Detection of Dog Anxiety
Computer Vision Final Project Report

Justin Chiang

Jan. 20th 2022

1 Abstract

For years, scientists have been trying to communicate with animals directly. Being able to do so will allow pet owners to know the needs of their dogs but more importantly it will allow scientists and veterinarians to effectively study and treat animals.

This idea inspired me and my partner to use image processing and model training to create a program that can detect dog anxiety. I plan to use ideas such as deep learning and convolution to train my program to classify dogs within images and more accurately detect emotions expressed by them.

In this report, I first demonstrate how machinelearningmastery.com [3] develops a baseline model for a different data set called "dogs_vs_cats" (the tutorial is based on creating training a model to identify dogs from cats in images), then in section 4.3 I recreate the model for our group's data set to separate happy dogs from those feeling anxious.

2 Introduction

At first, I wanted the starting point for my model training to be identifying dogs and classifying them into different breeds. I thought this was important because different breeds of dogs have different personalities which may factor into the accuracy of anxiety detection. However, after some research on emotion detection in humans, I realized that the better way to approach this project was to try implementing existing Python libraries that can analyze images using a convolutional neural network (deep learning). [1]

This leads into the tricky part, which is to detect whether or not the dog is feeling anxious (or any emotion for that matter). This would require a baseline convolutional neural network model for our collected data set, where I follow these steps to eventually achieve my goal:

(i) Plot out the first few files/images in our data set to see that each photos have different colors, shapes, and sizes.

(ii) Pre-process all of the images in our data set by loading the images, reshaping them into the same size, and store them in a NumPy array for modeling.

(iii) Pre-process the images into standard directories.

(iv) Develop our Baseline CNN Model (convolutional neural network model)

(v) Get the accuracy of our model and data summary.

(vi) Make our model make a prediction on a sample image.

Under the assumptions that everything goes smoothly, the program should be able to detect dog anxiety with an acceptable accuracy.

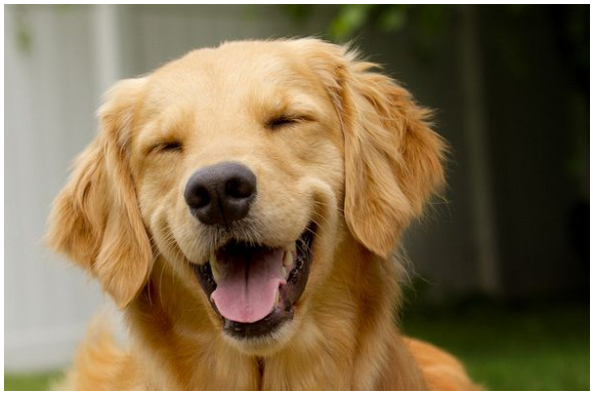


Figure 1: Dog Without Anxiety

Figure 2: Dog With Anxiety

3 Methods

In this section, I demonstrate the steps presented on machinelearningmastery.com [3] by following them and recreating the model on my local device.

3.1 Plotting First Few Images

Similar to the data set we used in problem set 5, I first extracted the "dogs vs cats" data set (mentioned above) containing around 3 million manually annotated photos (the data set was developed as a partnership between Petfinder.com and Microsoft) [3].

From the data set, I extracted and plotted the first 9 photos of dogs and cats with the code below:

```
1
2 # plot dog photos from the dogs vs cats data set
3 from matplotlib import pyplot
4 from matplotlib.image import imread
```

```
# define location of data set
folder = 'train/'
# plot first few images
for i in range(9):
  # define subplot
  pyplot.subplot(330 + 1 + i)
  # define filename
  filename = folder + 'dog.' + str(i) + '.jpg'
  # load image pixels
  image = imread(filename)
  # plot raw pixel data
  pyplot.imshow(image)
# show the figure
pyplot.show()
```

As expected, the resulting figure showed me that the photos were in different shapes and sizes, where some were square and some were landscape, etc.

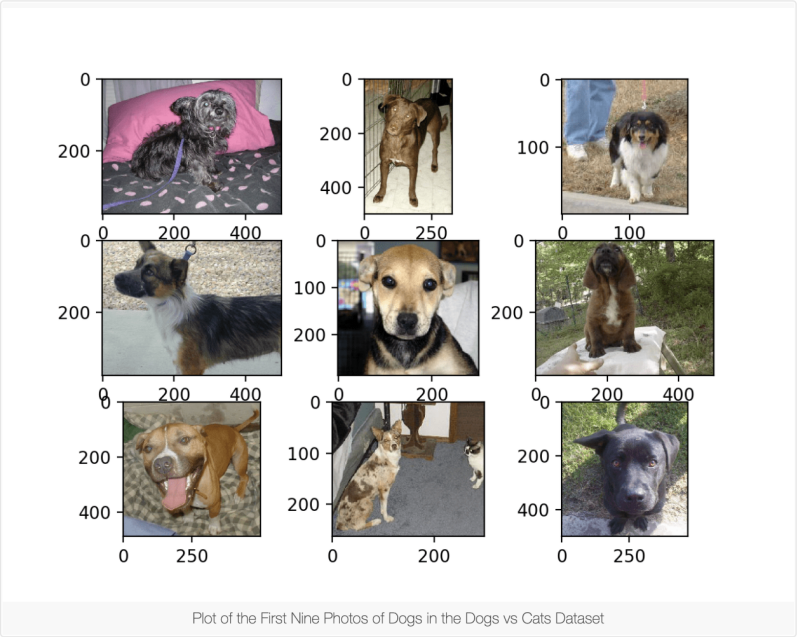


Figure 3: Figure displaying first 9 images of dogs

3.2 Pre-Process Images using Keras

To counter this problem, I used machinelearningmastery.com's suggestion [3] to implement the Keras image processing API to load all the photos within the data and reshape all of them to 200×200 square photos.

The code found online looks like this, where we go in to the file "train", resize each photo

as seen on line 17, and finally save the images into the files named 'dogs_vs_cats_photos.npy' and 'dogs_vs_cats_labels.npy':

```
# load dogs vs cats data, reshape and save to a new file
from os import listdir
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
# define location of data
folder = 'train/'
photos, labels = list(), list()
# enumerate files in the directory
for file in listdir(folder):
  # determine class
  output = 0.0
  if file.startswith('dog'):
    output = 1.0
  # load image
  photo = load_img(folder + file, target_size=(200, 200))
  # convert to numpy array
  photo = img_to_array(photo)
  # store
  photos.append(photo)
  labels.append(output)
```

3.3 Pre-Process Images into Standard Directories

Now that the images in our data set are ready for the final model, we can now prepare the training data set so that it can be loaded by the ImageDataGenerator. We can do this by creating a new directory with all training images organized into dogs/ and cats/ subdirectories (when i recreate this step it will be happydogs/ and anxiety/).

After we do this, the structure will look as follows [3]:

```
finalize_dogs_vs_cats
|--- cats
|___ dogs
```

The complete code:

```
# organize data set into a useful structure
from os import makedirs
from os import listdir
from shutil import copyfile
# create directories
dataset_home = 'finalize_dogs_vs_cats/'
# create label subdirectories
labeldirs = ['dogs/', 'cats/']
for labldir in labeldirs:
  newdir = dataset_home + labldir
  makedirs(newdir, exist_ok=True)
# copy training data set images into subdirectories
```

```
src_directory = 'dogs-vs-cats/train/'
for file in listdir(src_directory):
  src = src_directory + '/' + file
  if file.startswith('cat'):
    dst = dataset_home + 'cats/'  + file
    copyfile(src, dst)
  elif file.startswith('dog'):
    dst = dataset_home + 'dogs/'  + file
    copyfile(src, dst)
```

3.4 Saving the Final Model

We can now fit a final model on the training data set:

```
# prepare iterator
train_it = datagen.flow_from_directory('finalize_dogs_vs_cats/',
  class_mode='binary', batch_size=64, target_size=(224, 224))
```

Once fit, we can save the final model to an H5 file:

```
# save model
model.save('final_model.h5')
```

3.5 Develop a Baseline CNN Model

Proposition 1 (Develop a Baseline CNN Model)**.** *Develop a baseline convolutional neural network model for new data set.*

Although this method is not the most ideal approach for model training because it only establishes a minimal model performance, I still decided to use it because it was the most straightforward.

According to [3], this approach involves the stacking of convolutional layers that consist of 3×3 filters followed by a max pooling layer. These layers can combine to form blocks that can be repeated where the number of filters in each block is increased with the depth of the network such as 32, 64, 128, 256 for the first four blocks of the model. Additionally, we use padding on the convolutional layers to ensure the dimensions of the output feature maps match the inputs.

For example, below is a code representation of Keras definition of a 3-block VGG-style architecture where each block has a single convolutional and pooling layer [2]:

```
# block 1
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same', input_shape=(200, 200, 3)))
model.add(MaxPooling2D((2, 2)))
# block 2
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
# block 3
```

```
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same'))
model.add(MaxPooling2D((2, 2)))
```

Moving forward, [2] recommends creating a define_model() function that can define a model and return it in a form that is ready to be fit on the dataset.

We fit the model with a stochastic gradient descent and start with a learning rate of 0.001 and a momentum of 0.9, which is considered to be pretty safe.

Because our problem is a binary classification task where we want to see an output of either 0 or 1 (0 being a cat and 1 being a dog), an output layer with 1 node and a sigmoid activation will be used and the model will be optimized using the binary cross-entropy loss function.

Below is machinelearningmastery's example of the define_model() function for a convolutional neural network model with one vgg-style block:

```
# define cnn model
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='
    he_uniform', padding='same', input_shape=(200, 200, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform')
    )
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['
    accuracy'])
  return model
```

The function can be called as such:

```
# define model
model = define_model()
```

Next, we need to prepare the data. To do this, we need to first define an instance of the ImageDataGenerator that will scale the pixel values to the range of 0-1.

```
# create data generator
datagen = ImageDataGenerator(rescale=1.0/255.0)
```

After we do this, we need to prepare iterators for both the train and test data.

Machinelearningmastery.com [3] recommends that we use the flow_from_directory() function on the data generator to create one iterator for each of the train/ and test/ directories.

Here, we specify that the problem is a binary classification problem by using the "class_mode" argument, and load 200x200 images with the with "target_size. Additionally, following the recommendations of [3], we will fix the batch size at 64.

```
# prepare iterators
train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
  class_mode='binary', batch_size=64, target_size=(200, 200))
test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
  class_mode='binary', batch_size=64, target_size=(200, 200))
```

We can then finally fit the model under the train iterator and use the test iterator to check our model's accuracy.

The model will be fit for 20 epochs:

```
# fit model
history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
  validation_data=test_it, validation_steps=len(test_it), epochs=20,
   verbose=0)
```

Now the final model can be evaluated on the test dataset directly and we can check our results with the resulted classification accuracy.

```
# evaluate model
_, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
print('> %.3f' % (acc * 100.0))
```

Finally, we can create a summarize_diagnostics() function that takes the history directory and creates a figure with a line plot of the loss a figure for accuracy.

```
# plot diagnostic learning curves
def summarize_diagnostics(history):
  # plot loss
  pyplot.subplot(211)
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='blue', label='train')
  pyplot.plot(history.history['val_loss'], color='orange', label='test')
  # plot accuracy
  pyplot.subplot(212)
  pyplot.title('Classification Accuracy')
  pyplot.plot(history.history['accuracy'], color='blue', label='train')
  pyplot.plot(history.history['val_accuracy'], color='orange', label='test
   ')
  # save plot to file
  filename = sys.argv[0].split('/')[-1]
  pyplot.savefig(filename + '_plot.png')
  pyplot.close()
```

We can apply all the above to see how our model performs under different VGG models. For simplicity, we will only see the results for the One Block and Two Block VGG Model in this report.

The complete example of evaluating a one-block baseline model on the dogs and cats dataset is listed below.

```
# baseline model for the dogs vs cats data set
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
```

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

# define cnn model
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='
   he_uniform', padding='same', input_shape=(200, 200, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform')
   )
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['
   accuracy'])
  return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
  # plot loss
  pyplot.subplot(211)
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='blue', label='train')
  pyplot.plot(history.history['val_loss'], color='orange', label='test')
  # plot accuracy
  pyplot.subplot(212)
  pyplot.title('Classification Accuracy')
  pyplot.plot(history.history['accuracy'], color='blue', label='train')
  pyplot.plot(history.history['val_accuracy'], color='orange', label='test
   ')
  # save plot to file
  filename = sys.argv[0].split('/')[-1]
  pyplot.savefig(filename + '_plot.png')
  pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
  # define model
  model = define_model()
  # create data generator
  datagen = ImageDataGenerator(rescale=1.0/255.0)
  # prepare iterators
  train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
    class_mode='binary', batch_size=64, target_size=(200, 200))
  test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
    class_mode='binary', batch_size=64, target_size=(200, 200))
  # fit model
```

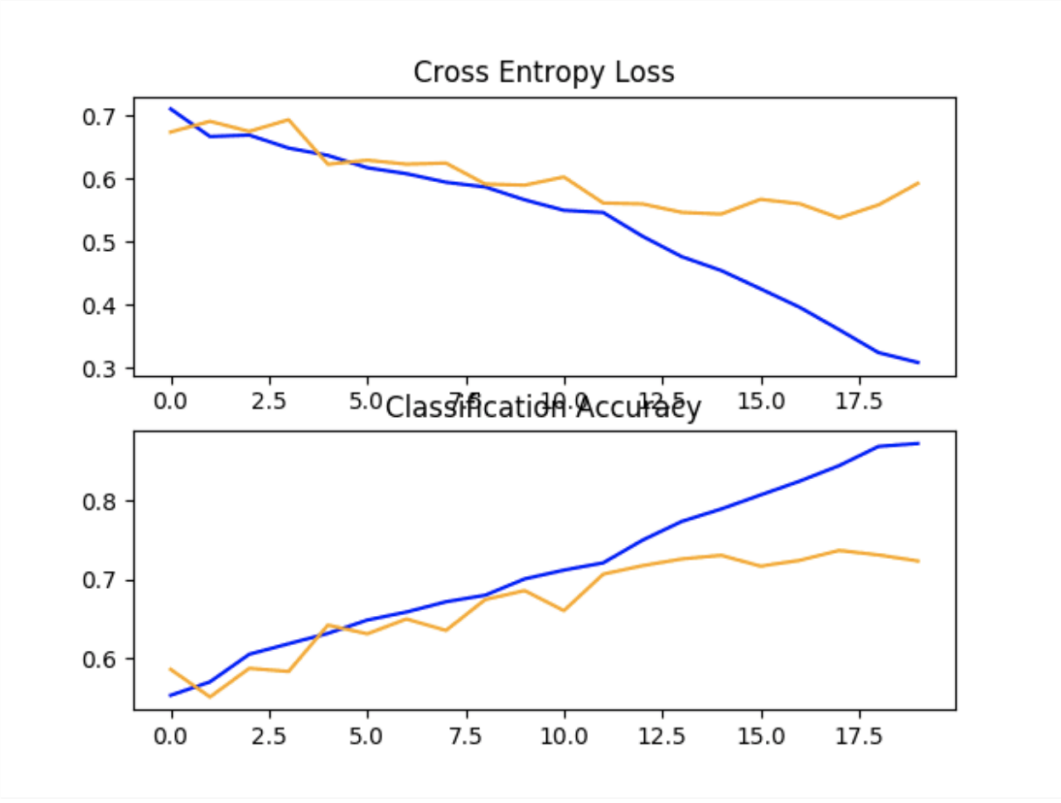
```
  history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
    validation_data=test_it, validation_steps=len(test_it), epochs=20,
    verbose=0)
  # evaluate model
  _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose
    =0)
  print('> %.3f' % (acc * 100.0))
  # learning curves
  summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()
```

Which results in a model with an accuracy of around 72 percent:

```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
> 72.331
```

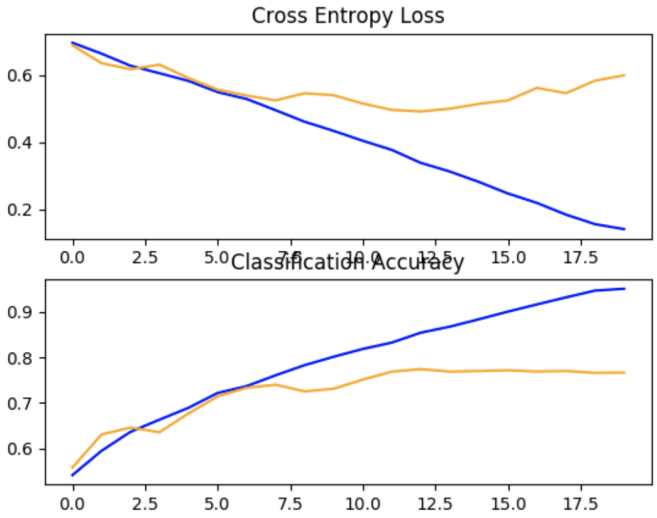
And we also get a line plot that tells us the Cross Entropy Loss and Accuracy of the model.

From this plot we can see that the model has "overfit" the training data set at about 12 epochs [3].



Now, with a Two Block VGG model, we get a 76 percent accuracy and an overfit of the data set at around 8 epochs:

```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
> 76.646
```



4 Results and Discussion

4.1 Before We Make A Prediction

Now, with our model we can finally get ready to make a prediction.

First, we need to prepare the final dataset, which is quite similar to the new directory step above:

```
# organize dataset into a useful structure
from os import makedirs
from os import listdir
from shutil import copyfile
# create directories
dataset_home = 'happy_vs_anxiety/'
# create label subdirectories
labeldirs = ['happydog/', 'anxiety/']
for labldir in labeldirs:
  newdir = dataset_home + labldir
  makedirs(newdir, exist_ok=True)
# copy training dataset images into subdirectories
src_directory = 'train/'
for file in listdir(src_directory):
  src = src_directory + '/' + file
  if file.startswith('anxiety'):
    dst = dataset_home + 'anxiety/'  + file
    copyfile(src, dst)
  elif file.startswith('happydog'):
    dst = dataset_home + 'happydog/'  + file
```

```
    copyfile(src, dst)
```

Then, we save the model:

```

# save the final model to file
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

# define cnn model
def define_model():
  # load model
  model = VGG16(include_top=False, input_shape=(224, 224, 3))
  # mark loaded layers as not trainable
  for layer in model.layers:
    layer.trainable = False
  # add new classifier layers
  flat1 = Flatten()(model.layers[-1].output)
  class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(
   flat1)
  output = Dense(1, activation='sigmoid')(class1)
  # define new model
  model = Model(inputs=model.inputs, outputs=output)
  # compile model
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['
   accuracy'])
  return model

# run the test harness for evaluating a model
def run_test_harness():
# define model
  model = define_model()
  # create data generator
  datagen = ImageDataGenerator(featurewise_center=True)
  # specify imagenet mean values for centering
  datagen.mean = [123.68, 116.779, 103.939]
  # prepare iterator
  train_it = datagen.flow_from_directory('finalize_dogs_vs_cats/',
    class_mode='binary', batch_size=64, target_size=(224, 224))
  # fit model
  model.fit_generator(train_it, steps_per_epoch=len(train_it), epochs=10,
   verbose=0)
  # save model
  model.save('final_model.h5')

# entry point, run the test harness
run_test_harness()
```

4.2 Make A Prediction

We take a closer look at how our model handles the image below (one of the dog images from the test data set):

Figure 4: Example of dog

To see if our model is well trained enough, we are expecting an output of "1" for "Dog".

We load our image and resize it into 224×224 pixels:

```
# load and prepare the image
def load_image(filename):
  # load the image
  img = load_img(filename, target_size=(224, 224))
  # convert to array
  img = img_to_array(img)
  # reshape into a single sample with 3 channels
  img = img.reshape(1, 224, 224, 3)
  # center pixel data
  img = img.astype('float32')
  img = img - [123.68, 116.779, 103.939]
  return img
```

Next, we load the model and call the predict() function to predict the animal in the image:

```
# predict the class
result = model.predict(img)
```

The complete code:

```

# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

```

```
# load and prepare the image
def load_image(filename):
  # load the image
  img = load_img(filename, target_size=(224, 224))
  # convert to array
  img = img_to_array(img)
  # reshape into a single sample with 3 channels
  img = img.reshape(1, 224, 224, 3)
  # center pixel data
  img = img.astype('float32')
  img = img - [123.68, 116.779, 103.939]
  return img

# load an image and predict the class
def run_example():
  # load the image
  img = load_image('sample_image.jpg')
  # load model
  model = load_model('final_model.h5')
  # predict the class
  result = model.predict(img)
  print(result[0])

# entry point, run the example
run_example()
```

As expected, our model spits out the result:

```
1
```

4.3 Repeating Process on Own Data Set

Now, as a crucial part of testing the method of everything mentioned above, I repeated the process on our own group's data set. Our data set consisted of 719 images of Positive (happy dogs) and 215 images that suggested the dogs in them showed signs of anxiety.

So, as following the steps above, I first graphed the first 9 dog images from our group's own data set:

Figure 5: Display of 9 images of dogs from own data set

After this display, I attempted to use the same code above to resize the images to 200x200 pixels and load the photos in the training data set. This is to prepare for the modelling step, but during the process some of the jpg files from our own data set seemed to be corrupted and caused errors as shown below.

However, after debugging, I realized that it was a file called .DS_Store that was causing the error, and I just had to ignore it. Below is the successful run showing our array sizes and image sizes:

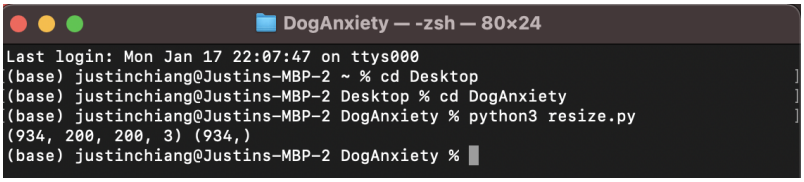


Figure 6: Successful run with images from our own data set

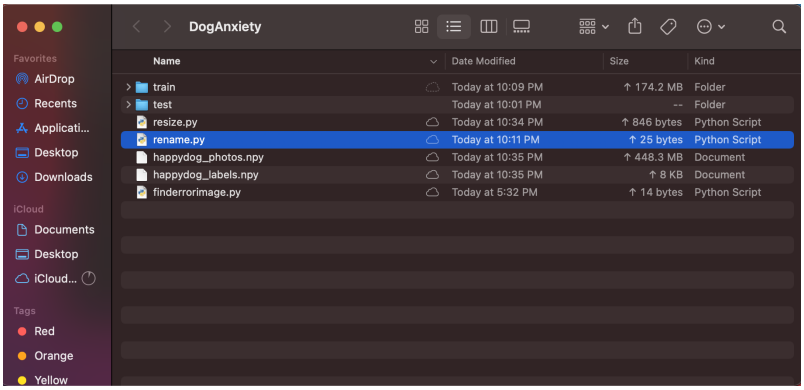


Figure 7: Folder containing new .npy files indicating successful photo pre-processing

Then, using the code from [3] (with a few appropriate changes), we make a directory separating the two categories of images: happydog and anxiety:

```
1
2  # organize data set into a useful structure
3  from os import makedirs
4  from os import listdir
5  from shutil import copyfile
6  from random import seed
7  from random import random
8  # create directories
9  dataset_home = 'dataset_dogs_anxiety/'
10 subdirs = ['train/', 'test/']
11 for subdir in subdirs:
12   # create label subdirectories
13   labeldirs = ['happydog/', 'anxiety/']
```

```
  for labldir in labeldirs:
    newdir = dataset_home + subdir + labldir
    makedirs(newdir, exist_ok=True)
# seed random number generator
seed(1)
# define ratio of pictures to use for validation
val_ratio = 0.25
# copy training data set images into subdirectories
src_directory = 'train/'
for file in listdir(src_directory):
  src = src_directory + '/' + file
  dst_dir = 'train/'
  if random() < val_ratio:
    dst_dir = 'test/'
  if file.startswith('anxiety'):
    dst = dataset_home + dst_dir + 'anxiety/'  + file
    copyfile(src, dst)
  elif file.startswith('happydog'):
    dst = dataset_home + dst_dir + 'happydog/'  + file
    copyfile(src, dst)
```

Figure below shows what the new directory looks like:

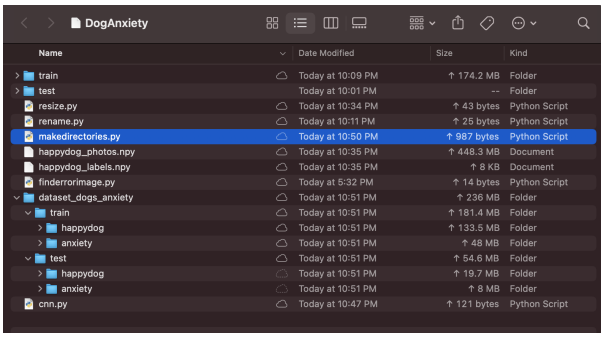


Figure 8: Folder with new directories

Now we run the One-Block VGG code (cnn.py) shown above [3] to get this message in the terminal:

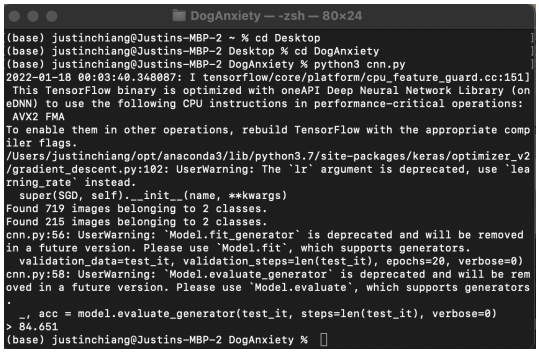


Figure 9: Successful run using One-Block VGG Model, 84.651 accuracy

And after we finish waiting for our model to train, we get a png file of our summary diagnostics indicating our cross entropy loss and classification accuracy:

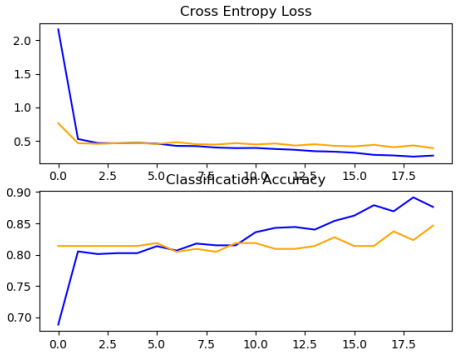


Figure 10: Summary diagnostics from my own model

And finally, we test our model on a Two-Block VGG Model, which surprisingly resulted in a slightly lower accuracy:

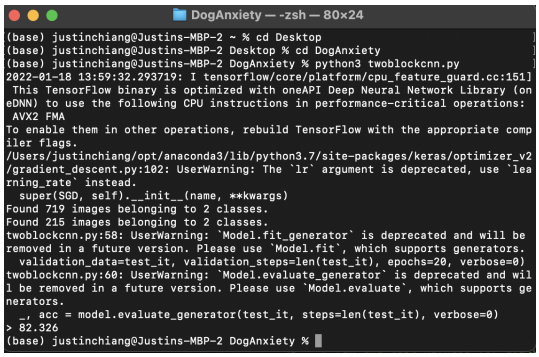


Figure 11: Terminal output from two-block VGG model

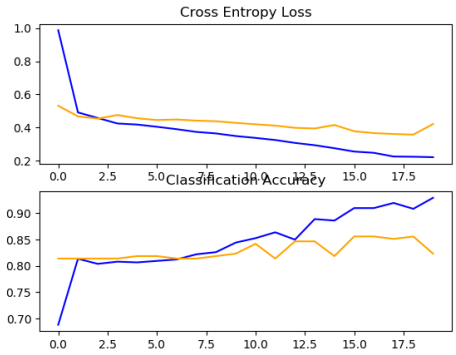


Figure 12: Summary diagnostics from the two-block VGG model

4.4 Make A Prediction on Own Data Set

Last but not least, we can finally make a prediction using our model and data. In our case, an output of 1 indicates a happy dog and 0 indicates a dog feeling anxious.

We repeat the steps above, first using "preparefinaldataset.py" then "savefinalmodel.py" to get this terminal output and out h5 file:

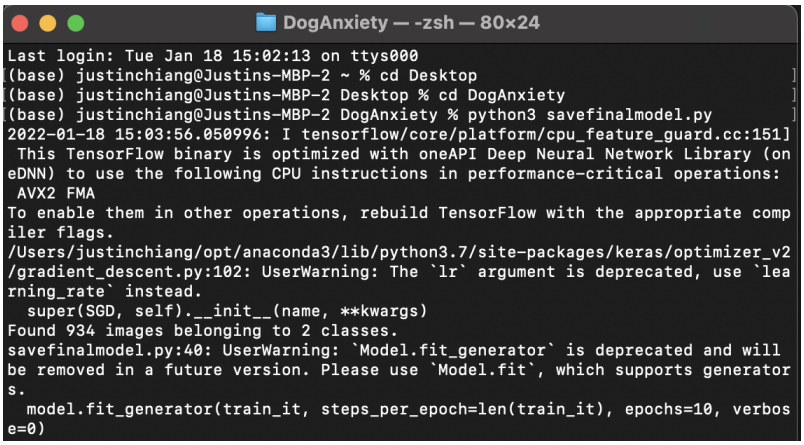


Figure 13: Terminal output from two-block VGG model

And finally "makeprediction.py":

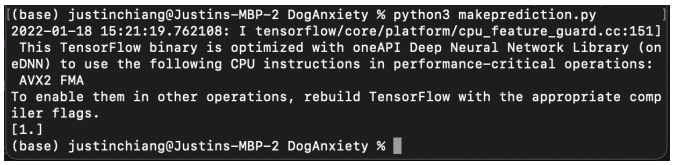


Figure 14: Terminal output for prediction: 1 is happy 0 is anxiety

5 Conclusion

Through learning how to train a model by following the steps on machinelearningmastery.com [3] and repeating them on my own data set (with appropriate code changes and methods), I managed to successfully train my model to identify happy dogs from those with anxiety. Although our group's data set was not big enough to actually become a functioning program for people to use, I could definitely the potential in it as long as I gather more data.

Throughout this project, I encountered errors in the code such as "corrupted images" that turned out to be caused by a file named .DS_Store, and spending too much time debugging.

Looking back at this project, I learned the importance of neural networks, model training, and how to process images in the proper way. In the future, I see a possibility to expand this project into an application that can be actually used.

References

[1] Singh, Rahulraj. (2021, July). *The Ultimate Guide to Emotion Recognition from Facial Expressions Using Python*. Medium, Towards Data Science. Retrieved from https://towardsdatascience.com/the-ultimate-guide-to-emotion-recognition-from-facial-expressions-using-python-64e58d4324ff.

[2] Basavarajaiah, Madhushree. (2019, April). *6 Basic Things to Know about Convolution*. Retrieved from https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411.

[3] Brownlee, Jason. (2021, December). *How to Classify Photos of Dogs and Cats*. Machine Learning Mastery, Retrieved from https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/.