Group 12
Members: Ernso Jean-Louis, Willene Nazaire, Aaron Reich

Problem Description and Introduction

We were presented with a task that is rather trivial for humans: determining whether an image is a picture of a dog or of a cat. As it turns out, training an intelligent model to do so is not so trivial. We implemented the classic machine learning workflow: getting our data and processing it, feature selection, partitioning our data for training, cross-validation, and testing, and finally performing training, cross-validation, and testing. Ultimately, we ended up with a model that may not be as good as a human but can certainly give one a run for it's money.

Our Choice of Classifiers

KNN Classifier

We chose the KNN classifier because we didn't want to start with the SVM classifier right away. KNN was not a classifier we discussed at length in class so it offered us the opportunity to do some research about how the classifier worked.

SVM Classifier

We chose the SVM classifier because throughout the semester that it consistently delivered the best results. We thought that using this classifier would help us to achieve the highest accuracy that we could

Workflow and Methodology

We already had our data from the Kaggle data set. After taking a look at the starter code, we also had a means of processing images and extracting features. It was up to us to choose classifiers, partition our data and then train, cross-validate and test our data.

Each individual on the team had some time to play with the starter code data so we could come together and make a choice about the classifiers that we would use. Together continued through the rest of the workflow iteratively to try our different classifiers on small sets of images, troubleshooting issues that came our way. Lastly we were able to chose a single classifier to try on all our images.

Technical Discussion

The most important steps that helped us achieve our goals in this project were the partitioning of the dataset, and determining which classifier to run against the whole data set.

At first, we had a lot of trouble determining how many pictures to pick to do the actual training. One of the group members first tried to do the training with 585 images for cats and 585 images for dogs for the 2 classifiers, which came out total of 1170 images. We encountered some issues because MatLab was not working properly at times. First, we encountered a GPU performance problem, and one of one team member had to email TGA to address that issue. After that issue with the GPU was addressed, we kept running into other issues with MatLab that kept stalling our progress. We debated about increasing that amount of pictures to 15000 (7500 images for cat and 7500 images for dog). We managed to build all the features for the 15000, but because we could do the feature extraction because we got ‘Maximum number of users for Statistics_Toolbox reached. Try again later’ error message in MatLab. Because of that error, we could not run the code to train the multiclass SVM classifier on the features that that were made for the 15000 images. We decided to drop the number of pictures to 5000 (2500 images for cat, 2500 images for dog). The second time we tried to run it with the 5000 pictures, it actually took forever to build the training features.

Design decisions

After some debate within the group, we decided to go with the Cubic SVM classifier, because it was the classifier that ran best from the 1170 images. After we have decided to go with the features from the 1170 images, we had a lot of issues formatting the table the way that it needed to be formatted. There was clearly a turning point when we were able to turn the feature in the classifiers into a table to import into the Classification Learner App (CLA). It was very hard to find the correct code to turn them into the correct table that was needed by the Classification Learner Application to make use of the data. The Cubic SVM classifier was the one that ran against the 25000 images.

Summary table

These classifiers were trained in the Classification App on a set of about 580 images with Hold out validation and 40% left out

Family: KNN		
	Blur	No Blur
Fine	85.7%	86.4%
Medium	84.3%	89.3%
Coarse	75.7%	80.0%
Cosine	92.9%	91.4%
Cubic	83.6%	88.6%
Weighted	87.1%	90.7%

Family: SVM		
	Blur	No Blur
Linear	92.1%	95.7%
Quadratic	93.6%	95.7%
Cubic	95.0%	95.7%
Fine Gaussian	65.7%	66.4%
Medium Gaussian	94.3%	95.7%
Coarse Gaussian	91.4%	91.4%

It wasn't surprising to us that that the KNN classifier family in general performed better with the features extracted from the images that were slightly blurred. It was our understanding that blurring the images would distract the classifier less. It was incredibly surprising however to see that the SVM classifier family performed better with the features extracted from images that weren't blurred. We have surmised though that this may be the case because the SVM classifier works to create a boundary that is the farthest away from both classes. Blurring the images may have made this more difficult

since it artificially enlarges the areas which it will use to create a boundary. This in turn created a boundary that is closer to each of the classes which may cause errors.

After a discussion in class about how bursts of colors or some other patterns may distract from more important features of an image, we decided to add a slight blur to images as a part of processing.

Experiments and Results

We created two classifiers resulting from feature extraction from 585 images of cats and 585 images of dogs (1,170 images). We partitioned the 1,170 image data set into 52.5% for training, 17.5% for cross-validation, and 30% for testing. The classifiers are a CosineKNN and a CubicSVM. The Cosine KNN makes medium distinctions between classes and utilizes a Cosine distance metric while setting the number of nearest neighbors to be 10. The CubicSVM finds the optimal hyperplane to separate various data points of one class from another class along with using the Cubic Kernal function to compute the classifier. We also created a CubicSVM classifier from feature extraction from 25,000 images, 12,500 images of cat images and 12,500 images of dogs. We partitioned the 25,000 image data set into 60% for training with 5 fold cross validation and 40% for testing.

When comparing the two classifiers after training and cross-validation produced from the feature extraction of 1,170 images, the CubicSVM clearly outperformed the CosineKNN. As shown by the Confusion Matrices, dog was mislabeled as a cat six times by the CosineKNN, but only mislabeled as a cat three times by the CubicSVM. Cat was mislabeled as a dog two times by both classifiers. The AUC of the ROC graph for the CubicSVM was 0.98 while the AUC of the ROC graph for the CosineKNN was only 0.97. While these ROC graphs and AUC values both demonstrate a clear and effective discrimination between the two classes, the CubicSVM did outperform the CosineKNN by 0.01.

When comparing the two classifiers after testing produced from the feature extraction of 1,170 images, the CubicSVM clearly outperformed the CosineKNN again. As shown by the Confusion Matrices, dog was mislabeled as a cat 11.25% of the time and cat was mislabeled as a dog 1.47% of the time by the CosineKNN classifier. This can be compared to the CubicSVM where dog was mislabeled as a cat 4.16% of the time and cat was mislabeled as a dog 1.22% of the time. So dog was mislabeled as a cat 7.09% more by the CosineKNN classifier. Cat was mislabeled as a dog 0.25% more by the CosineKNN classifier compared to the CubicSVM. Dog images are clearly mis-classified as cats far more than cat images are mis-classified as dogs. The AUC of

the ROC graph for the CubicSVM was clearly greater than the AUC of the ROC graph for CosineKNN.

The CubicSVM classifier from the feature extraction of 1,170 images can be compared with itself from training and cross-validation to its performance when testing. It mislabeled a cat as a dog 4.54% of the time during training and cross-validation but only 1.22% of the time during testing. It mislabeled a dog as a cat 6.81% of the time during training and cross-validation but only 4.16% of the time during testing. So it mislabeled a cat as a dog 3.32% of the time less during testing than in training and cross-validation. It mislabeled a dog as a cat 2.65% of the time less during testing than in training and cross-validation. So it had better improvement from training and cross-validation to testing when it came to classifying a cat than when it had to classify a dog.

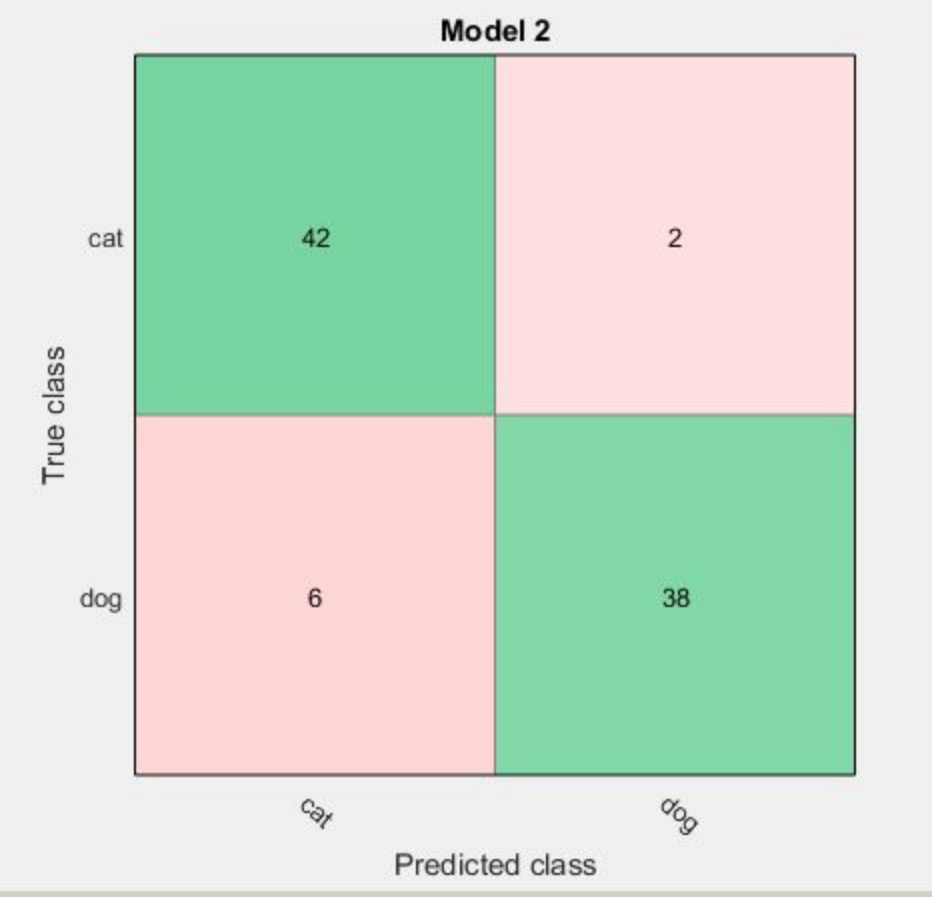
The CosineKNN classifier from the feature extraction of 1,170 images can be compared with itself from training and cross-validation to its performance when testing as well. It mislabeled a cat as a dog 4.54% of the time during training and cross-validation but only 1.47% of the time during testing. It mislabeled a dog as a cat 13.6% of the time during training and cross-validation but only 11.25% of the time during testing. So it mislabeled a cat as a dog 3.07% of the time less during testing than in training and cross-validation. It mislabeled a dog as a cat 2.35% of the time less during testing than in training and cross-validation. So CosineKNN also had better improvement from training and cross-validation to testing when it came to classifying a cat than when it had to classify a dog.

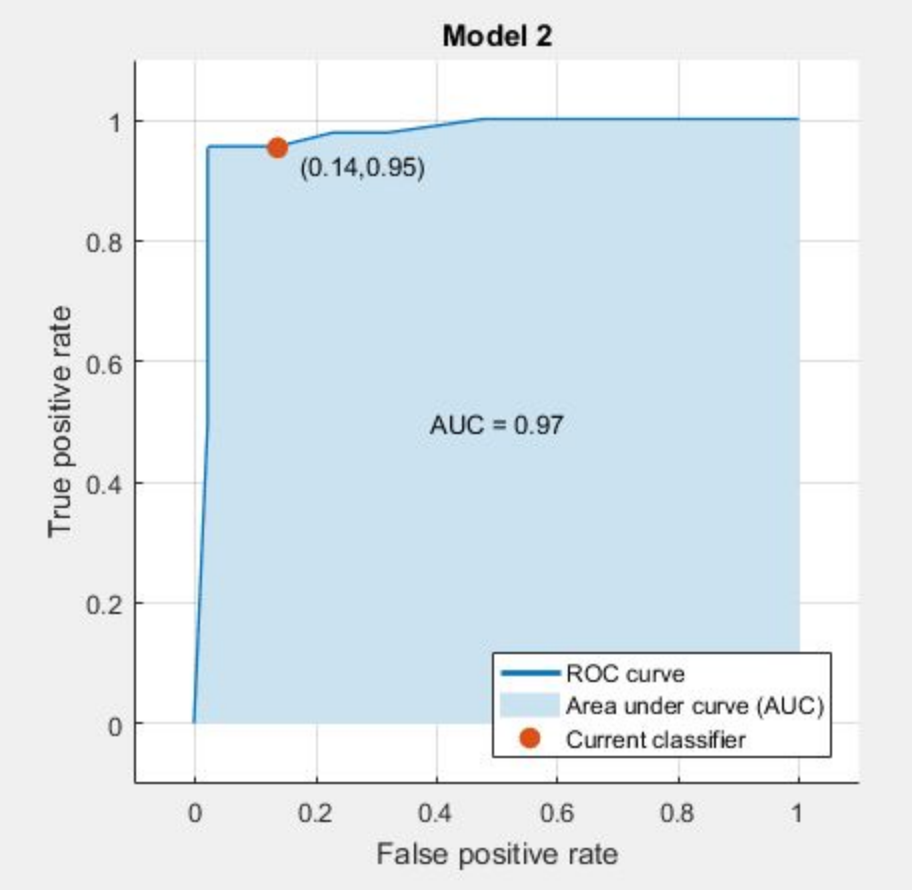
We used the CubicSVM for the classifier from feature extraction from 25,000 images because this was clearly the better performing classifier. It mislabeled a cat as a dog 3.90% of the time during training and cross-validation but only 3.66% of the time during testing. It mislabeled a dog as a cat 4.65% of the time during training and cross-validation but only 4.62% of the time during testing. When trying to compare the training and cross-validation ROC plot with the test ROC plot, it is hard to visually be able to decipher which curve is farther away from the 50% chance line going from the bottom left to the upper right corner of the plot. However from the Confusion Matrix, we are able to realize that there it mislabeled a cat as a dog 0.24% of the time less during testing than in training and cross-validation. It mislabeled a dog as a cat 0.03% of the time less during testing than in training and cross-validation. It can be noted that there is much less improvement from training and cross-validation to training than the CubicSVM classifier with 1,170 images. But this can be attributed to the difficulty in making improvements at such a higher level of accuracy.

Tables and Plots

CosineKNN Classifier from Training and Cross-Validation from 1,170 Images

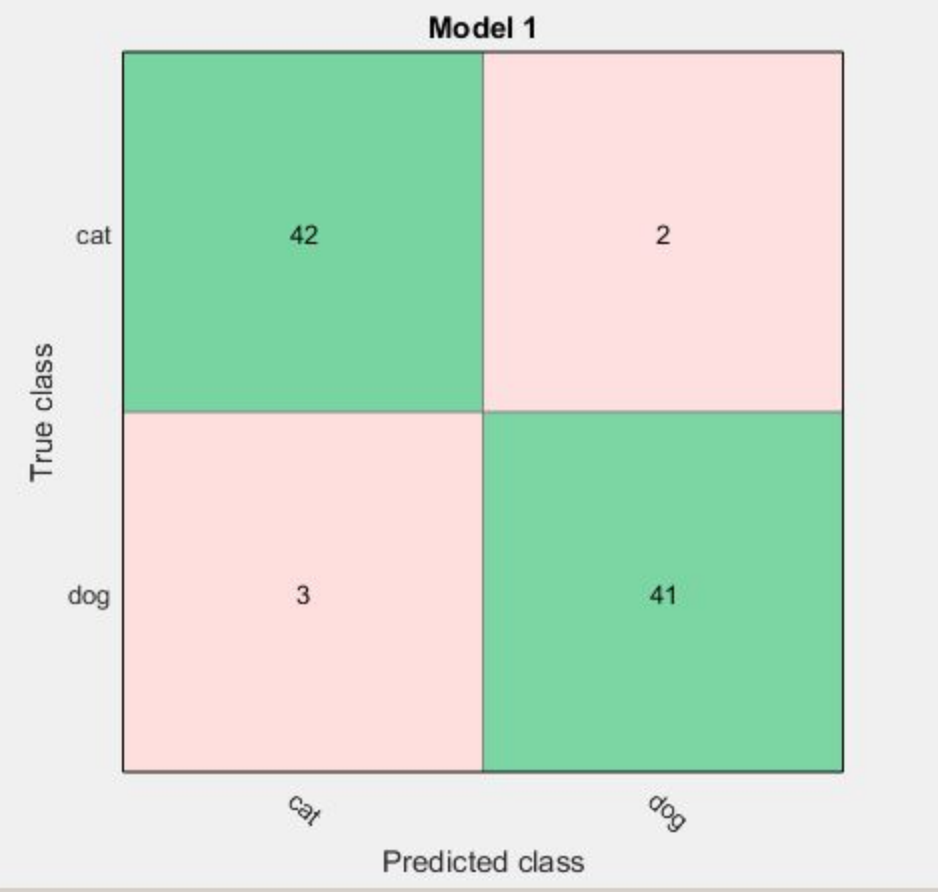
```
confMat =

    0.9886    0.0114
    0.1250    0.8750

Warning: Converting input data to single.
> In pdist2 (line 232)
  In ExhaustiveSearcher/knnsearch (line 207)
  In ClassificationKNN/score (line 425)
  In ClassificationKNN/predict (line 703)
    0.9880
```

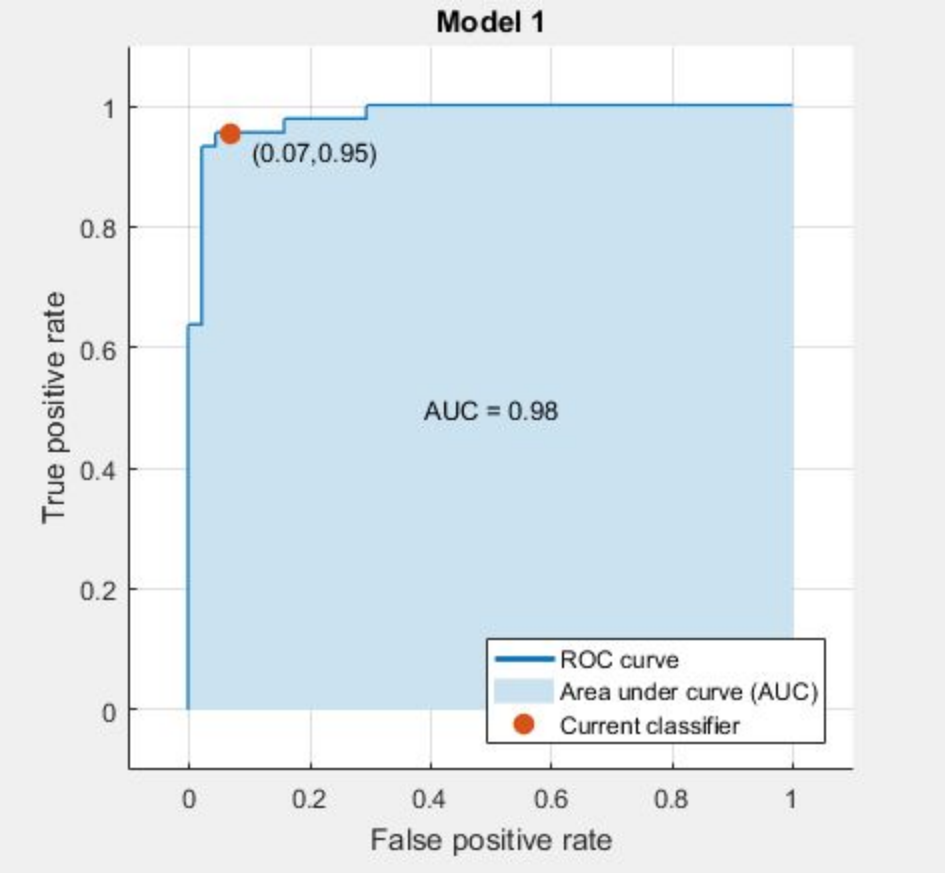




CubicSVM Classifier from Training and Cross-Validation from 1,170 Images

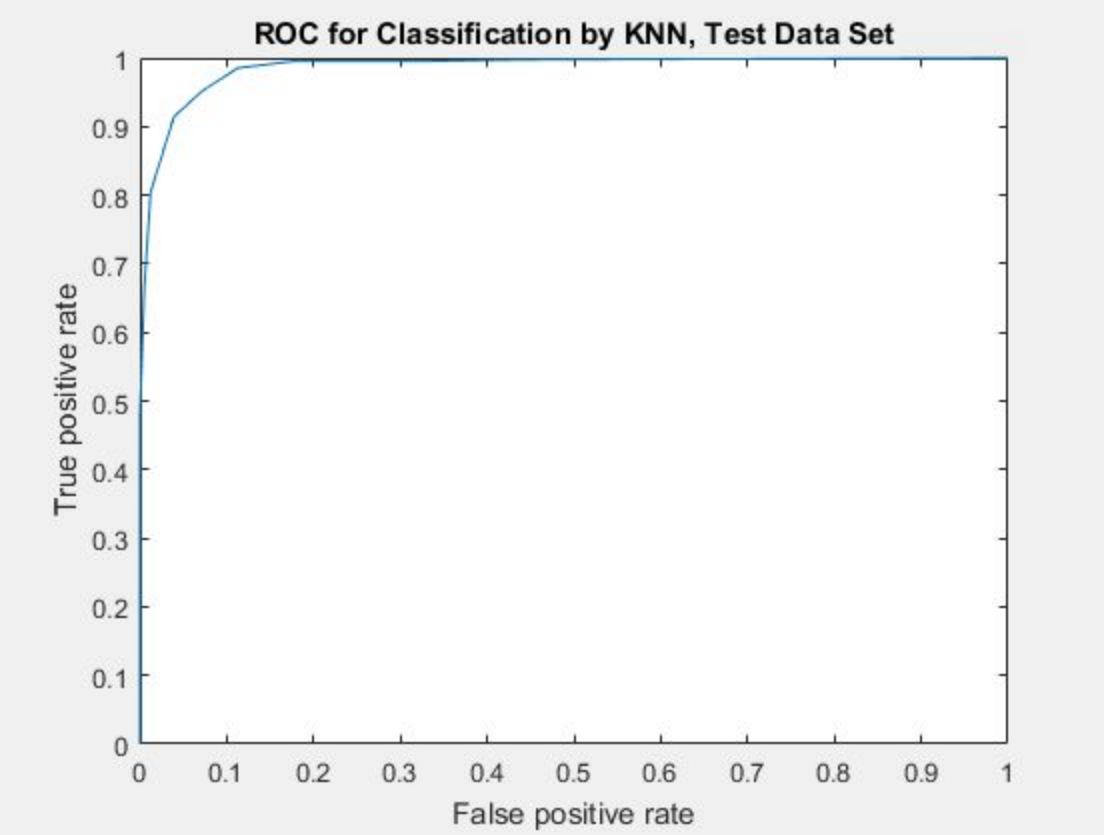
```
confMat =

     1     0
     0     1

     1
```


Model 1
cat
dog
True class
42
2
3
41
cat
dog
Predicted class



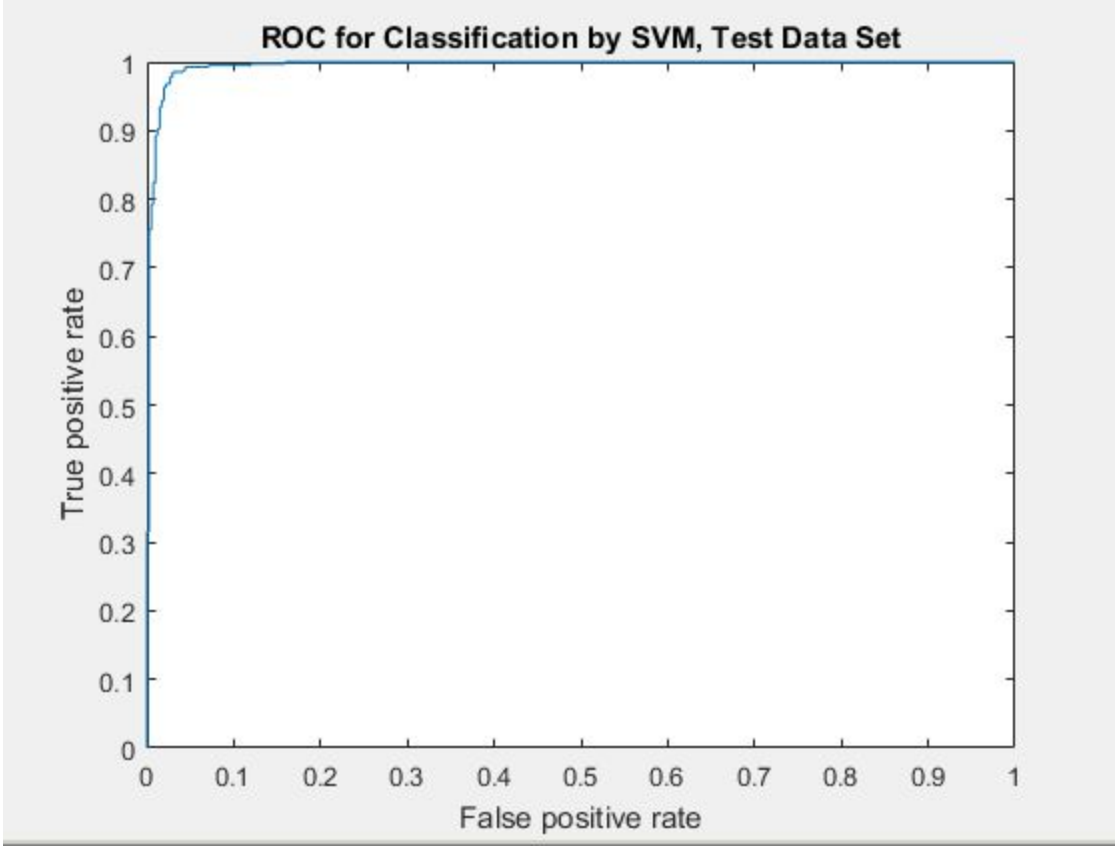
CosineKNN Classifier from Testing from 1,170 Images

```
confMat1 =

    0.9853    0.0147
    0.1125    0.8875

Warning: Converting input data to single.
> In pdist2 (line 232)
  In ExhaustiveSearcher/knnsearch (line 207)
  In ClassificationKNN/score (line 425)
  In ClassificationKNN/predict (line 703)
    0.9863
```



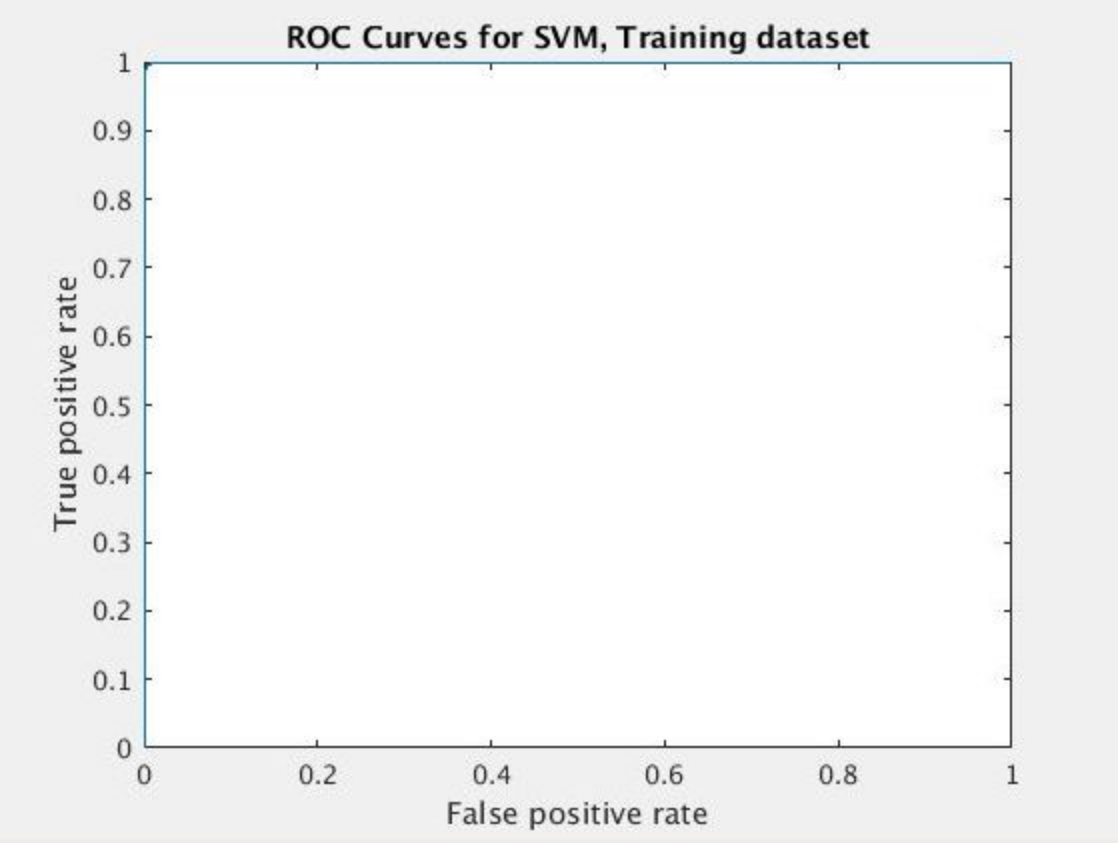
CubicSVM Classifier from Testing from 1,170 Images

```
confMat1 =

    0.9878    0.0122
    0.0416    0.9584

    0.9949
```



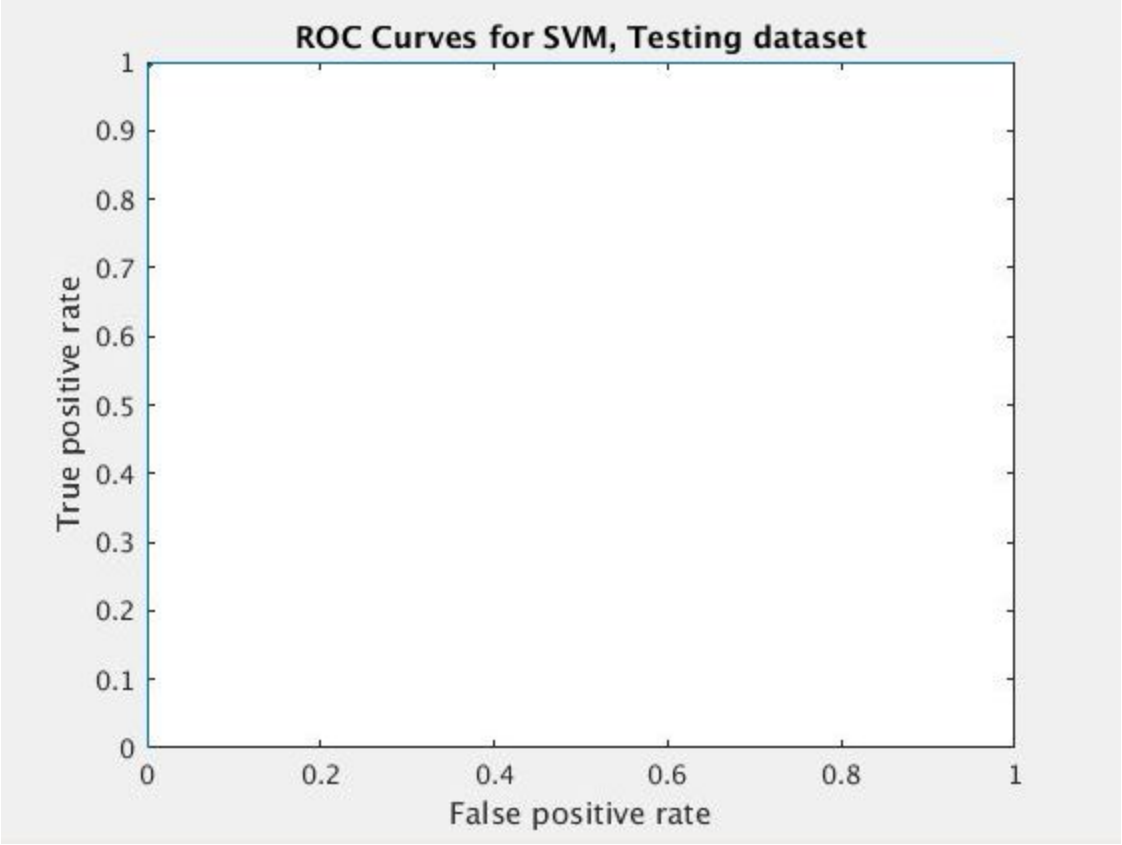
CubicSVM Classifier from Training and Cross-Validation from 25,000 Images

```
    7207         293
     349        7151

0.9998
```



CubicSVM Classifier from Testing from 25,000 Images

```
    4817         183
     231        4769

0.9997
```


ROC Curves for SVM, Testing dataset
True positive rate
False positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1

Answers to Starter Code Problems

#4. The auxiliary function readAndPreprocessImage is used to prepare the images to be used with the CNN by using the filenames in Files and converting to RGB if necessary and crop.

#5. The montage with network weights for the second convolutional layer displays what the CNN has seen before so primitive patterns such as striped bright and dark lines in different directions and patches of color such as green, red, purple, blue, and more.

#6. The other function that could have been used to build the SVM classifier in this binary classification problem case is the fitcsvm MATLAB function.

#7 Yes my classifier recognized 'Doge' as a dog.

Closing Remarks

This assignment was a wonderful way to apply concepts that we have learned all semester. We were excited as a team to create an “intelligent” model. While this assignment was fraught issues that were completely unanticipated, we were able to rally together as a class to solve them. Through this experience, we have become more educated about the machine learning workflow, have been extremely humbled by Matlab, but emerge as better problem solvers and machine learning practitioners.

README

We have included 3 scripts in our .zp file: CosineKNNClassifierScript.m, CubicSVMClassifierForAllImagesScript.m, and CubicSVMClassifierScript.m. These files have the scripts for different classifiers preformed on different sets. CosineKNNClassifierScript.m has a Cosine KNN classifier that was trained on a smaller set of 580 some images. Likewise, CubicSVMClassifierScript.m uses a Cubic SVM classifer that was trained on the same set of smaller images. With the SVM classifier performing the best, we used that classifier to train all 25,000 images which is in the CubicSVMClassifierForAllImagesScript.m file.

- All file are dependant on readAndProcessImages which blurs the images slightly in addition to processing them.
- CosineKNNClassifier.mat and CubicSVMClassifierScript.mat contain the struct for the classifier that befits their name.

- trainedSVMClassifier.m contains the code for its classifier
- CosineKNNClassifierScript.m and CosineKNNClassifierScript.m expect the cat and dog images to be placed in data/SomePetImages, separated into different folder for cats and dogs
- CubicSVMClassifierForAllImagesScript.m expects the cat and dog images to be places in data/train, separated into different folder for cats and dogs
- Empty folders designating where images should have been included
- The features each script uses has been included

After the workspace is configured properly, simply run any of the scripts: CosineKNNClassifierScript.m, CubicSVMClassifierForAllImagesScript.m, and CubicSVMClassifierScript.m