“It Depends”
Presents
“Deciphering Doggies:
An Image Classification Project”

Adam Burrows
Nick Zamora

Table of Contents

Background 3

Motivation 4

Previous Research 5

Convolutional Neural Networks 6

Goals of Analysis 7

Data Source 8

Preliminary Analysis 8

Analysis Results 10

Recommendations & Future Direction 13

Appendix 15

- **Code**
- **Other Models**

Background:

During our Master of Science in Data Analytics (MSDA) program, we have been introduced to, implemented, and created several types of predictive models. These models are useful in quickly giving a prediction or classifying a dependent variable when given an independent variable. With the help of a statistical program, one can write a few lines of code, and get a copious amount of statistics that can help explain his or her data. For these types of algorithms, an average computer far outperforms the smartest of humans in computational speed, accuracy, and efficiency. However, there are still a few tasks that humans are still superior than computers. One of these tasks is Image Recognition.

For an average person, distinguishing objects from one another is almost an effortless task. Our highly developed brains enable us to identify a pen from a pencil, a Ford truck from a Chevy truck, and a a pizza from a cake with relative ease. In turns out that these tasks become much more complicated and daunting when using a computer. Luckily there have been recent breakthroughs in this field to help improve a computer's ability to complete these tasks. TensorFlow, an open-source software library for dataflow programming across a range of tasks, had the following to say about the recent developments in Image Recognition:

"Researchers have demonstrated steady progress in computer vision by validating their work against ImageNet -- an academic benchmark for computer vision. Successive models continue to show improvements, each time achieving a new state-of-the-art result: QuocNet, AlexNet, Inception (GoogLeNet), BN-Inception-v2. Researchers both internal and external to Google have published papers describing all these models but the results are still hard to reproduce."

The purpose of this paper is to show how we solved an image classification problem. To help us solve this problem, we went to Kaggle and discovered there was an ongoing image classification competition which challenges "Kagglers" to build a model that accurately classifies pictures of dogs into their respective "class" or breed. In addition to our main goal of building a model with a high classification rate, we hope to answer other questions such as: which dog breed is most common, which dog breed is the easiest/hardest to classify, and which two dog breeds are mistaken for each other the most. Next, we will go into more detail about why we chose to solve this problem.

Motivation:

According to the Kaggle competition, the goal is to see "how well you can tell your Norfolk Terriers from your Norwich Terriers? With 120 breeds of dogs and a limited number training images per class, you might find the problem more, err, ruff than you anticipated." Despite this competition having a relatively insignificant purpose, Image Recognition is an important topic in many fields. Just some of the fields that can benefit from Image Recognition are Science/Medicine, Agriculture, Finance, and Astronomy.

Our motivation to solve this problem stemmed from Image Recognition having a multitude of applications. Like many of our peers, we entered the MSDA program with the hope of gaining experience and skills in a field that is growing and in high demand. However, to stand out amongst our peers in the eyes of potential employers we need to have a unique experience and/or skill. Knowing that Image Recognition was not covered in our Data Analytics program, we saw an opportunity to gain a unique experience and skill that would differentiate us from our peers.

If we could solve this problem with respectable results, we could showcase to potential employers that we are active learners that are willing to take on new, challenging tasks. Furthermore, by solving this problem we would increase our marketability to potential employers in various fields, thus exponentially increasing our chances of getting an interview and securing a full-time position. After reviewing all these examples of how we could benefit from this type of project, we decided that they outweighed the challenges we would undoubtedly experience.

It should also be noted that being able to look at thousands of dog pictures *may have* played a factor in our decision to choose this project.

Previous Research:

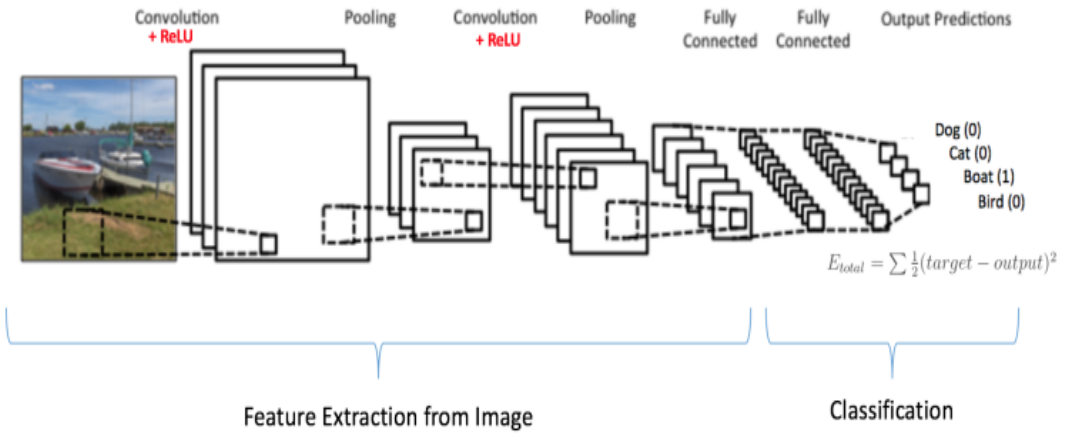
Leading up to this project, we wanted to gather more information about the tools and techniques needed to analyze these pictures. We began to conduct searches into statistical programs such as TensorFlow and Keras and searches into tools and techniques such as deep learning and neural networks. The following statement was made in an academic paper published by Cornell University:

"In the last couple of years, deep learning has led to very good performance on a variety of problems, such as visual recognition, speech recognition and natural language processing. Among different types of deep neural networks, convolutional neural networks have been most extensively studied. Leveraging on the rapid growth in the amount of the annotated data and the great improvements in the strengths of graphics processor units, the research on convolutional neural networks has been emerged swiftly and achieved state-of-the-art results on various tasks" (Gu, Wang, Kuen).

Convolutional Neural Networks:

The overall goal of this project was to build a model that would help to identify the breed of a dog by using just the dog's picture. However, to accomplish this goal, we realized there would be a lot of smaller goals that would need to be met along the way as well. The first goal was to conduct more research into our model's method of using a type of neural network called a convolutional neural network (CNN). Upon completing this research, we gained a better understanding about how these CNN's worked, and how to interpret the possible results of our data set.

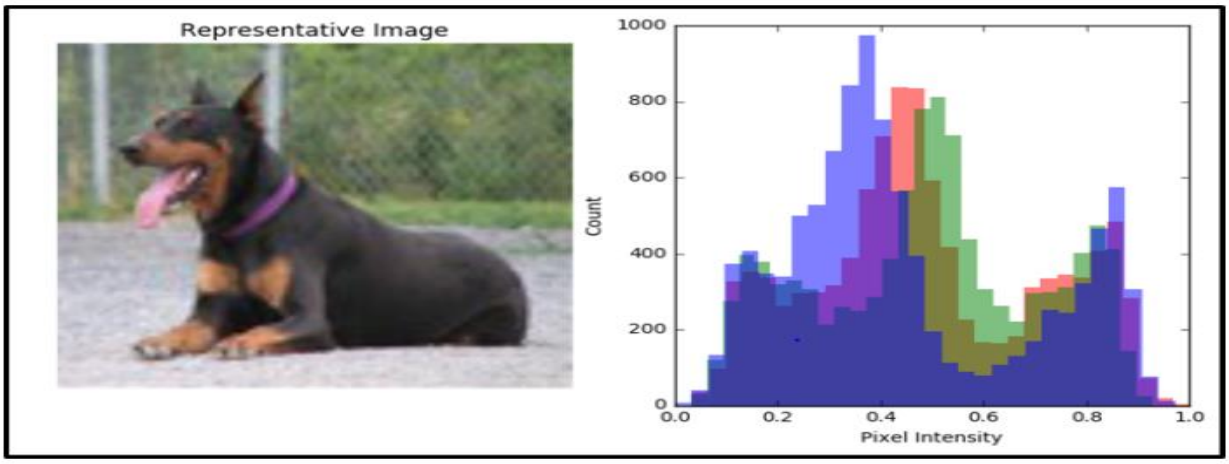
After reading several articles and "how to" guides, we were able to break down our goals into simpler, more practical steps. The steps included building a model that accurately classifies our training data, assess our model with our test set, and repeat the first two steps with differing parameter estimates to find the best performing model. Below is a visual representation on how a CNN works.



From this visual representation, we can gather that there are about 5 steps to train the CNN before it can used to classify the data in the test set. The 5 steps can be seen in the list below:

Step1: We initialize all filters and parameters / weights with random values.

Step 2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class. The output for one of the training images from our data set can be seen below.



Step 3: Calculate the total error at the output layer (summation over all 10 breeds)

Total Error = $\sum$ ½ (target probability – output probability) ^{2}

Step 4: Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error. The weights are then adjusted in proportion to their contribution to the total error. When the same image is input again, the output probabilities might change and make them closer to the target vector.

Step 5: Repeat steps 2-4 with all images in the training set.

Goals of the Analysis:

Despite these obstacles and our low domain knowledge of CNNs, we were confident that we could still make a model that would classify accurately. We hoped for a goal of 90%

accuracy, however we deemed it necessary to set a more realistic goal of 75%. Along with this goal, we hoped to gain knowledge in a field that was previously foreign to us. By taking on this challenge, we could show our fellow peers and potential employers that we were willing to and able to take on difficult new tasks, learn quickly, manage time effectively, and produce useful results that were easy to interpret.

Data Source:

Our data set came from a competition from Kaggle called "Dog Breed Identification". The original data was pre-separated into a training set and a test set each with 10,222 photos of dogs. However, due to this data set coming from an active Kaggle competition, only the training set contained label of the correct breed in addition to the image. The test set did not contain labels for the dog images.

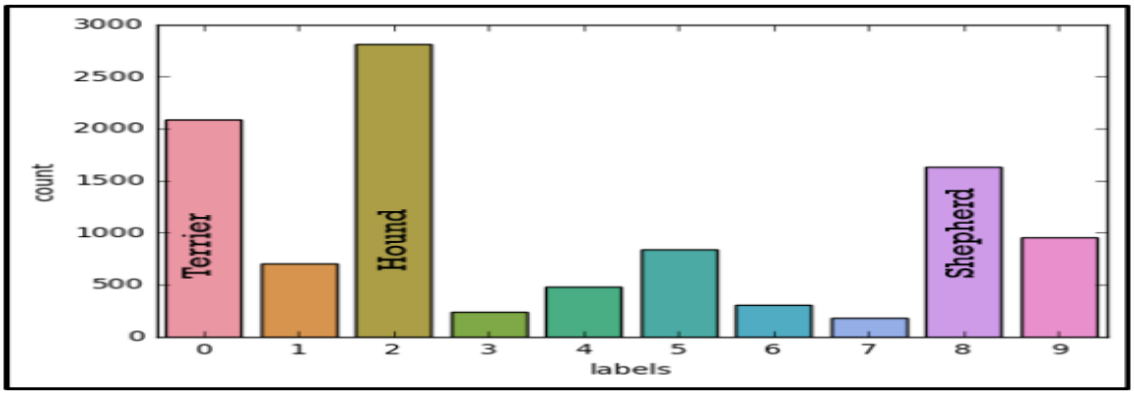
Preliminary Analysis:

This presented our first major problem and decision for us, because without correct labels, we would have no way of determining how well our model was predicting. At this point, we decided to exclude our "test set" images, deeming them no longer useful to our analysis. Then, we conducted a random 80:20 split of the 10,222 images in the original training set to create a new training and test set.

The second problem that we needed to address was that there was not a constant pixel count for each of the pictures in the data set. To be consistent, and create an un-biased model, we decided that we needed to convert them all to the same size (100x100 pixels). After doing some more research, we were able to write code that would automatically take the images in our data set and convert them all into the same 100x100 pixel size. The technique used for the image resizing is called Antialiasing, which is a high-quality down sampling filter.

Our third major problem was having to deal with there being 120 different breeds in our data set. In order to build a more practical and useful model, we decided that re-classifying these 120 breeds into 10 generic dog “breeds” would yield the best results. Unfortunately, there was no foreseeable way to automate the re-classification process, and therefore it was done manually. The csv file containing labeled images did not contain dog breed characteristics, which would have improved the re-classifying by organizing the dogs by features. The following is the list of the 10 generic dog breeds that were used in the model: Terrier, Retriever, Hound, Poodle, Bull, Spaniel, Schnauzer, Husky, Shepherd, and Toy.

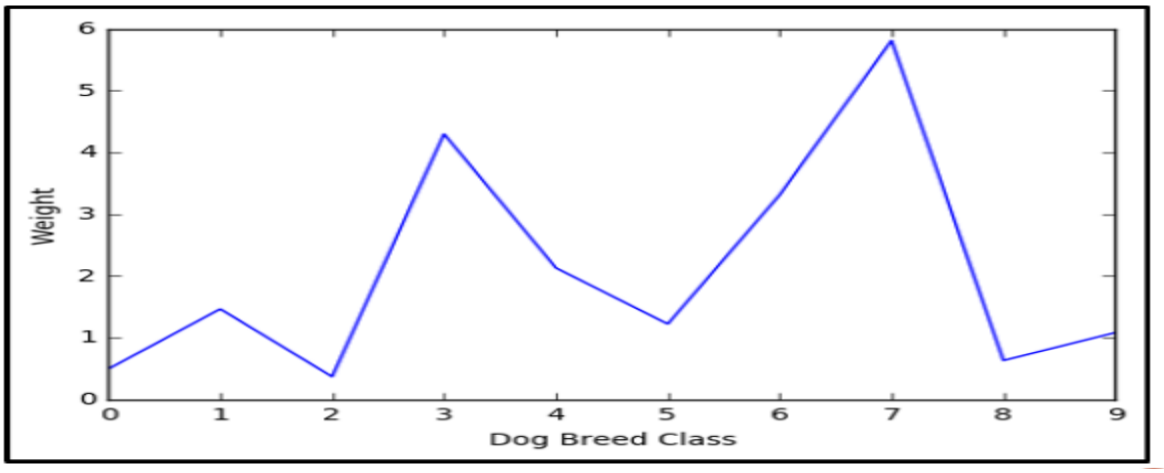
The fourth, and final major problem that we needed to overcome was the resulting effect of re-classifying the images. After reclassifying, we had a large disparity between the frequencies of some of the new dog breeds compared to others. Below is a frequency distribution of the dog breeds.



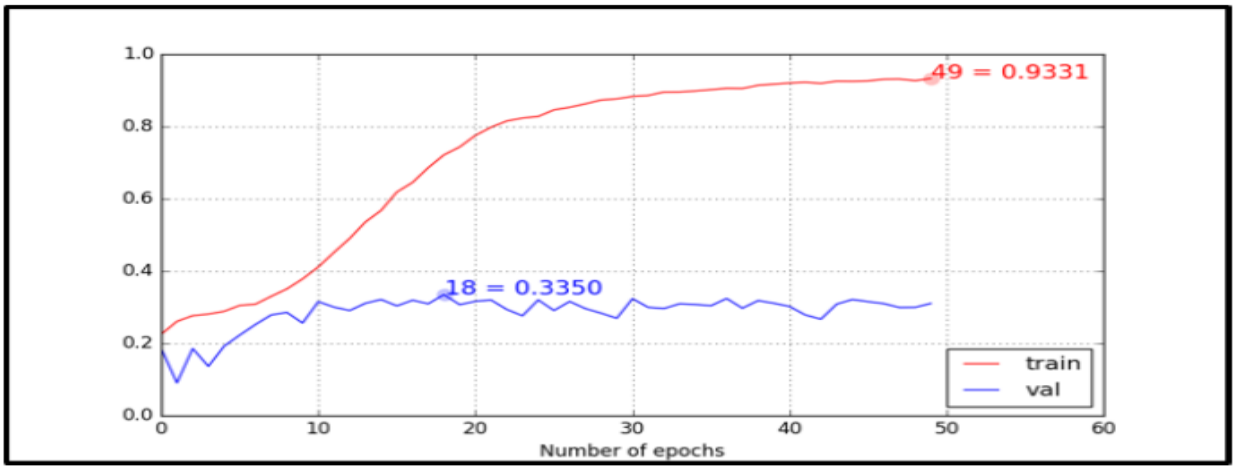
As you can see from the graph, the top three new breeds in terms of frequency, Hound, Terrier, and Shepherd, made up over 62% of the data. Because of this, we decided that we would need to make an adjustment either by using an under or over sampling technique or by assigning weights to each of the breeds depending on their frequencies.

Analysis Results:

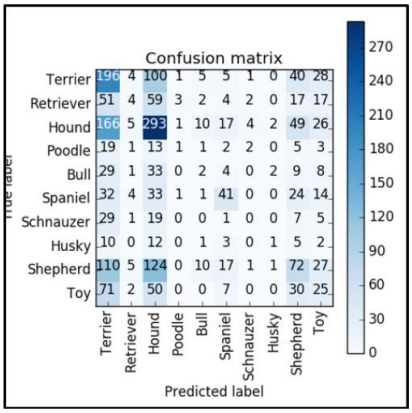
After discovering the imbalance of frequencies, we decided to continue by assigning weights for each of the dog breeds. This would avoid us from having to explain the unknown effects the under and over sampling techniques might have on our model. The graph of the weights associated with each of the breeds can be seen below.



At this point, we were ready to proceed to building our model. We decided that we would build at least two different models using different percentage splits for the training and test sets and different number of epochs (rotation through each image in the training set). By using these two models we hoped we could find one that would get us to our realistic goal of 75% accuracy. The first model we decided to train and test was a CNN that utilized an 80:20 split and ran through 50 epochs each. With access to a Lambda Quad supercomputer with 4 GPU processing units, each epoch took 8 seconds, resulting in total run time of under 7 minutes. Unfortunately, our results were less than we had hoped for. A graphical representation of our results can be seen below.



As you can see from the graph, our model did a great job in classifying our training set, but also produced a much lower accuracy rate than we originally had hoped for. We believe that this variance could be caused by the high variance within each of the dog breeds and our model over-fitting to our test set causing the results of the training set to suffer. Another explanation could be that our model focused its attention on trying to correctly classify the top three dog breeds, Hound, Terrier, and Shepherd. Further proof of this point can be seen in the confusion matrix and table shown below.

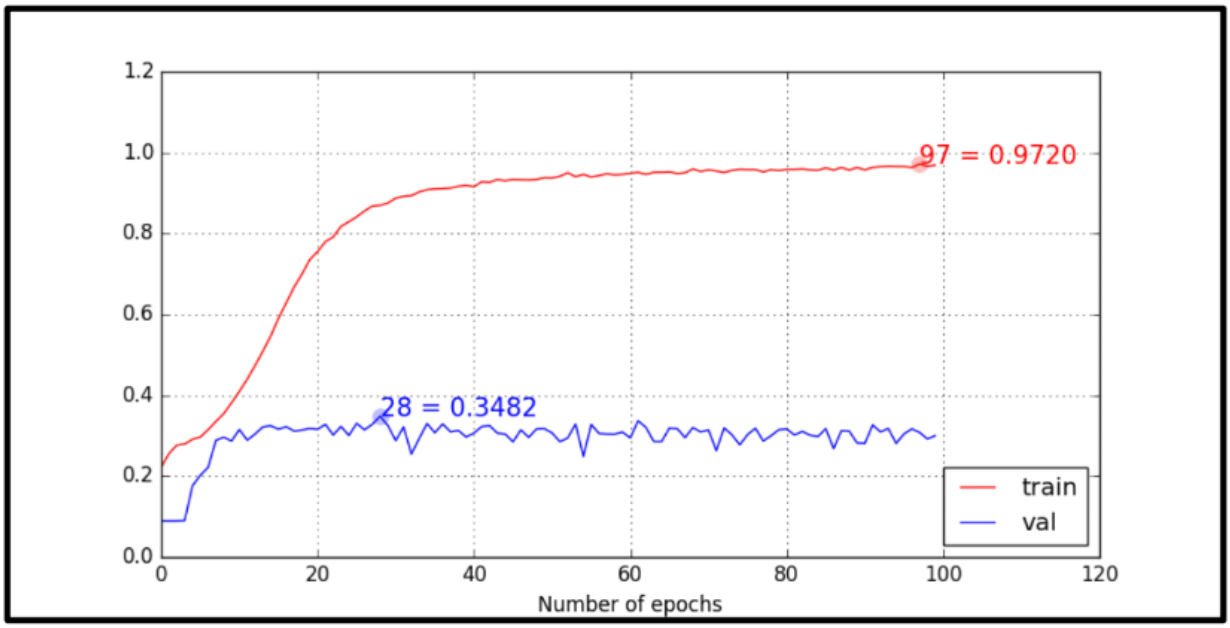


	precision	recall	f1-score	support
Terrier	0.27	0.52	0.36	380
Retriever	0.15	0.03	0.04	159
Hound	0.40	0.51	0.45	573
Poodle	0.14	0.02	0.04	47
Bull	0.06	0.02	0.03	88
Spaniel	0.41	0.27	0.33	150
Schnauzer	0.00	0.00	0.00	62
Husky	0.17	0.03	0.05	34
Shepherd	0.28	0.20	0.23	367
Toy	0.16	0.14	0.15	185
avg / total	0.28	0.31	0.28	2045

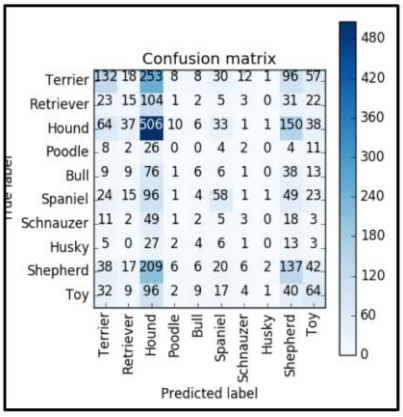
From the confusion matrix we were able to confirm that most images that were correctly classified came from the three most popular breeds. In addition, by looking at the table we can see that 3 of the top 4 performing breeds in terms of precision and recall were the top three most

frequent breeds. Despite the top breeds having somewhat respectful results, our total f1-score was a mere 0.28, far below what was hoped for.

At this point, we decided to train and test our second model in hopes of getting closer to our accuracy goal. This model would utilize a 70:30 split and run through 100 epochs each. The total run time for this model was barely 13 minutes. Unfortunately, our results resembled a lot like the first model. The graph below shows our results in more detail.



As you can see from the graph, this model had the same characteristics as the first model. It did a great job in classifying our training set, but also produced a much lower accuracy rate than we originally had hoped for. However, this model was able to classify 1.32% better than the first model. The confusion matrix and table below will show more evidence that our model heavily focused on classifying the top three breeds in terms of frequency.



	precision	recall	f1-score	support
Terrier	0.38	0.21	0.27	615
Retriever	0.12	0.07	0.09	206
Hound	0.35	0.60	0.44	846
Poodle	0.00	0.00	0.00	57
Bull	0.13	0.04	0.06	159
Spaniel	0.32	0.21	0.25	272
Schnauzer	0.09	0.03	0.05	94
Husky	0.00	0.00	0.00	61
Shepherd	0.24	0.28	0.26	483
Toy	0.23	0.23	0.23	274
avg / total	0.28	0.30	0.27	3067

Again, the decision matrix shows that most of our correctly classified images were from the top three breeds. This model also replicated the result of the top 3 breeds in terms of frequency fall within the top 4 in terms of precision and recall. However, despite having a higher classification rate, this model had a lower total f1 score.

Recommendations and Future Direction:

Although both models came up short of our goals, we were not discouraged. We realized that our models far outperform the result from random chance. We also recognize that there may have been other techniques for CNNs that we did not attempt but would have produced a more accurate model. One way we could implement these changes is by utilizing TensorFlow instead of Keras packages. This would allow for more customization with our models, and hopefully better results. Another option we considered is to make a grey scale model. Although it adds another pre-processing step, we believe that our model might benefit from having less variation to predict and by speeding up processing times. Resizing the images to a width and height indistinguishable to the naked eye is not out of the question; computers and deep learning networks are optimized for this sort of microscopic inspection.

CNNs are extremely robust to various parameters features such as hidden layers, convoluted layers, and optimizers. Keras contains a handful of different optimizers that each

specialize in different fields. These optimizers have a learning rate that can be tweaked depending on the power one desires; however, a high learning rate could lead to overfitting, so its's important to note that this is a sensitive parameter. Layers are an important concept in neural networks because it makes up the foundation of the analysis. Each layer provides the option to customize the number of filters, the kernel size, and the activation function. This project could be refined and retested in an infinite number of ways, which is what makes deep learning an interesting field!

Appendix:

Cleaning the data

```
import pandas as pd
dir='C:/Users/adam_/Documents/MSDA/Applications/Project 2/'
df=pd.read_csv(dir+'labels.csv')
#use to recode the 120 breeds in to 10 main breeds
from collections import Counter
Counter(df.breed) #check breeds before recode

#replace old breeds with new, organized breeds
#this was done manually, would be more efficient with more characteristics of the dogs
df['breed'].replace(['yorkshire_terrier', 'wire-haired_fox_terrier', 'west_highland_white_terrier',
'toy_terrier', 'tibetan_terrier', 'soft-coated_wheaten_terrier', 'silky_terrier', 'sealyham_terrier',
'scotch_terrier', 'norwich_terrier', 'norfolk_terrier', 'lakeland_terrier', 'kerry_blue_terrier',
'irish_terrier', 'dingo', 'dhole', 'dandie_dinmont', 'cairn', 'border_terrier', 'bedlington_terrier',
'australian_terrier', 'american_staffordshire_terrier', 'airedale', 'african_hunting_dog'],'Terrier',
inplace=True)
df['breed'].replace(['vizsla', 'labrador_retriever', 'irish_setter', 'gordon_setter', 'golden_retriever',
'flat-coated_retriever', 'english_setter', 'curly-coated_retriever',
'chesapeake_bay_retriever'],'Retriever', inplace=True)
df['breed'].replace(['whippet', 'weimaraner', 'walker_hound', 'tibetan_mastiff',
'scottish_deerhound', 'saluki', 'saint_bernard', 'rhodesian_ridgeback', 'redbone', 'otterhound',
'norwegian_elkhound', 'newfoundland', 'miniature_pinscher', 'leonberg', 'italian_greyhound',
'irish_wolfhound', 'ibizan_hound', 'great_dane', 'german_short-haired_pointer',
'english_foxhound', 'doberman', 'bull_mastiff', 'borzoi', 'bluetick', 'bloodhound', 'black-and-
tan_coonhound', 'beagle', 'basset', 'basenji', 'appenzeller', 'afghan_hound',
'affenpinscher'],'Hound', inplace=True)
df['breed'].replace(['toy_poodle', 'standard_poodle', 'miniature_poodle'],'Poodle', inplace=True)
df['breed'].replace(['staffordshire_bullterrier', 'rottweiler', 'pug', 'french_bulldog', 'boxer',
'boston_bull'],'Bull', inplace=True)
df['breed'].replace(['welsh_springer_spaniel', 'sussex_spaniel', 'papillon', 'japanese_spaniel',
'irish_water_spaniel', 'english_springer', 'cocker_spaniel', 'clumber', 'brittany_spaniel',
'blenheim_spaniel'],'Spaniel', inplace=True)
df['breed'].replace(['standard_schnauzer', 'miniature_schnauzer', 'lhasa',
'giant_schnauzer'],'Schnauzer', inplace=True)
df['breed'].replace(['siberian_husky', 'malamute'],'Husky', inplace=True)
df['breed'].replace(['shetland_sheepdog', 'schipperke', 'samoyed', 'pembroke',
'old_english_sheepdog', 'malinois', 'kuvasz', 'komondor', 'kelpie', 'groenendael',
'greater_swiss_mountain_dog', 'great_pyrenees', 'german_shepherd', 'entlebucher', 'collie',
'briard', 'bouvier_des_flandres', 'border_collie', 'bernese_mountain_dog'],'Shepherd',
inplace=True)
df['breed'].replace(['pomeranian', 'pekinese', 'mexican_hairless', 'maltese_dog', 'shih-tzu',
'keeshond', 'eskimo_dog', 'chow', 'chihuahua', 'cardigan', 'brabancon_griffon'],'Toy',
inplace=True)
```

```
#used to see the 10 new breeds
Counter(df.breed) #check breeds after recode

#output cleaned data to a new csv file
df.to_csv(dir+'labels_new.csv', index=False)

#resize images
import PIL
from PIL import Image
import os
import sys

path = "C:/Users/adam_/Documents/MSDA/Applications/Project 2/train/" #specify folder of
images
dirs = os.listdir( path ) #list all image names in folder

#program that resizes all the images to a specified size (i.e. 100x100)
def resize():
    for item in dirs: #iterate through all images
        if os.path.isfile(path+item):
            im = Image.open(path+item) #opens the image
            f, e = os.path.splitext(path+item) #split path and item (f=path, e=image name)
            imResize = im.resize((100,100), Image.ANTIALIAS) #resize to 100x100 pixels
            imResize.save(f + '.jpg', 'JPEG', quality=90) #preserve image quality and save file as jpeg
#call function
resize()
```

Modeling the data

```
import pandas as pd
import numpy as np
import os
from glob import glob
import random
import matplotlib.pylab as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import zlib

PATH = os.path.abspath(os.path.join('C:/Users/adam_/Documents/MSDA/Applications/Project
2/'))
SOURCE_IMAGES = os.path.join(PATH, "images_resized")
images = glob(os.path.join(SOURCE_IMAGES, "*.jpg"))
```

```
images[0:10] # image paths

labels = pd.read_csv('C:/Users/adam_/Documents/MSDA/Applications/Project
2/labels_new.csv')
labels.head(10) #first 10 rows in csv file

#add .jpg to end of id name in csv file
for i in range(len(labels)):
    name=labels['id'][i]+'.jpg'
    labels['id'][i]=name
labels.head(10)

from PIL import Image
def proc_images():
    """
    Returns two arrays:
        x is an array of images
        y is an array of labels
    """
    Terrier = "Terrier" #0
    Retriever = "Retriever" #1
    Hound = "Hound" #2
    Poodle = "Poodle" #3
    Bull = "Bull" #4
    Spaniel = "Spaniel" #5
    Schnauzer = "Schnauzer" #6
    Husky = "Husky" #7
    Shepherd = "Shepherd" #8
    Toy = "Toy" #9
    x = [] # images as arrays
    y = [] # labels
    for img in images:
        base = os.path.basename(img)
        finding = labels["breed"][labels["id"] == base].values[0]
        full_image=plt.imread(img)
        x.append(full_image)
        # Labels
        if Terrier in finding:
            finding = 0
            y.append(finding)
        elif Retriever in finding:
            finding = 1
            y.append(finding)
        elif Hound in finding:
            finding = 2
            y.append(finding)
```

```
        elif Poodle in finding:
            finding = 3
            y.append(finding)
        elif Bull in finding:
            finding = 4
            y.append(finding)
        elif Spaniel in finding:
            finding = 5
            y.append(finding)
        elif Schnauzer in finding:
            finding = 6
            y.append(finding)
        elif Husky in finding:
            finding = 7
            y.append(finding)
        elif Shepherd in finding:
            finding = 8
            y.append(finding)
        elif Toy in finding:
            finding = 9
            y.append(finding)
    return x,y

X,y = proc_images()
df = pd.DataFrame()
df["images"]=X
df["labels"]=y

# Describe the distribution of pixel intensities within a representative image
def plotHistogram(a):
    """
    Plot histogram of RGB Pixel Intensities
    """
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.title('Representative Image')
    plt.imshow(a)
    plt.axis('off')
    histo = plt.subplot(1,2,2)
    histo.set_ylabel('Count')
    histo.set_xlabel('Pixel Intensity')
    n_bins = 30
    plt.hist(a[:,:,0].flatten(), bins= n_bins, lw = 0, color='r', alpha=0.5);
    plt.hist(a[:,:,1].flatten(), bins= n_bins, lw = 0, color='g', alpha=0.5);
    plt.hist(a[:,:,2].flatten(), bins= n_bins, lw = 0, color='b', alpha=0.5);
plotHistogram(X[20])
```

```
# Normalize the pixel intensities between zero and one.
X=np.array(X)
X=X/255.0
plotHistogram(X[20])

#Describe new numpy arrays
dict_characters = {0: 'Terrier', 1: 'Retriever', 2: 'Hound',
      3: 'Poodle', 4:'Bull', 5: 'Spaniel', 6: 'Schnauzer', 7: 'Husky', 8: 'Shepherd', 9: 'Toy'}

#print the class counts for 10 breeds
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)

from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)

# Reduce Sample Size for DeBugging
#X_train = X_train[0:1000]
#Y_train = Y_train[0:1000]
#X_test = X_test[0:200]
#Y_test = Y_test[0:200]

# Encode labels to hot vectors
Y_trainHot = to_categorical(Y_train, num_classes = 10)
Y_testHot = to_categorical(Y_test, num_classes = 10)

# In order to avoid having a biased model because of skewed class sizes,
# We need to modify the class_weights parameter in order to give more weight to the rare
classes.
# In this case the class_weights parameter will eventually be passed to the model.fit function.

from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced', np.unique(y), y)
plt.plot(class_weight)
plt.xlabel('Dog Breed Class')
plt.ylabel('Weight')
print(class_weight) #print the weights applied to each class


# Helper Functions  Learning Curves and Confusion Matrix
```

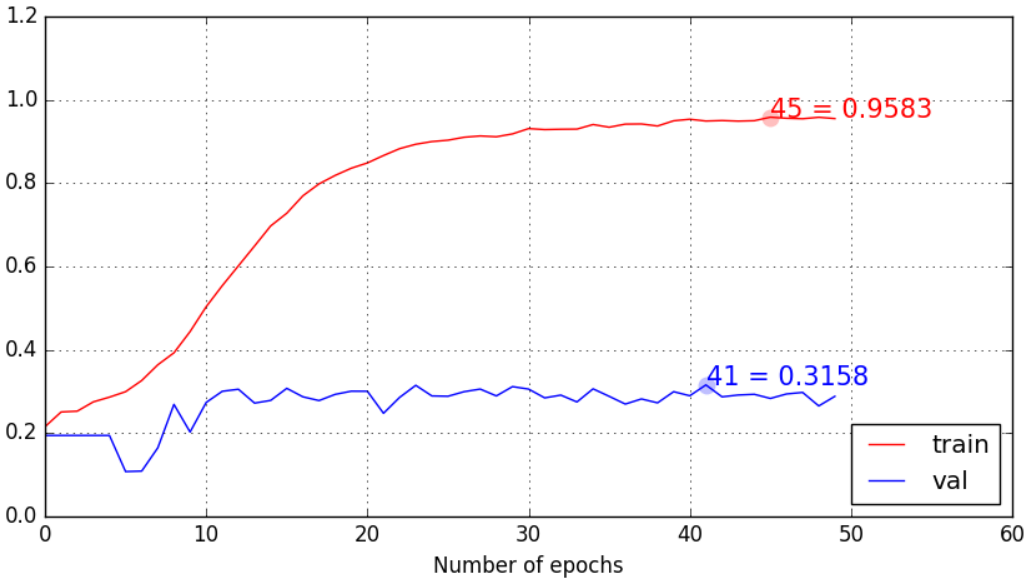
```
from keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)

def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning curve
    for k in filter(lambda x : np.any([kk in x for kk in filt]), metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val' if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color= 'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')

import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
```

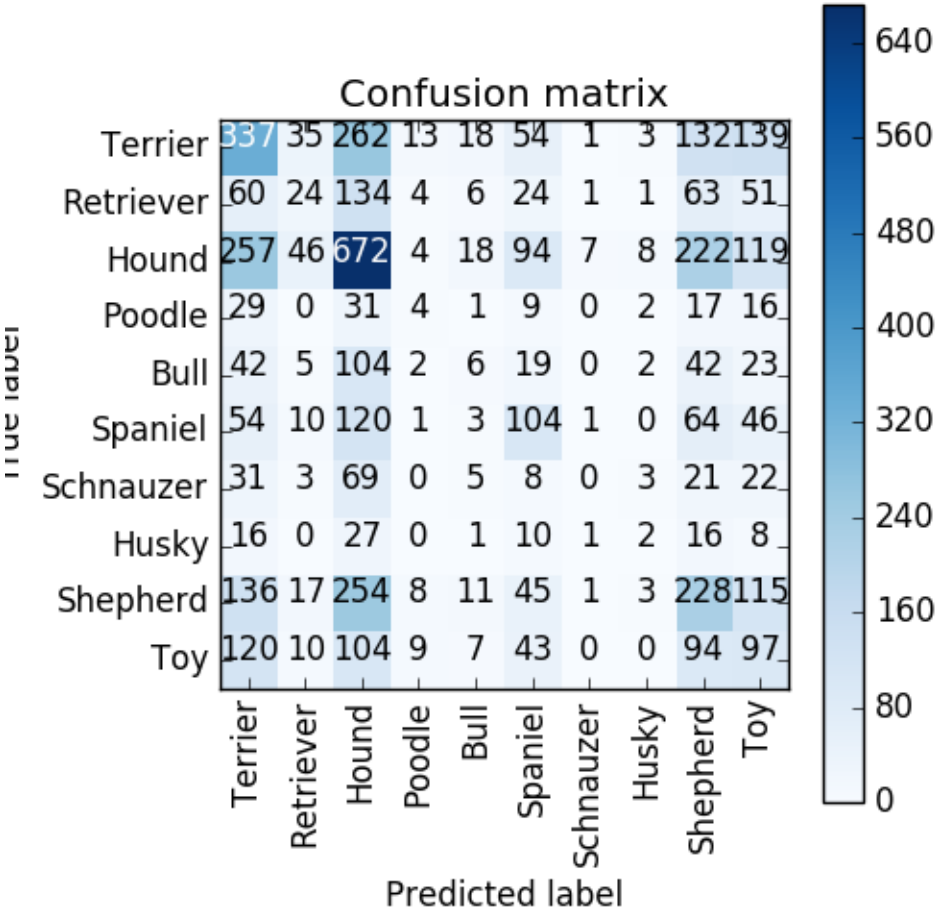
```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

#CNN
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from keras.layers.normalization import BatchNormalization
from keras import optimizers
import keras
from keras.preprocessing.image import ImageDataGenerator
import sklearn
from sklearn.metrics import classification_report, confusion_matrix

def runCNNconfusion(a,b,c,d):
    # In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten -> Dense -> Dropout ->
Out
    batch_size = 50
    num_classes = 10
    epochs = 10
    img_rows, img_cols = X_train.shape[1],X_train.shape[2]
    input_shape = (img_rows, img_cols, 3)
    model = Sequential()
    #Tier 1
    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                     activation ='relu', input_shape = input_shape))
    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    #Tier 2
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
```

```
    model.add(Dropout(0.25))
    #Tier 3
    model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(Conv2D(filters = 86, kernel_size = (3,3),padding = 'Same',
                     activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    #Finalize model
    model.add(Flatten())
    model.add(Dense(512, activation = "relu")) #hidden layers
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = "softmax")) #10 output nodes
    #Define the optimizer
    optimizer=keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-6)
    model.compile(optimizer = optimizer, loss = "categorical_crossentropy",
metrics=["accuracy"])
    model.fit(a, b, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(c, d),
callbacks=[MetricsCheckpoint('logs')])
    score = model.evaluate(c,d, verbose=0)
    print('\nTest loss:', score[0])
    print('\nTest accuracy:', score[1],'\n')
    Y_pred = model.predict(c)
    print('\n', sklearn.metrics.classification_report(np.where(d > 0)[1], np.argmax(Y_pred,
axis=1), target_names=list(dict_characters.values())), sep='')
    Y_pred_classes = np.argmax(Y_pred,axis = 1)
    Y_true = np.argmax(d,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plot_confusion_matrix(confusion_mtx, classes = list(dict_characters.values()))

#RUN THE MODEL!
runCNNconfusion(X_train, Y_trainHot, X_test, Y_testHot)
plotKerasLearningCurve()
```

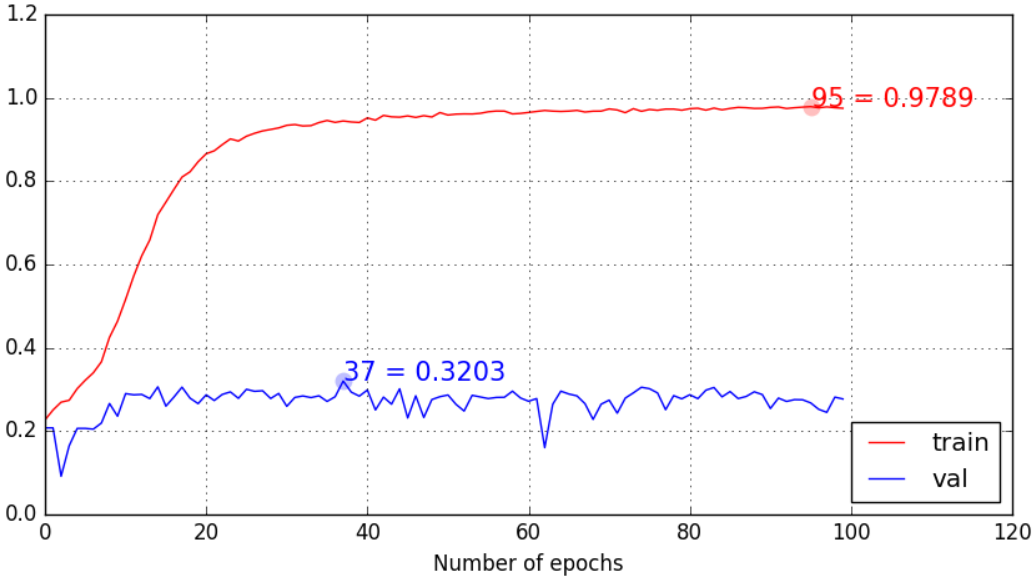
Other Models

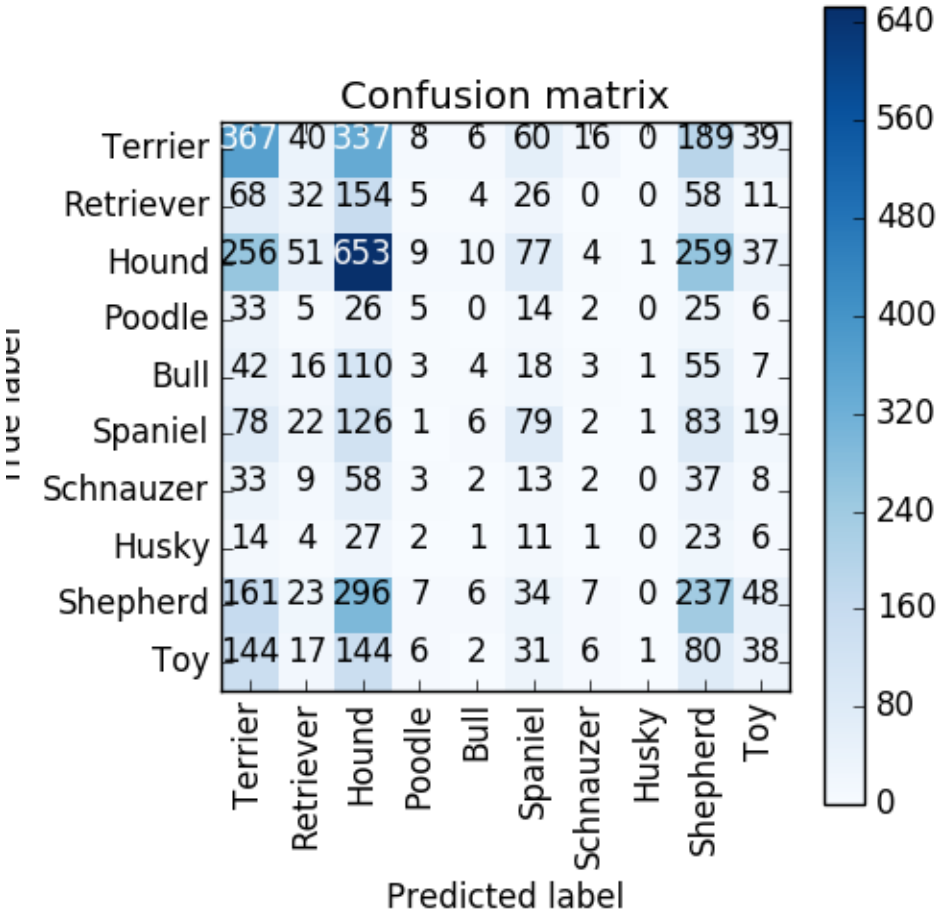
50:50 split, 50 epochs



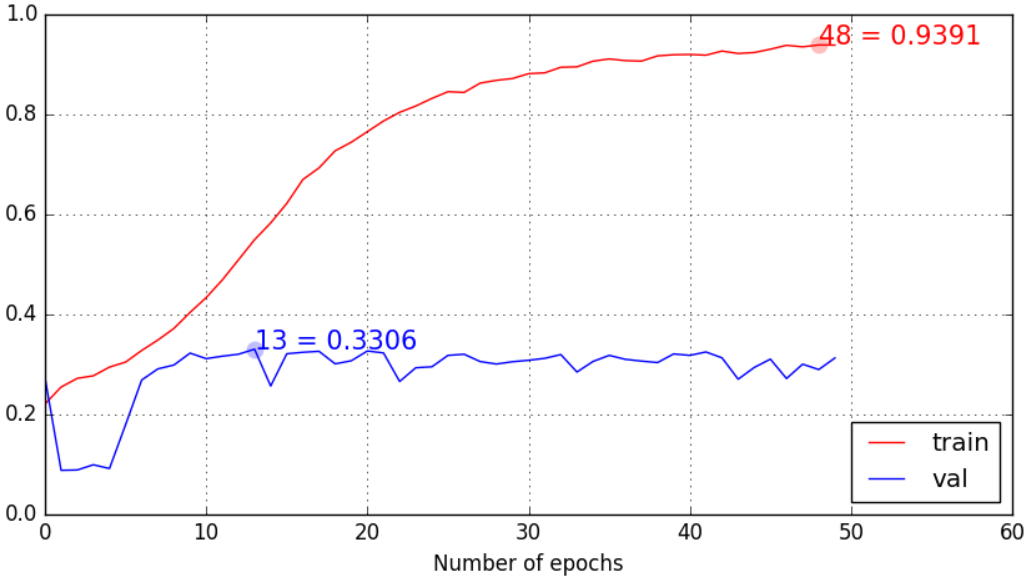


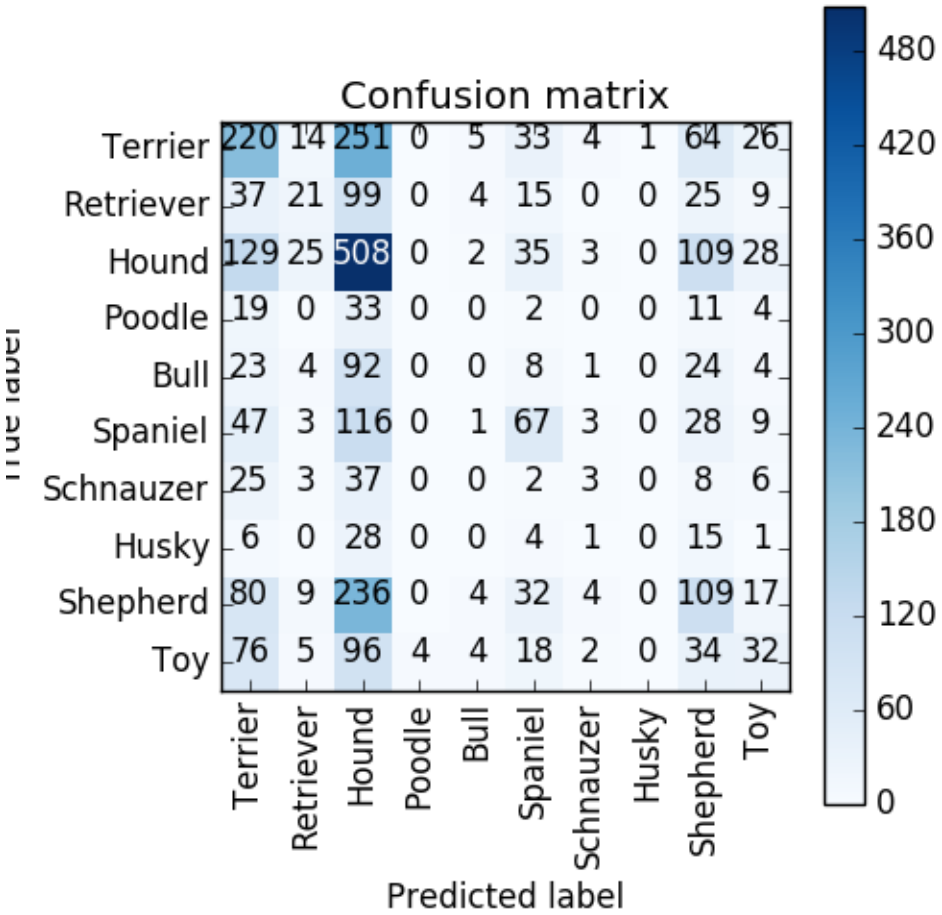
50:50 split, 100 epochs





70:30 split, 50 epochs





80:20 split, 100 epochs