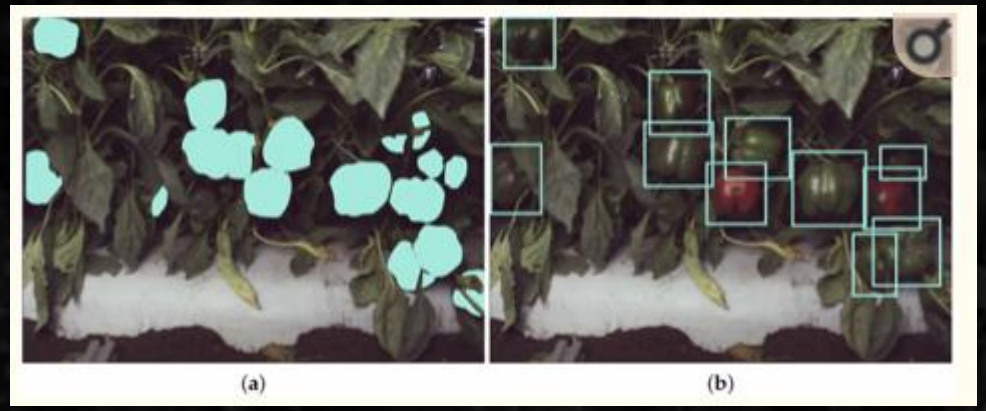
Fruit Recognition

Joel Yuhas

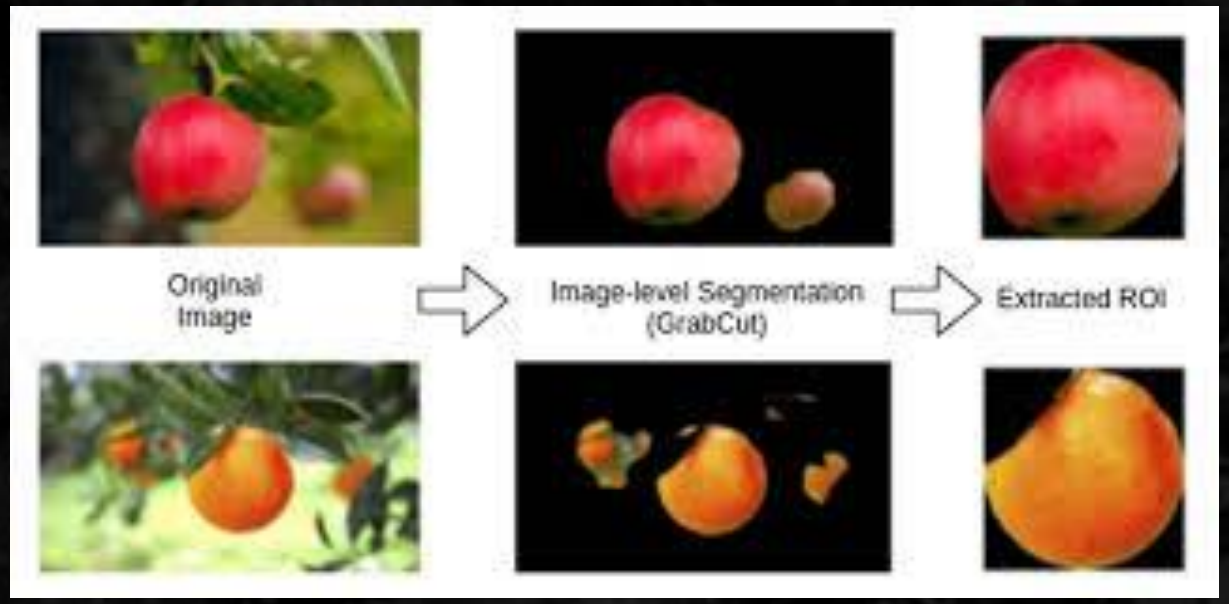
Related Work - DeepFruits

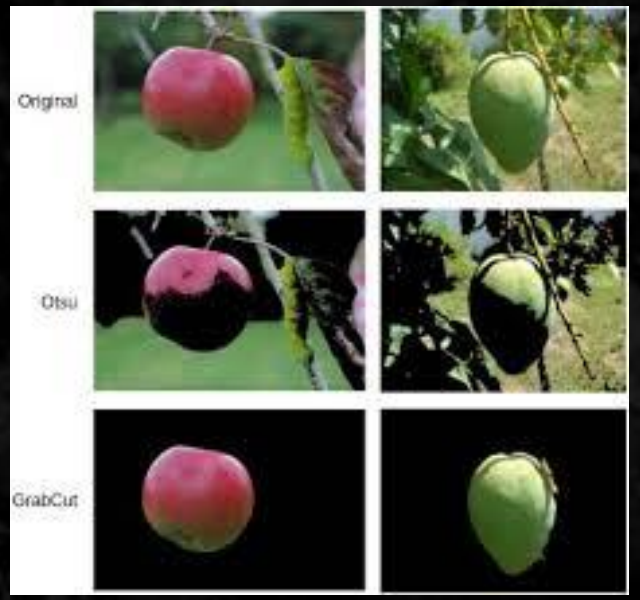
- “Fruit Detection System using deep neural networks”
- Created for autonomous agricultural robotic platform
- Used **Faster Region-based CNN**
- 0.838 accuracy for sweet pepper
- RGB and NIR light



Related Work – Automatic Fruit Recognition

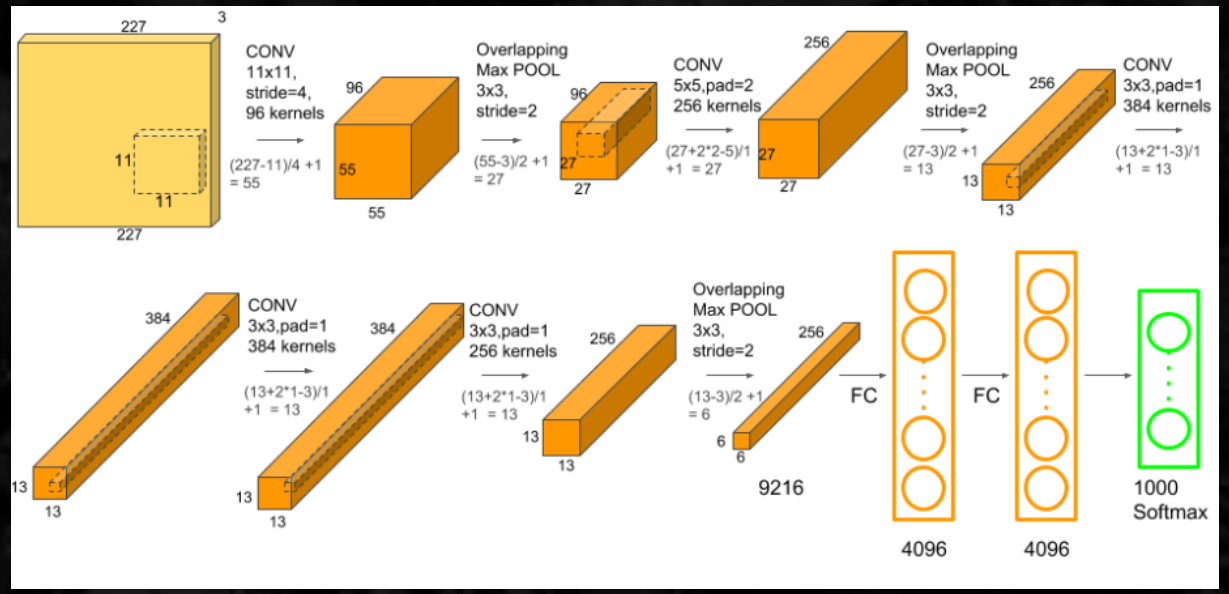
- Used color and texture features to recognize fruits
- Texture features from Gray-level Co-occurrence Matrix (GLCM)
- Support Vector Machine (SVM) Classification model





Methods

- **AlexNet** was the CNN used
- Started with default parameters
  - Learning rate 0.001 **decreased by 0.1 factor every 7 epochs*
  - Momentum 0.9
- **5 convolutional layers**
- Followed by max-pooling layers
- Last **3 fully connected layers**
- Only modified last layer to match class outputs
- Pre-Trained

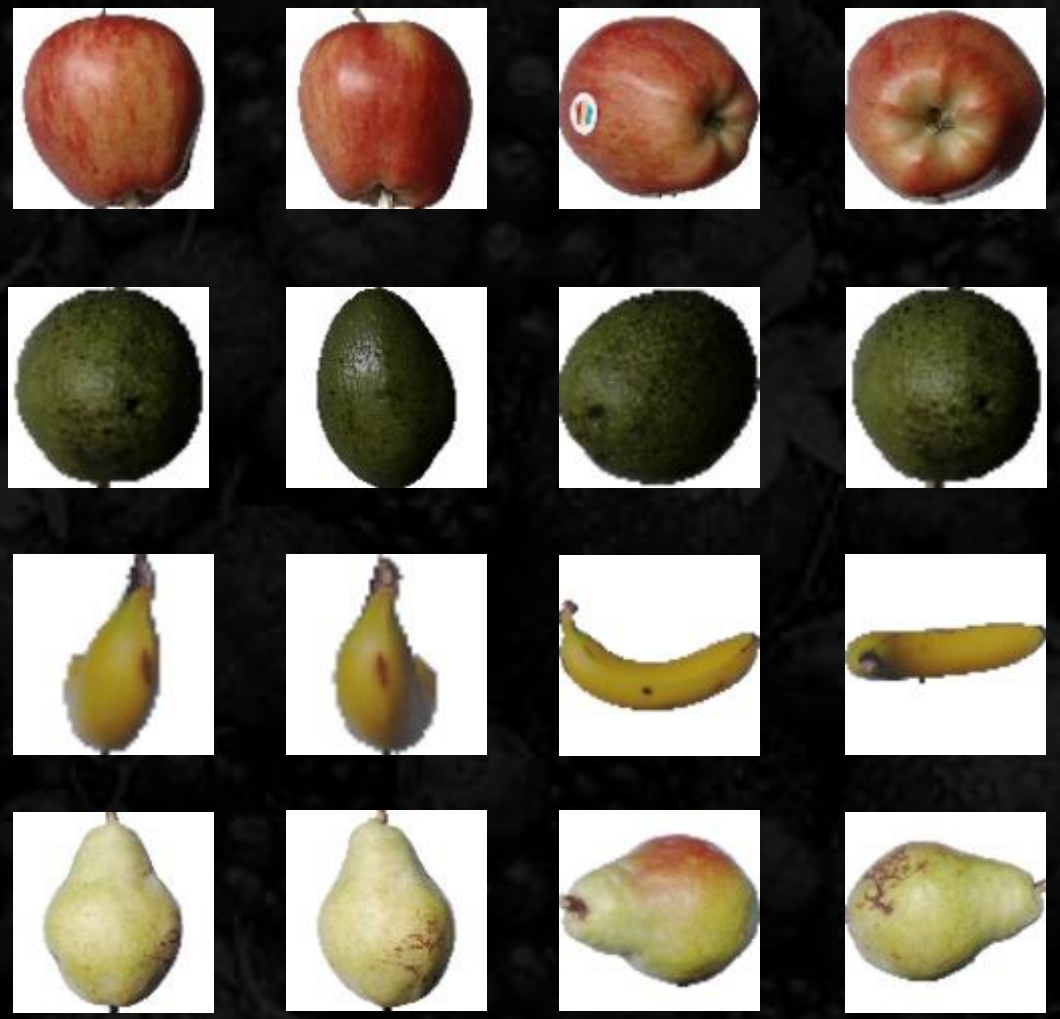


Methods

- Started with 10 classes, went up afterwards
- Images had to be scaled to 255 x 255
- Learning rate and magnitude changed as testing went on
- Batch size and layer configuration did not
- Transfer learning technique from homework 5 was employed as main foundation of the project

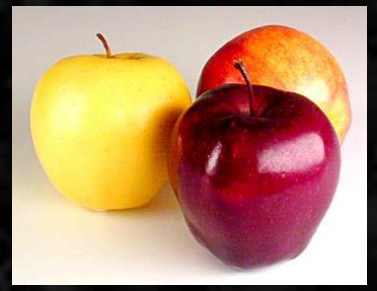
Experimental Methods – Datasets Fruits 360

- Fruits 360 Dataset
- 69905 images
  - ~ 62,000 training images
  - ~ 7,000 test images
- 101 Different classes
- Very clean images



Experimental Methods – Datasets FIDS30

- **FIDS30** data set
- Around 1500 images
- 30 different classes
- Images had much greater variety and was closer to real world situations
- Fruit “in the wild”
- Images ranged in size
- ~ 50 training images
- ~ 8 test *







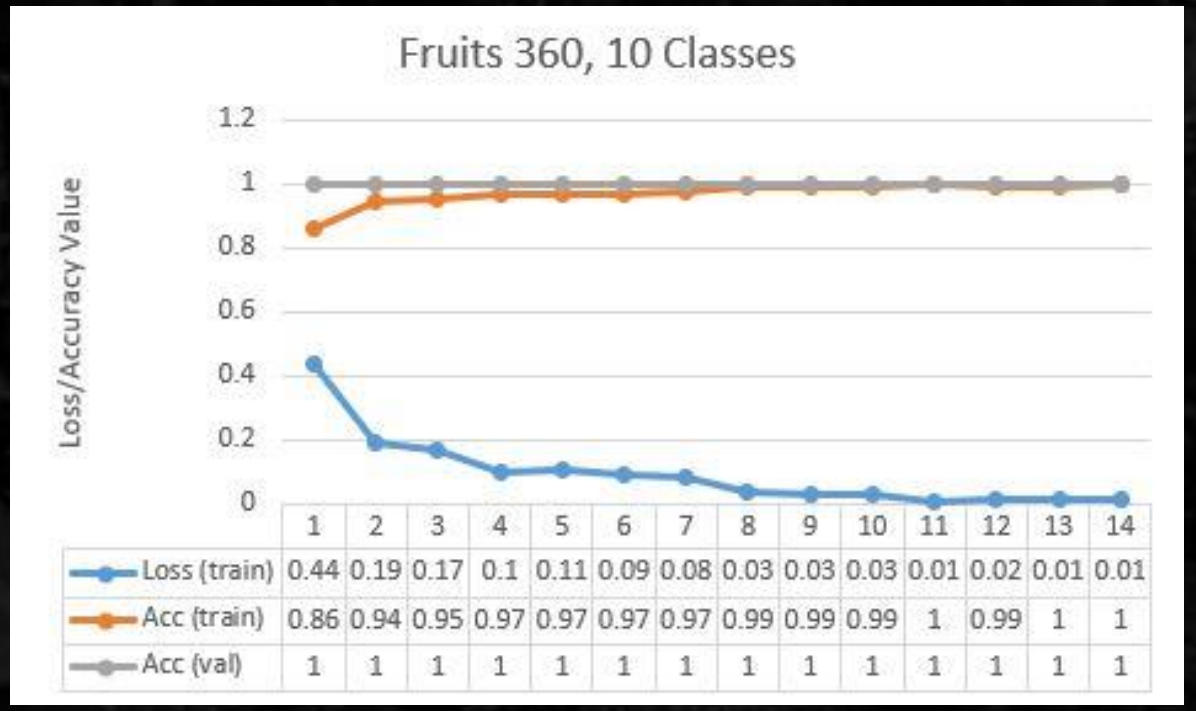


Experimental Methods – Protocol

- Test data with default parameters
- Adjust the learning rate and momentum per results
- Test 10 classes minimum, if good results increase classes
- Goal:
  - > 95% dataset 1
  - > 75% dataset 2

Results – Fruits 360 Dataset

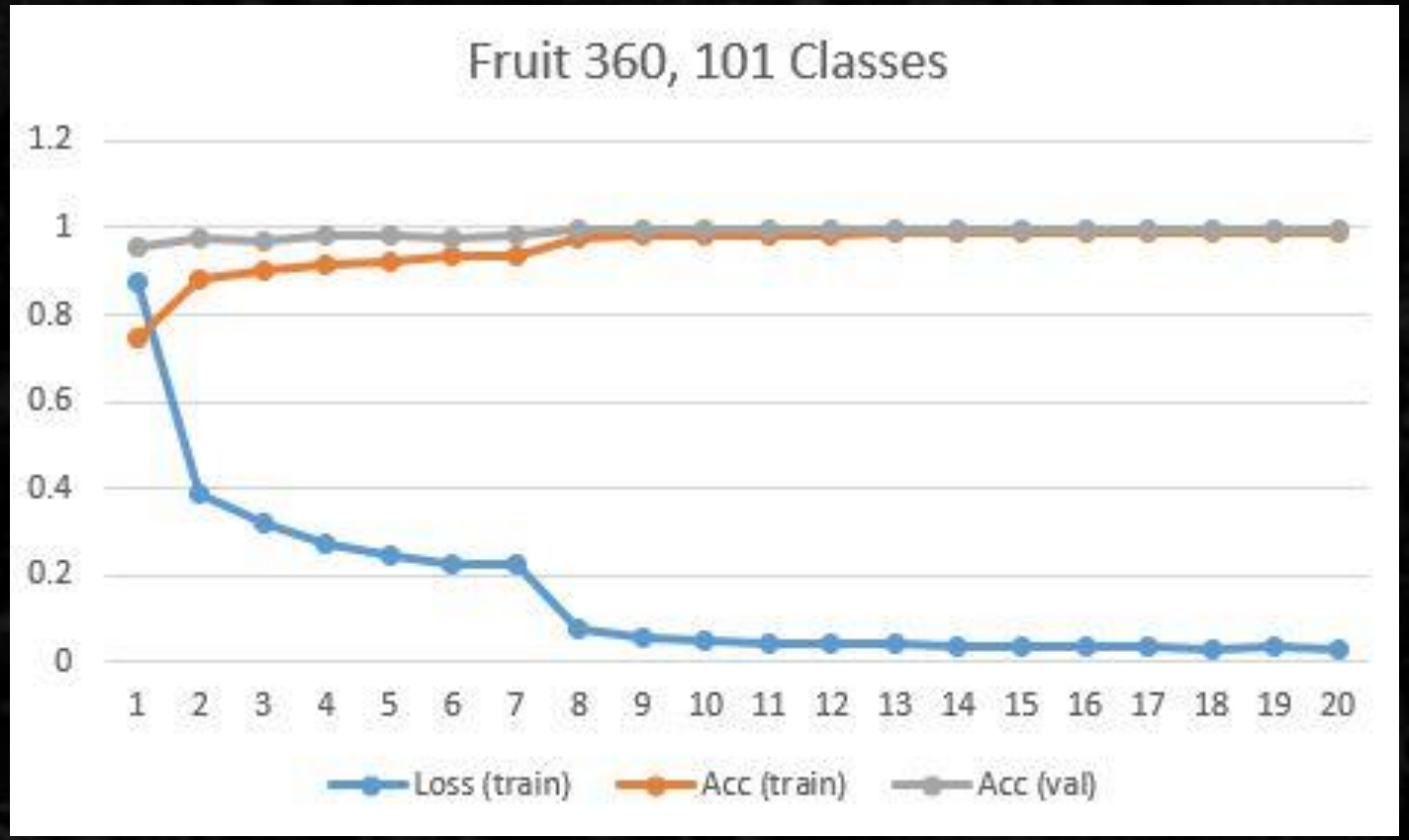
- Test with 10 classes had 99.99% training accuracy and 100% test accuracy with no changes
- Deemed too easy, so increased number of classes to 101



	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

Results – Dataset 1

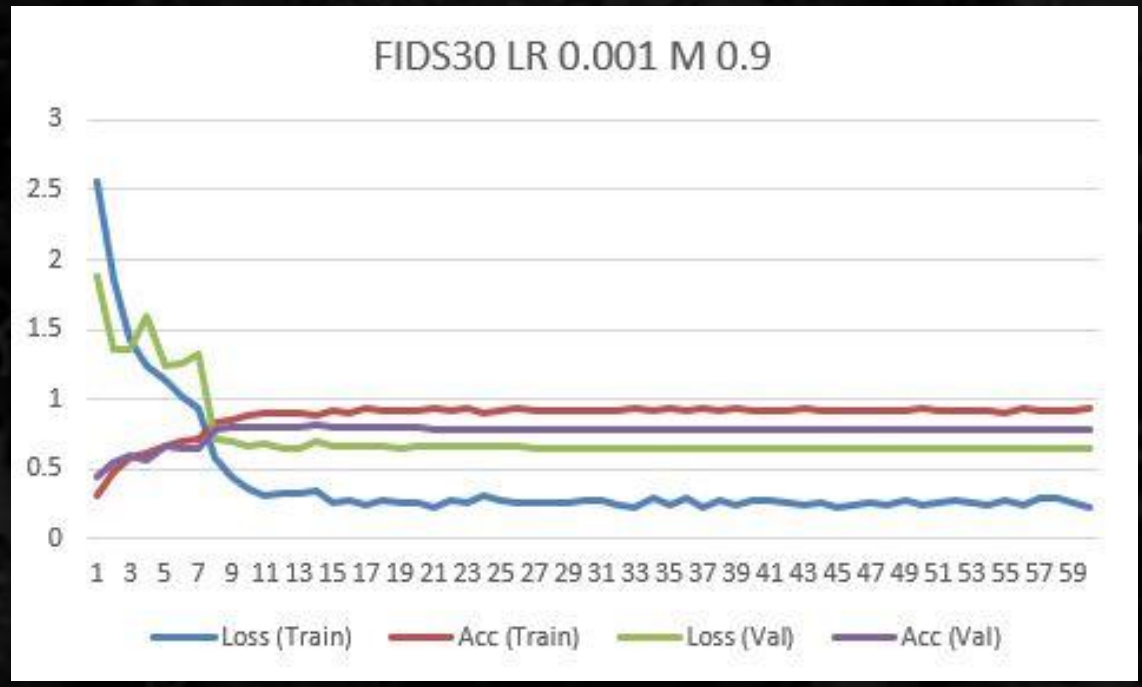
epoch	Loss (train)	Acc (train)	Acc (val)
1	0.8778	0.7451	0.9582
2	0.3866	0.8819	0.9761
3	0.3163	0.9046	0.968
4	0.2713	0.9179	0.985
5	0.2464	0.9251	0.9814
6	0.2252	0.9327	0.9784
7	0.2208	0.9336	0.984
8	0.0747	0.9757	0.9983
9	0.0573	0.9812	0.9994
10	0.0497	0.9833	0.9989
11	0.0438	0.9851	0.9997
12	0.0417	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



- Training accuracy of 98.94%, test accuracy of 99.92%
- Took 3+ hours to train

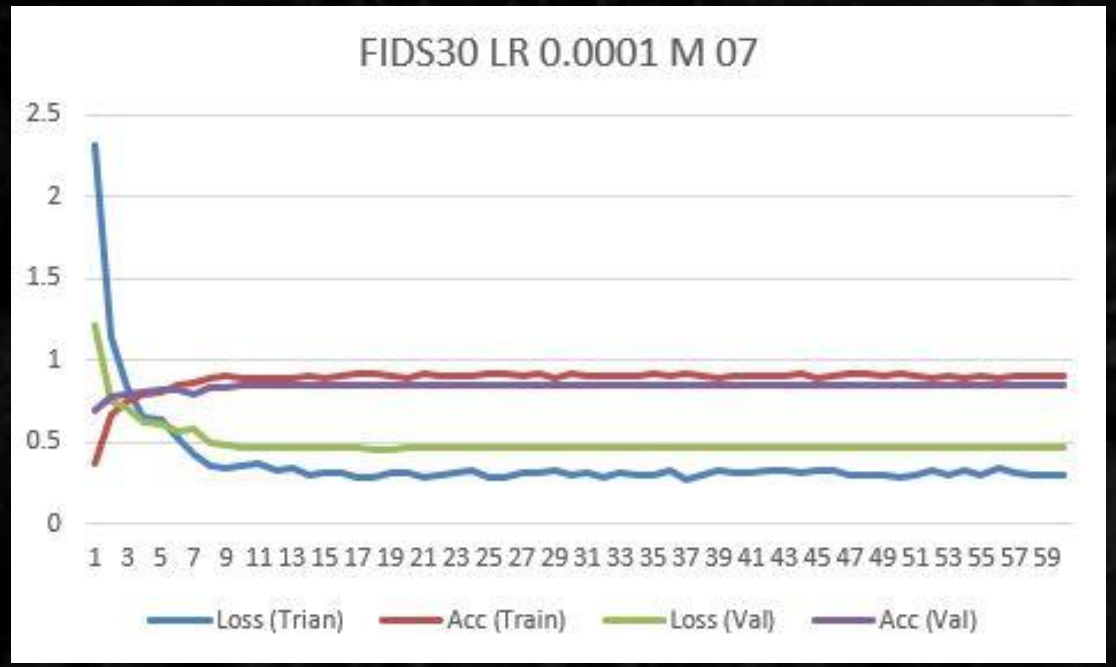
Results – FIDS30 Dataset 2

- Used full 30 classes
- Changed LR and M
- Ran with default parameters first
- Ran with 60 epochs each

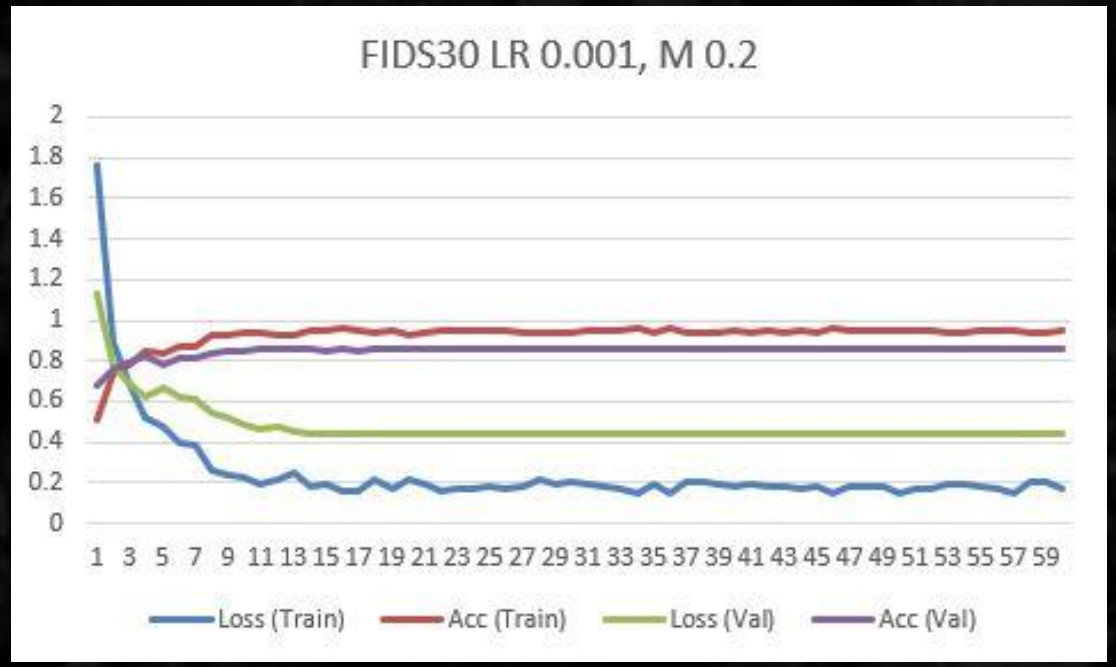


Loss (Train)	Acc (Train)	Loss (Val)	Acc (Val)
0.228397585	0.936148301	0.65276811	0.78813559

Results – Dataset 2



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



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

- Train accuracy above 95%
- Test accuracy above 85%

Critical Evaluation

- In both cases, AlexNet was able to get above goal accuracy
- Both data sets worked out extremely well
- However, large dataset took considerably longer to train
- Did not have to make many changes to get better accuracy
- Compared to other works, did not have to do segmentation portion

Conclusion

- Overall, project was very successful
- Exceeded all goals
- Results were very accurate and reliable
- Similar to other real world projects, but a bit more simplified


Alt Text: A bunch of bananas on display in a store
Alt Text: A bunch of bananas on display in a store