and test accuracy increased. The training values also took longer for the data to even out, and were considerably more jumpy than the previous test. Specifically, the loss for the training value was very volatile throughout. However, a training accuracy above 95 percent and test accuracy above 85 percent was deemed acceptable and no further testing was done.

6 Discussion

Overall, the slightly modified versions of Alexnet performed well above expectations. To have a training and test accuracy above 99.99 percent was an amazing feat, and was done so with very little modification. This was most likely due to the very consistent and reliable dataset that was Fruits 360. Since it had over 10,000 images, it made training very effective. However, the downside to this dataset was that it was slightly unrealistic in the sense that the images were so well kept. In the real world, the fruit would most likely be obstructed or interfacing with the background. For that reason, that is why the FIDS30 test was so valuable.

The FIDS30 data set allowed the comparison of how Alexnet reacted between ideal situations and more realistic ones. Some of the disadvantages with the

FIDS30 data set however was that it was rather small compared to Fruits 360. Still, 1500 images still was plenty to get some very good results after tuning the network.

Additionally, the tests that were run in this project were different than most other test run by other projects. Specifically, most other projects that were reviewed had two main stages, one that would separate out the fruit from the background while the other would recognize them. Having Alexnet try to handle both in one go put it at a slight disadvantage.

In the future, a R-CNN network would most likely be the best way to go for starting the project. R-CNN and Faster-R-CNN variations seem perfect for being able to separate out the fruits from their backgrounds. Faster-R-CNN was even used by one of the other related works. Then a more traditional neural network, such as VGG or even Alexnet could be used for the recognition. This 2 stage approach seems to be more consuming, but ultimately more effective at getting the job done in the real world.

Ultimately, the project was a great success at not only creating an effective fruit recognition system, but also at learning the intricacies of neural networks, transfer learning, and more.

7 Teamwork

This was a solo project. All work was done by Joel Yuhas

References

Pytorch Code Tutorial from Homework 5

`https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`

Alexnet graphic and information

`https://www.learnopencv.com/understanding-alexnet/`

Fruit 360 Dataset

`https://www.kaggle.com/moltean/fruits`

FIDS30 Fruit Recognition Dataset

`https://www.vicos.si/Downloads/FIDS30`

Automatic fruit recognition from natural images using color and texture features

`https://ieeexplore.ieee.org/document/8074025`

DeepFruits: A Fruit Detection System Using a Deep Neural Networks.

`https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5017387/#B5-sensors-16-0122`