William Noone
Machine Learning II – Final Project
Individual Final Report

1. Introduction. An overview of the project and an outline of the shared work.

The diagnosis of blood-based diseases often involves identifying and characterizing patient blood samples. Automated methods to detect and classify cells infected with blood borne diseases like Malaria have important medical applications. Malaria is typically diagnosed by the microscopic examination of blood using blood films. Methods that use the polymerase chain reaction to detect the parasite's DNA have been developed, but are not widely used in areas where malaria is common due to their cost and complexity. Convolution Neural Networks can be applied to this problem to analyze images of cells infected with Malaria. While the priority of our project was to demonstrate convolution methods for image analysis, we also wanted to explore image processing methods, and gauge their effect upon CNN model performance. We sourced data from the official NIH Website:
https://ceb.nlm.nih.gov/repositories/malaria-datasets
The dataset contains 2 folders (Infected, Uninfected) with a total of 27,558 images of various sizes in .png format.

2. Description of your individual work. Provide some background information on the development of the algorithm and include necessary equations and figures.

We used a Convolution Neural Network built in Caffe, to analyze (2) variations of the same cell images: processed (resized 100x100, greyscaled, Gaussian filtered, masked) and unprocessed (resized 100x100). Within these (2) image sets, we split the data (70/30) into training and validation sets. Training the CNN involved creating a multi-layer feedforward network, whereby convolution and pooling operations extracted sets of features maps from respective layers. Following the forward propagation, the loss function (Softmax) was employed to gauge the performance of the binary class label prediction. Backpropagation then updated weights and bias of by calculating the gradient & sensitivities w.r.t. the parameters. This operation series was performed over 20,000 iterations with accuracy typically hovering in the low 90% range.

3. Describe the portion of the work that you did on the project in detail. It can be figures, codes, explanation, pre-processing, training, etc.

My portion of the project was the technical implementation and researching Caffe CNN model structures to maximize performance accuracy. This work can be broken into (2) phases with the associated tasks:

Preprocessing and System Admin

1. File Directory Operations – create script to unzip NIH data download to working directory and export a dataframe indexing image names to class labels in a .csv file
2. Create Unprocessed Images – create a loop to resize raw images and convert from .png to .jpg format

3. Create Processed Images – create a loop to apply image processing packages to raw images (resized, greyscale, Gaussian filter, mask) and convert from .png to .jpg format.
4. Create Training and Validation Directories – write a script to split the image sets into training and validation directories.
5. Create .lmdb packages for the unprocessed and processed .jpg images – write a script to convert .jpg files to .lmdb files in preparation for import into the Caffe framework.

CNN Build & Training

1. Implement CNN for alternative layer structures – created of 2 Layer and 3 Layer CNNs for various blob configuration.
2. Research hyper-parameter settings for performance tuning.
3. Compare results between models and processed vs unprocessed image groups.

4. Results. Describe the results of your experiments, using figures and tables wherever possible. Include all results (including all figures and tables) in the main body of the report, not in appendices. Provide an explanation of each figure and table that you include. Your discussions in this section will be the most important part of the report.

The study aimed to gauge the impact of image processing upon the ability of a CNN to accurately predict binary classifiers. Our approach was to first analyze both underlying image sets (unprocessed, processed) within identical CNN models (2 layer & 3 layer all else equal), then take a high performing image/model combination and fine-tune the hyper-parameters.

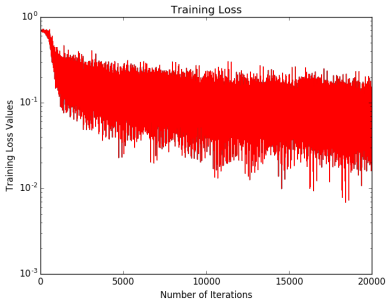
Our first model (Model1_Unprocessed) was a 3 layer CNN (20 FMs per layer, kernel_size = 2, stride =2 across all layers) with (2) fully connected layers and a soft-max loss function. The corresponding model (Model1_Processed) was a 3 layer CNN with kernel_size = 2, stride =2 across all layers with (2) fully connected layers and a soft-max loss function. Using mean test_accuracy as a benchmark, we found that unprocessed images performed best with 91.7% with training loss decay oscillating the tighter bounds (as measure by std dev).

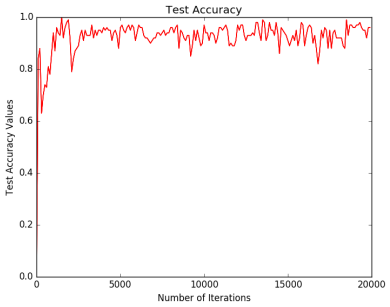
Our next model (Model2_Unprocessed) was a 2 layer CNN (30 FMs per layer, kernel_size = 2, stride =2 across all layers with (2) fully connected layers and a soft-max loss function. The corresponding model (Model2_Processed) was a 3 layer CNN with kernel_size = 2, stride =2 across all layers with (2) fully connected layers and a soft-max loss function. Using mean test_accuracy as a benchmark, we found that unprocessed images again performed best with 91.9% with training loss decay oscillating the tighter bounds (as measure by std dev).

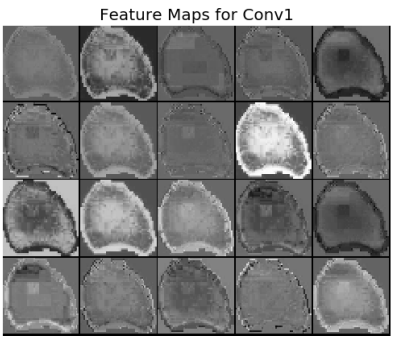
Our final model (Model3_Unprocessed) was a 3 layer CNN with kernel_size = 4, stride =2 across all layers with (2) fully connected layers and a soft-max loss function. After researching multiple configurations, we were able to increase mean test_accuracy to 93.7%.

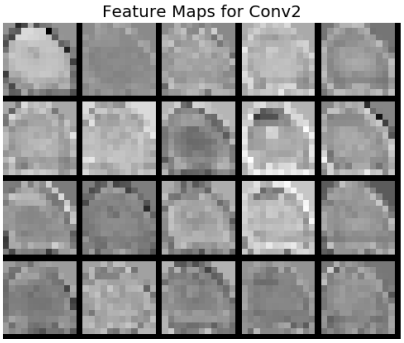
Model1_Unprocessed

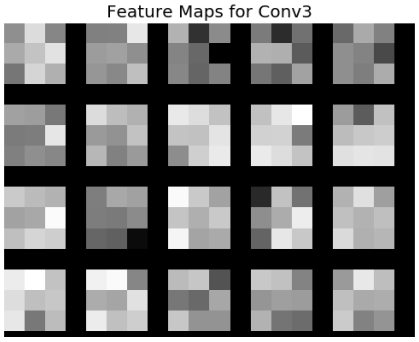
Layer	Shape	Kernel_size	Stride
data	(100, 3, 100, 100)	-	-
label	(100,)	-	-
conv1	(100, 20, 50, 50)	2	2
relu1	(100, 20, 50, 50)	-	-
pool1	(100, 20, 25, 25)	2	2
conv2	(100, 20, 12, 12)	2	2
relu2	(100, 20, 12, 12)	-	-
pool2	(100, 20, 6, 6)	2	2
conv3	(100, 20, 3, 3)	2	2
relu3	(100, 20, 3, 3)	-	-
pool3	(100, 20, 2, 2)	2	2
fc1	(100, 10)	-	-
relu4	(100, 10)	-	-
fc2	(100, 2)	-	-
loss	()	-	-
Weight, Bias	**Shape**	**Measure**	**Value**
conv1	(20, 3, 2, 2) (20,)	Mean Train_Loss	0.13354048
conv2	(20, 20, 2, 2) (20,)	Std Dev Train_Loss	0.109245778
conv3	(20, 20, 2, 2) (20,)	**Mean Test_Acc**	**0.917450002**
fc1	(10, 80) (10,)	Std Dev Test_Acc	0.087166495
fc2	(2, 10) (2,)		

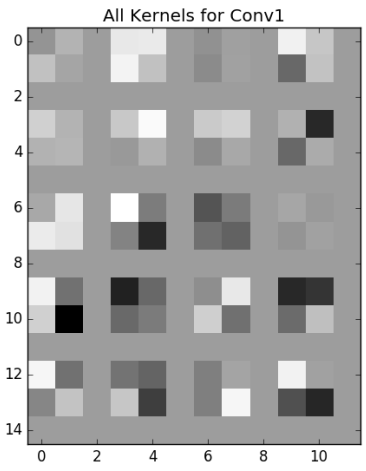


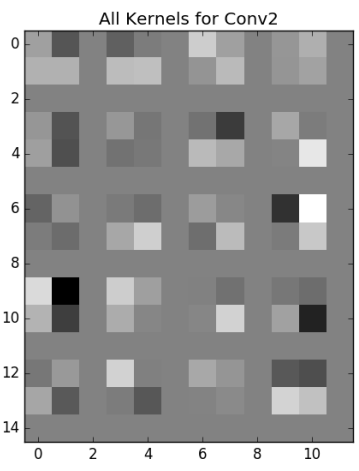


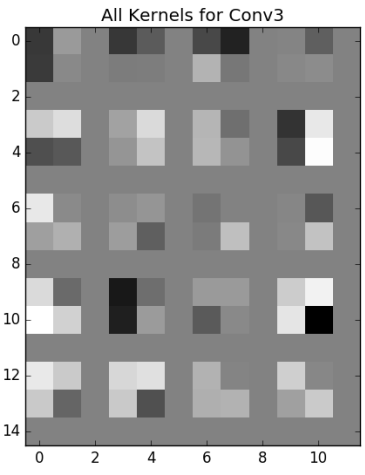








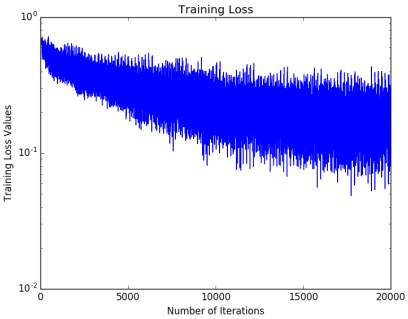


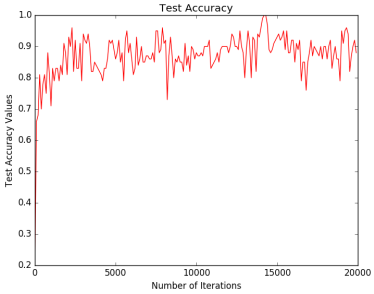


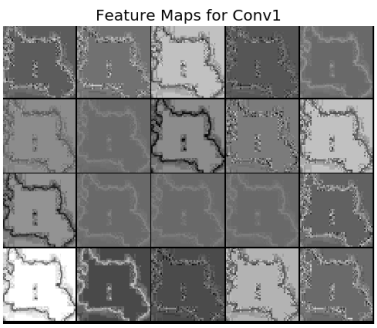
Model1_Processed

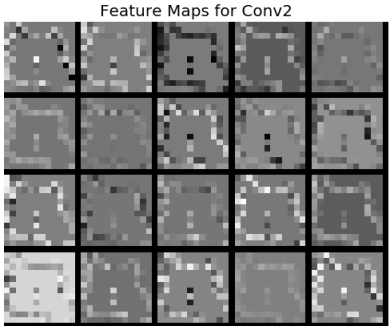
Layer	Shape	Kernel_size	Stride
data	(100, 1, 100, 100)	-	-
label	(100,)	-	-
conv1	(100, 20, 50, 50)	2	2
relu1	(100, 20, 50, 50)	-	-
pool1	(100, 20, 25, 25)	2	2
conv2	(100, 20, 12, 12)	2	2
relu2	(100, 20, 12, 12)	-	-
pool2	(100, 20, 6, 6)	2	2
conv3	(100, 20, 3, 3)	2	2
relu3	(100, 20, 3, 3)	-	-
pool3	(100, 20, 2, 2)	2	2
fc1	(100, 10)	-	-
relu4	(100, 10)	-	-
fc2	(100, 2)	-	-
loss	()	-	-

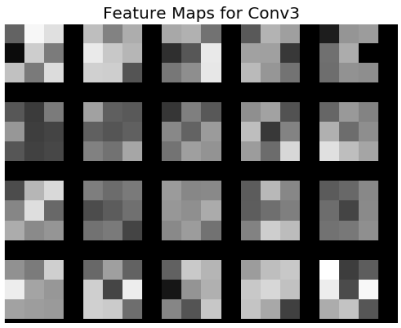
Weight, Bias	Shape	Measure	Value
conv1	(20, 1, 2, 2) (20,)	Mean Train_Loss	0.265589719
conv2	(20, 20, 2, 2) (20,)	Std Dev Train_Loss	0.111700857
conv3	(20, 20, 2, 2) (20,)	**Mean Test_Acc**	**0.8718**
fc1	(10, 80) (10,)	Std Dev Test_Acc	0.072916115
fc2	(2, 10) (2,)		

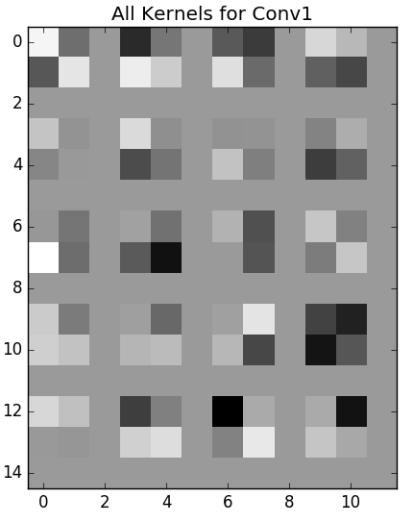


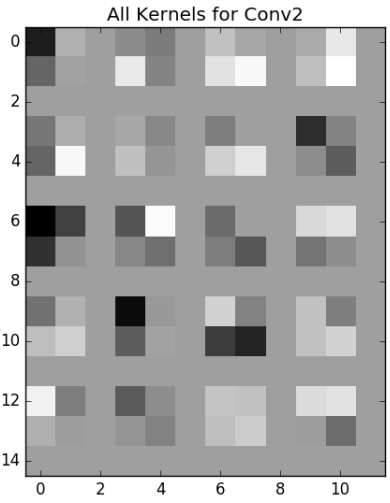


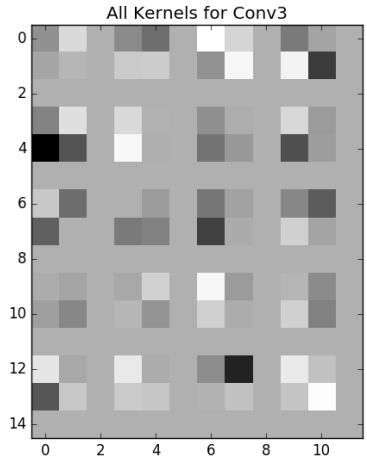






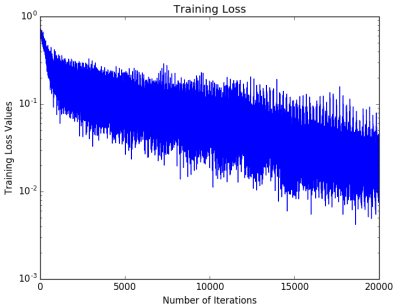


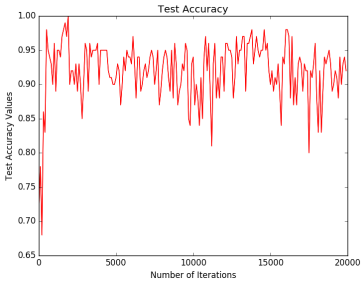


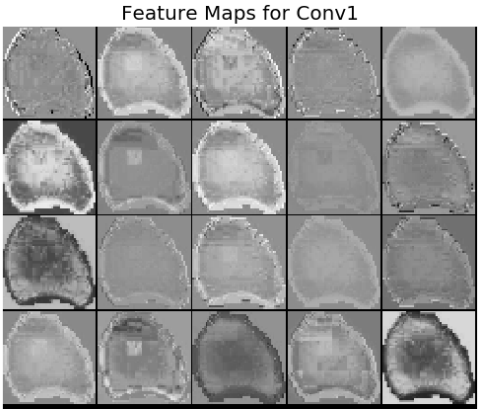


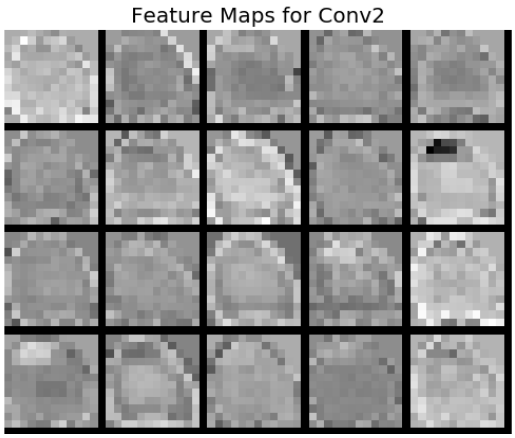
Model2_Unprocessed

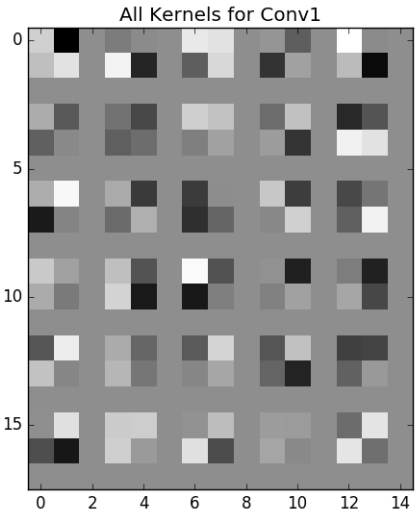
Layer	Shape	Kernel_size	Stride
data	(100, 3, 100, 100)	-	-
label	(100,)	-	-
conv1	(100, 30, 50, 50)	2	2
relu1	(100, 30, 50, 50)	-	-
pool1	(100, 30, 25, 25)	2	2
conv2	(100, 30, 12, 12)	2	2
relu2	(100, 30, 12, 12)	-	-
pool2	(100, 30, 6, 6)	2	2
fc1	(100, 10)	-	-
relu3	(100, 10)	-	-
fc2	(100, 2)	-	-
loss	()	-	-
Weight, Bias	**Shape**	**Measure**	**Value**
conv1	(30, 3, 2, 2) (30,)	Mean Train_Loss	0.08991731
conv2	(30, 30, 2, 2) (30,)	Std Dev Train_Loss	0.087200941
fc1	(10, 1080) (10,)	**Mean Test_Acc**	**0.919999999**
fc2	(2, 10) (2,)	Std Dev Test_Acc	0.043370498

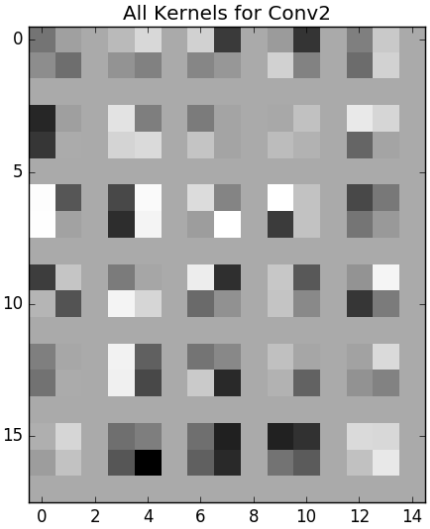






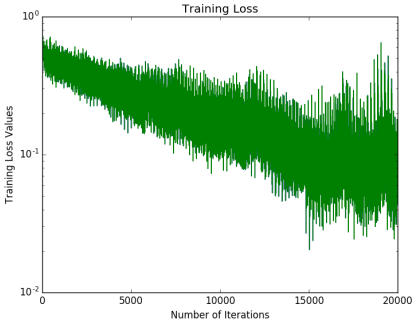


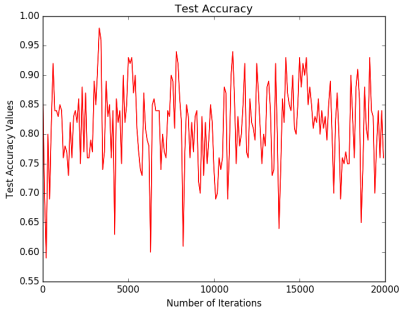


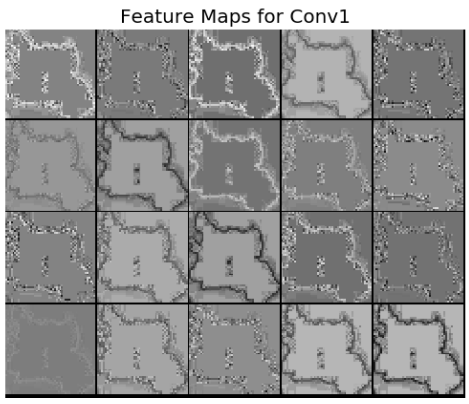


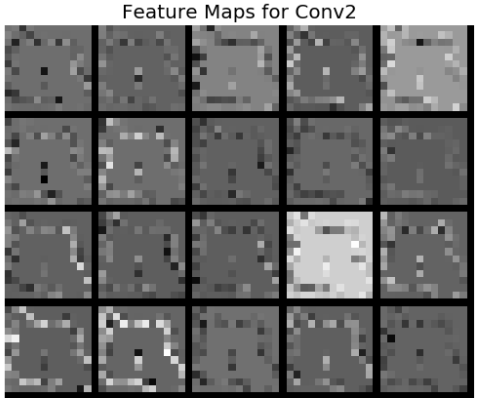
Model2_Processed

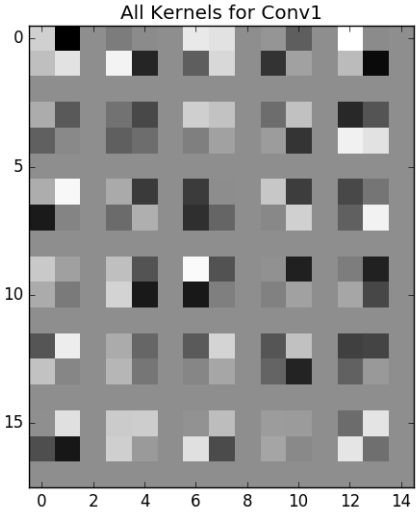
Layer	Shape	Kernel_size	Stride
data	(100, 1, 100, 100)	-	-
label	(100,)	-	-
conv1	(100, 30, 50, 50)	2	2
relu1	(100, 30, 50, 50)	-	-
pool1	(100, 30, 25, 25)	2	2
conv2	(100, 30, 12, 12)	2	2
relu2	(100, 30, 12, 12)	-	-
pool2	(100, 30, 6, 6)	2	2
fc1	(100, 10)	-	-
relu3	(100, 10)	-	-
fc2	(100, 2)	-	-
loss	()	-	-
Weight, Bias	**Shape**	**Measure**	**Value**
conv1	(30, 1, 2, 2) (30,)	Mean Train_Loss	0.213966045
conv2	(30, 30, 2, 2) (30,)	Std Dev Train_Loss	0.124579427
fc1	(10, 1080) (10,)	**Mean Test_Acc**	**0.814999998**
fc2	(2, 10) (2,)	Std Dev Test_Acc	0.07026379

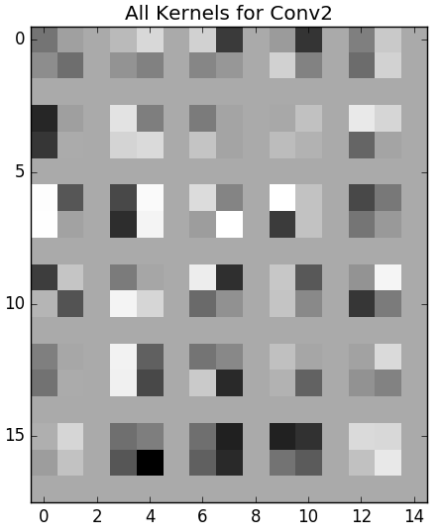






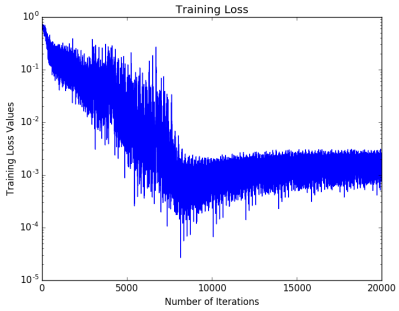


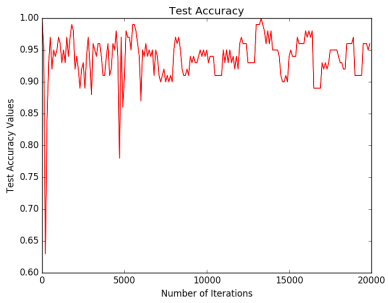


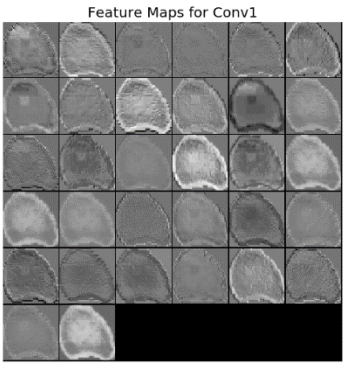


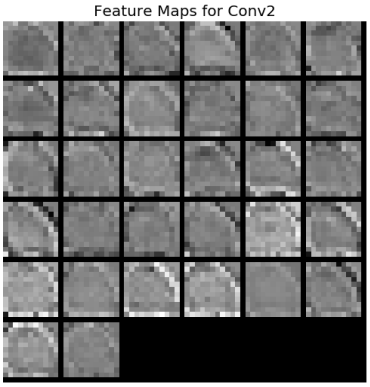
Model3_Unprocessed

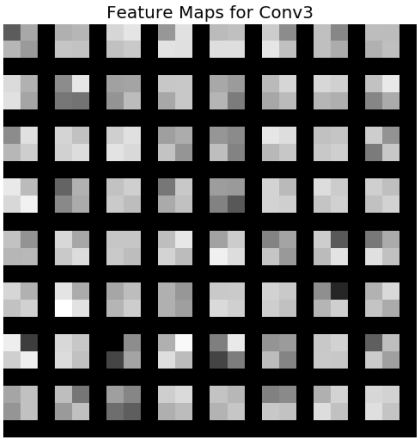
Layer	Shape	Kernel_size	Stride
data	(100, 3, 100, 100)	-	-
label	(100,)	-	-
conv1	(100, 32, 49, 49)	4	2
relu1	(100, 32, 49, 49)	-	-
pool1	(100, 32, 25, 25)	2	2
conv2	(100, 32, 11, 11)	4	2
relu2	(100, 32, 11, 11)	-	-
pool2	(100, 32, 6, 6)	2	2
conv3	(100, 64, 2, 2)	4	2
relu3	(100, 64, 2, 2)	-	-
pool3	(100, 64, 1, 1)	2	2
fc1	(100, 64)	-	-
relu4	(100, 64)	-	-
fc2	(100, 2)	-	-
loss	()	-	-
Weight, Bias	**Shape**	**Measure**	**Value**
conv1	(32, 3, 4, 4) (32,)	Mean Train_Loss	0.031756144
conv2	(32, 32, 4, 4) (32,)	Std Dev Train_Loss	0.081596197
conv3	(64, 32, 4, 4) (64,)	**Mean Test_Acc**	**0.937650002**
fc1	(64, 64) (64,)	Std Dev Test_Acc	0.036850747
fc2	(2, 64) (2,)		

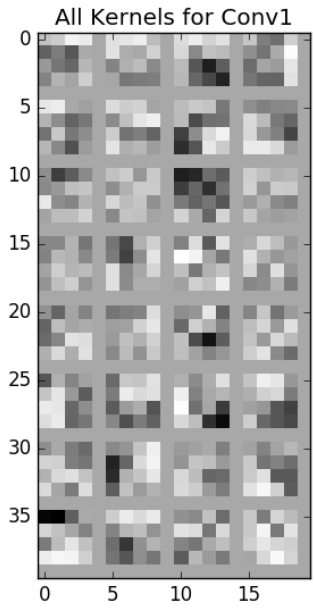


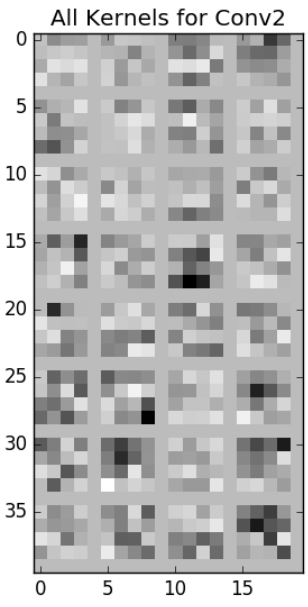


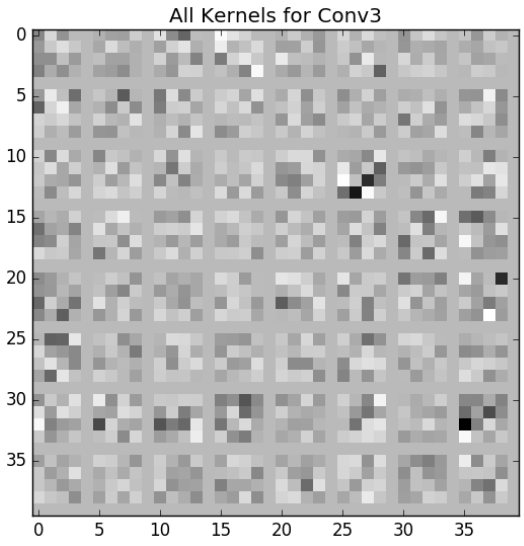












5. Summary and conclusions. Summarize the results you obtained, explain what you have learned, and suggest improvements that could be made in the future.

To our surprise, we found that our image processing techniques had a negative effect upon CNN prediction performance. This could be improved upon by modifying the parameters of the image filtering and masking steps. It is a time intensive process to convert image sets from .jpg to .lmdb format suitable for the Caffe framework though, so additional scenarios were prohibitive w.r.t. time and GCP server resources.

6. Calculate the percentage of the code that you found or copied from the internet.

Original code = 200 lines
Modified code (Dr. Jafari Caffe framework) = 300 per model (applied to 5 models)
Code from other sources = 100 lines (help with image processing, system admin tasks etc)
30% original code, 70-80% original + Jafari code

7. References.

Deep Learning for Medical Image Analysis 1st Edition, S. Kevin Zhou, 2017

Deep Learning and Convolutional Neural Networks for Medical Image Computing, Le Lu, Yefeng Zheng, Gustavo Carneiro, Lin Yang, 2017