Cat and Dog Image Classification using Neural Networks to Develop a Web App

Susan Nunez, Claire Robinson, Madison Yonash

Introduction

As the project for **CAP 4793 Advanced Data Science** was very open-ended, our team wanted to choose a project that encompasses both something that interests us and something that would challenge us to try something we had not yet done in our academic endeavors. Our ins**purr**ation came from our **fur**iends:

Figure 1: Claire's Dog, Macy, and Madi's Cat, Lazarus

If you're a human looking at those photos, you can probably easily tell that one is a cat and one is a (very stylish) dog. But if you have thousands of photos of cats and dogs, it would take quite a bit of time for a human to **cat**egorize all of them. This is where machine learning comes in. Our team created a convolutional neural network to perform the task of classifying photos as either cats or dogs. A dataset was used that contained 25,000 photos separated into two folders: cats and dogs. Additionally, we wanted to create an interface in which a user could upload their own photo and have the model classify it. This required the use of flask in Python (although the initial approach explored involved using Shiny to develop the web app). We were very **mew** to both tasks, so it provided a great learning opportunity for the team. (I **purr**omise- the pet puns end right **meow**!)

Data Description

This data set includes 25,000 images that are a subset of three million annotated images compiled by Petfinder.com and Microsoft. These 25000 images are split into two classes. These classes are not split into training and testing sets, but separated based on if they are a cat or dog. Many of the images have additional noise/obstructions in the images, duplicated animals, various angles and distances, and different poses. The additional noise, image shapes, and pixel colors in this data set means that the data is not uniform.

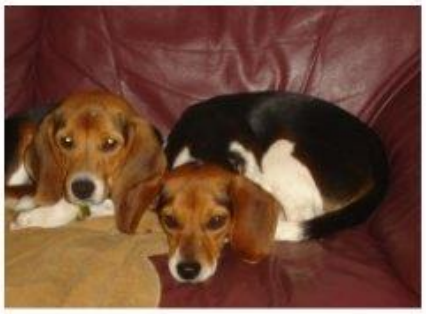
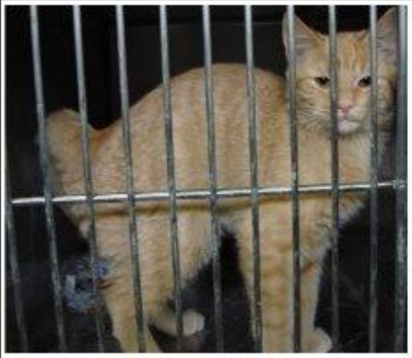
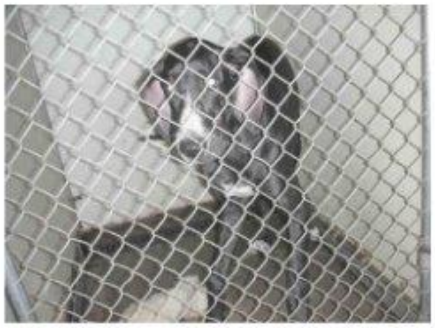
	Cat	Dog
Normal		
Duplicated		
Obstructed		

Figure 2: Some images from our dataset showcasing differences in the images

Exploratory Data Analysis

For the EDA of this project, we randomly sampled nine images from both the cat photos and dog photos so we could get an idea of what we were working with. This allowed us to find some of the issues within the data such as the duplications and obstructions mentioned in the data description.

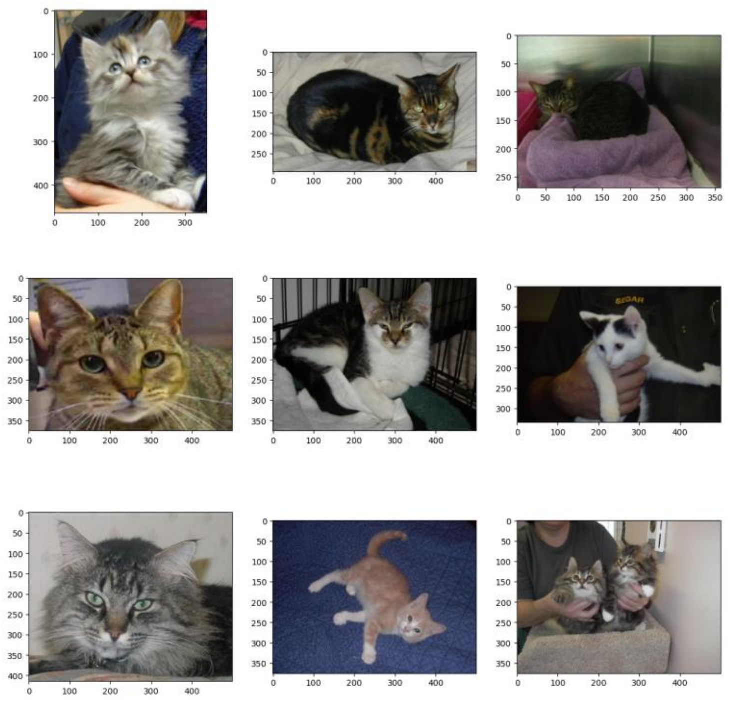


Figure 3: Nine randomly sampled photos from the Cat folder

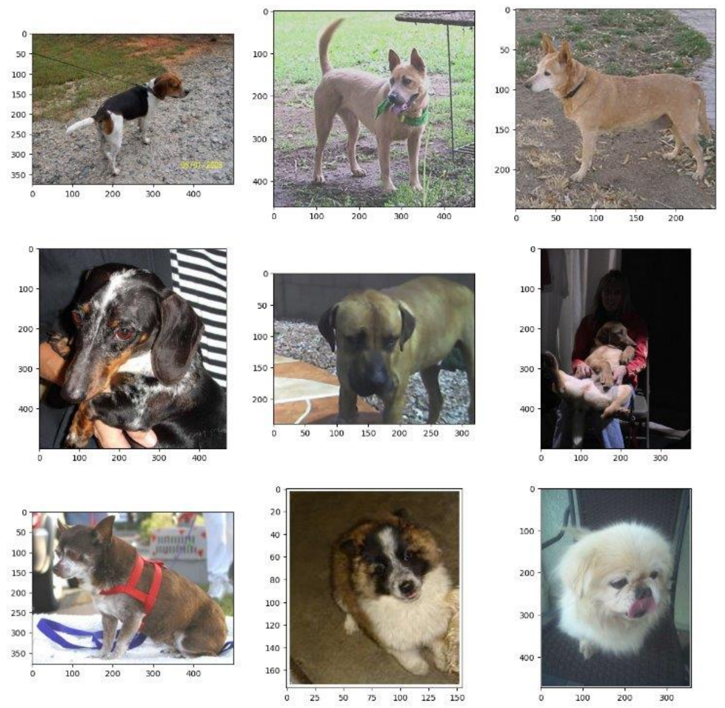


Figure 4: Nine randomly sampled photos from the Dog folder

Data Preprocessing

The raw data comes in all different shapes and colors that would not work very well to run through a model. In order to properly train and test these images we need to make them more uniform. All images were resized to 128x128 pixels, and their color values were normalized by dividing them by 255. Standardizing the colors in this way is a common method specifically for images in the Red, Green, and Blue (RGB) color space. Along with this, we added data labels to the image data with 0 being designated for cats and 1 for dogs.

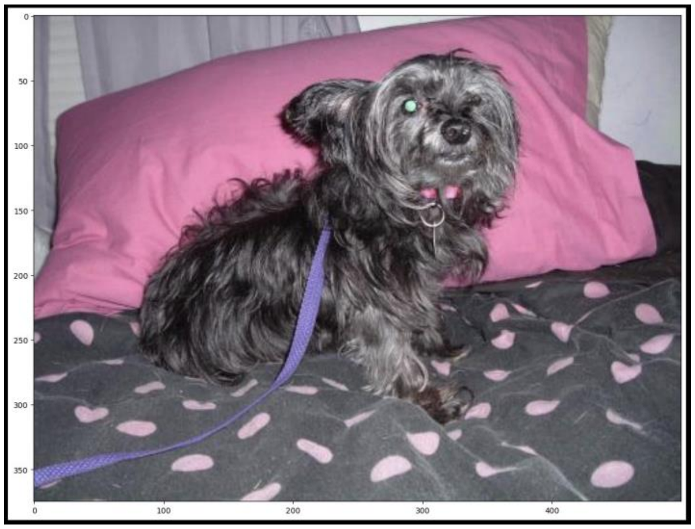
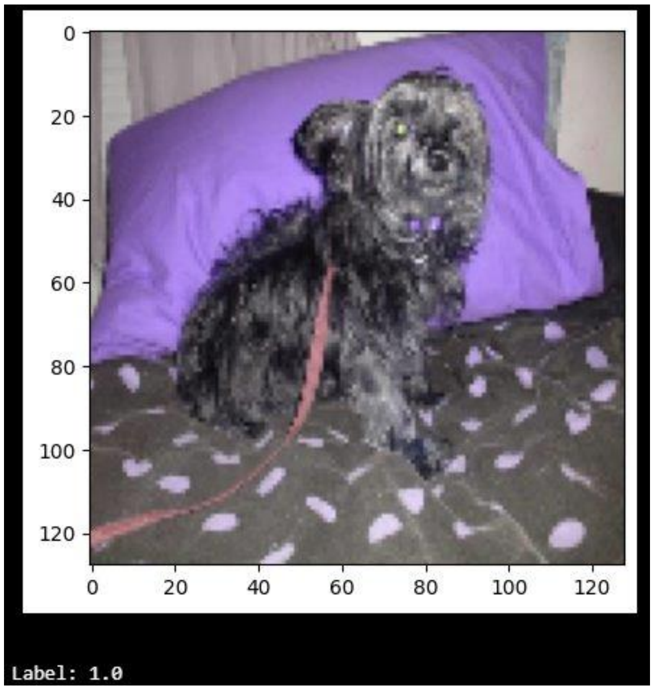


Figure 5: An image before preprocessing



```
Label: 1.0
```

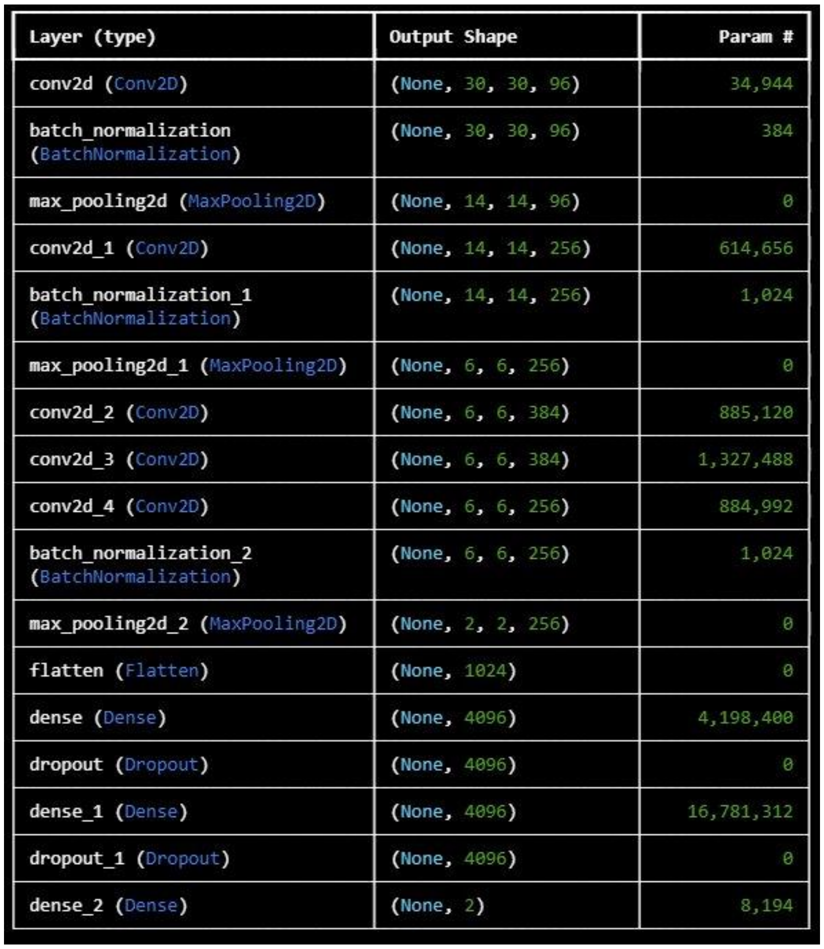
Figure 6: An image after preprocessing.

Lastly, a training and testing set was created using an 80/20 split. During the split, the data was also randomly shuffled as when the two folders were combined into one dataset, it followed an arbitrary order of all the cat images followed by all of the dog images.

Modeling

Our model used a prebuilt architecture made by Abdullah Al Asif on Kaggle (Found here.). This neural network is loosely based on the structure of an AlexNet model. AlexNets are a type of convolutional neural network (CNN) created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton that won the LSVRC (Large Scale Visual Recognition Challenge) in 2012 and gained international attention. The model has a total of 13 layers in the following structure:

Table 1: Model layer structure, output shape, and parameter numbers



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 96)	34,944
batch_normalization (BatchNormalization)	(None, 30, 30, 96)	384
max_pooling2d (MaxPooling2D)	(None, 14, 14, 96)	0
conv2d_1 (Conv2D)	(None, 14, 14, 256)	614,656
batch_normalization_1 (BatchNormalization)	(None, 14, 14, 256)	1,024
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 6, 6, 384)	885,120
conv2d_3 (Conv2D)	(None, 6, 6, 384)	1,327,488
conv2d_4 (Conv2D)	(None, 6, 6, 256)	884,992
batch_normalization_2 (BatchNormalization)	(None, 6, 6, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 4096)	4,198,400
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 2)	8,194

The final dense layer has an output layer of (None, 2) which is the probability of the image being a dog and the probability of the image being a cat. The flatten layer connects the convolutional layers to the dense layers by changing the output shape. The following table goes more indpeth into what each layer does:

Table 2: Layer types and their explanations

Layer	Explanation
Convolutional	Extract features such as edges, texture, and patterns
Max Pooling	Reduce dimensions by selecting max value from feature map regions
Batch Normalization	Normalize input from previous layer through recentering or rescaling (makes NN more stable and faster)
Flatten	Flatten input to 1D vector in order to put through Dense layers (convolutional to dense)
Dense	A layer of neurons that connects to other layers of neurons (Activation Function: ReLU aka Rectified Linear Unit)
Dropout	Randomly sets inputs to zero- aka it randomly drops neurons (helps prevent overfitting)

As images move between convolutional layers, the feature maps it creates to detect features within the images become more and more abstracted. This is due to things such as the max pooling layers that reduced the dimensions of the data.

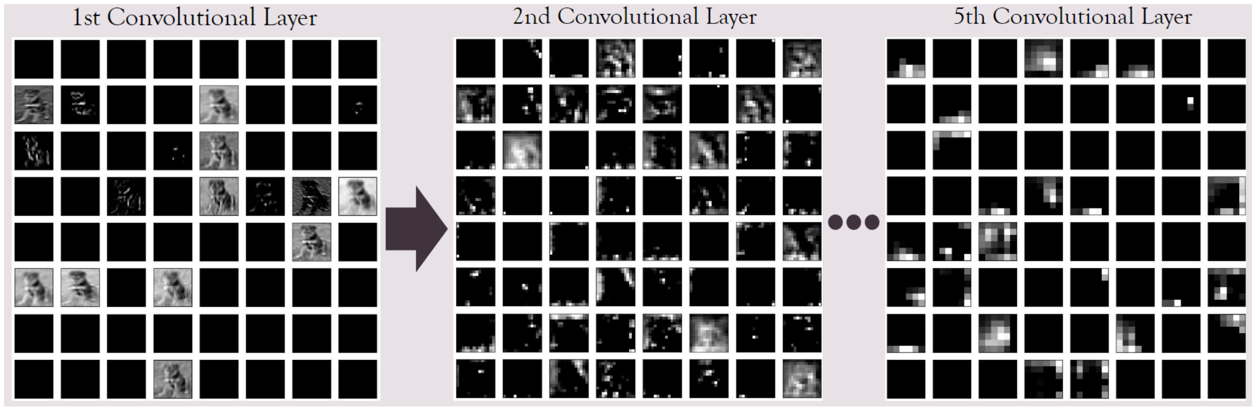


Figure 7: Progression of abstraction in feature maps in the 1st, 2nd, ..., and 5th convolutional layer.

The model was set to 100 epochs– meaning the model with be trained on the entire training dataset 100 times. With this number of epochs, we were able to achieve roughly 84% accuracy and a loss of roughly 1.5.

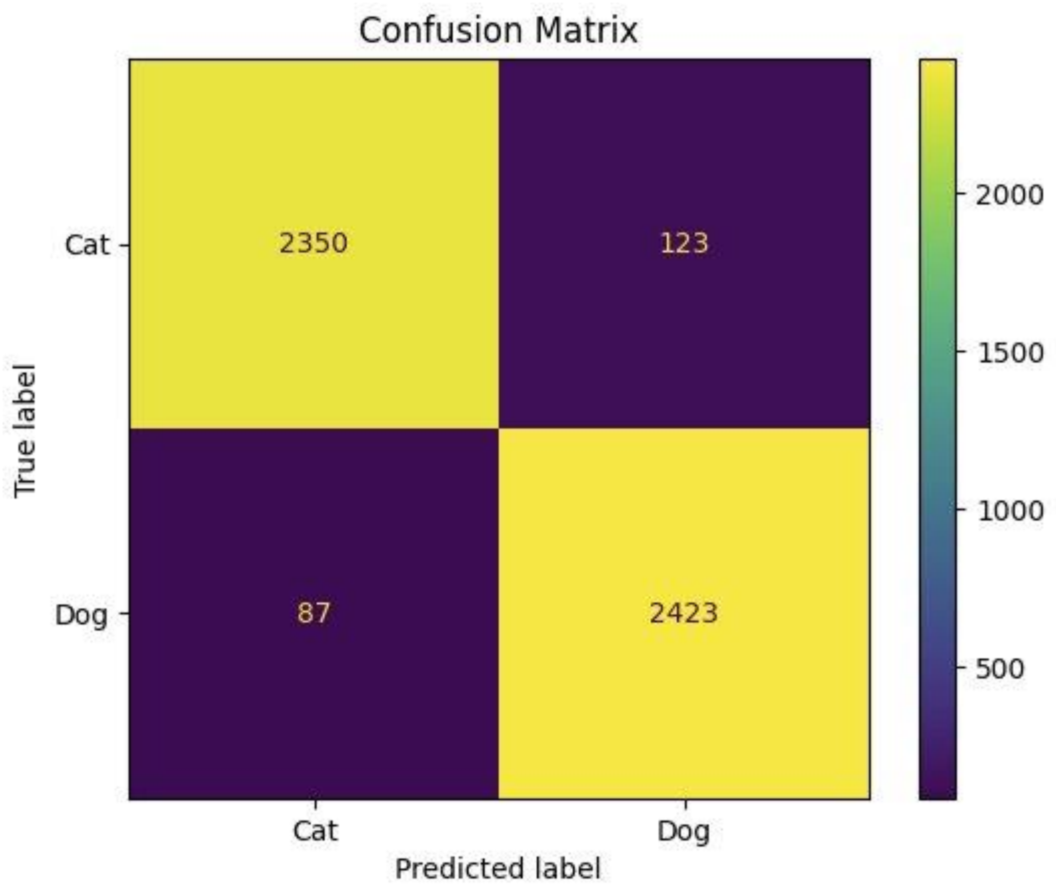


Figure 8: Confusion matrix of results from making prediction on test set with model.

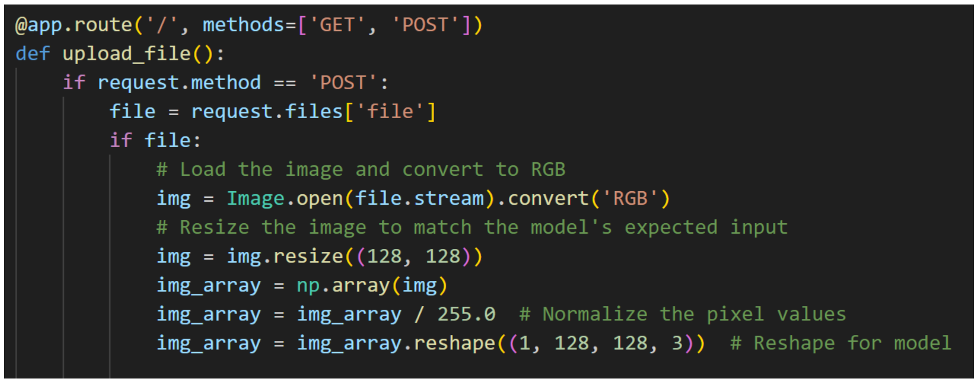
Note that the original model made by Abdullah Al Asif used 1000 epochs and achieved much higher accuracy. Unfortunately, due to both time and computing constraints, it was not possible to train the model with such a high number of epochs.

Web App

The web app was developed using the flask package in Python. Our initial approach involved using shiny, but due to the greater documentation for flask in Python, we switched approaches, and built our web app with that tool. The structure of the app is as follows: the user uploads an image, the image undergoes pre-processing, the image is classified using the model, and the web app outputs the photo, result, and probability to the user.

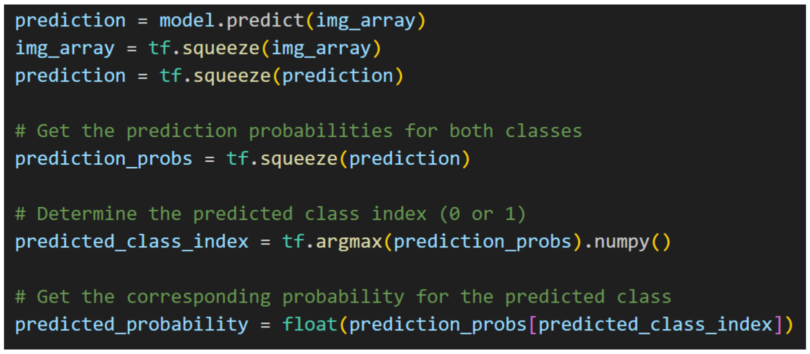
```
app = Flask(__name__)
model = tf.keras.models.load_model("model_alex.h5")  # Update to model's path
```

First, the app is initialized and the model that was created in the previous script is loaded into the script.



```python
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file:
            # Load the image and convert to RGB
            img = Image.open(file.stream).convert('RGB')
            # Resize the image to match the model's expected input
            img = img.resize((128, 128))
            img_array = np.array(img)
            img_array = img_array / 255.0  # Normalize the pixel values
            img_array = img_array.reshape((1, 128, 128, 3))  # Reshape for model
```

Preprocessing is performed to match the input required by the model. This involves reshaping the image, normalizing the pixel values, resizing the image, and converting the image to RGB.



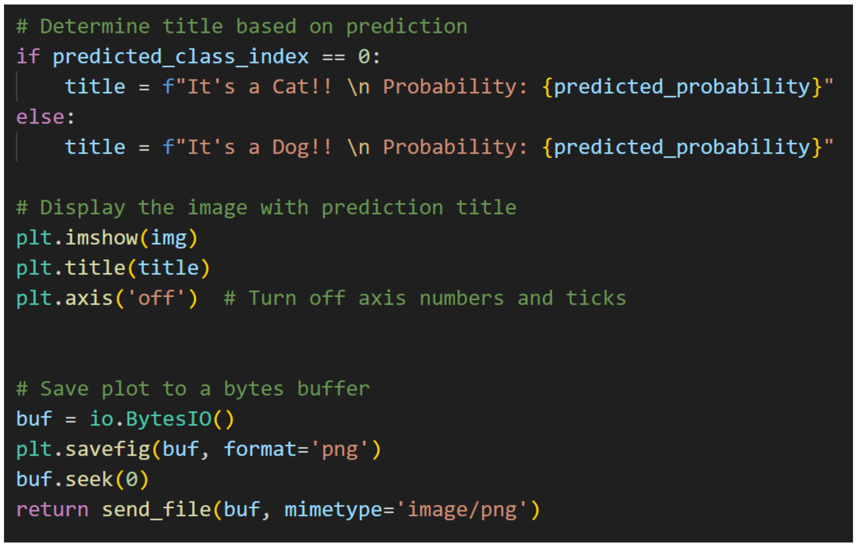
```python
prediction = model.predict(img_array)
img_array = tf.squeeze(img_array)
prediction = tf.squeeze(prediction)

# Get the prediction probabilities for both classes
prediction_probs = tf.squeeze(prediction)

# Determine the predicted class index (0 or 1)
predicted_class_index = tf.argmax(prediction_probs).numpy()

# Get the corresponding probability for the predicted class
predicted_probability = float(prediction_probs[predicted_class_index])
```

The image, having been converted into an array of pixels, is then passed through the model to be classified. From this, we take the predicted class and the probability of the image being of that predicted class.

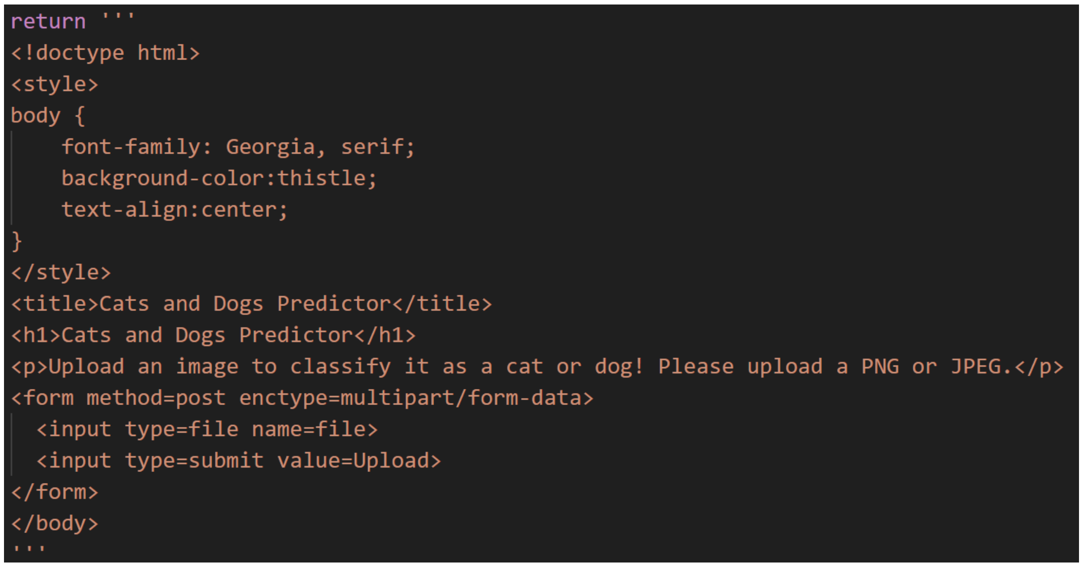


```python
# Determine title based on prediction
if predicted_class_index == 0:
    title = f"It's a Cat!! \n Probability: {predicted_probability}"
else:
    title = f"It's a Dog!! \n Probability: {predicted_probability}"

# Display the image with prediction title
plt.imshow(img)
plt.title(title)
plt.axis('off')  # Turn off axis numbers and ticks


# Save plot to a bytes buffer
buf = io.BytesIO()
plt.savefig(buf, format='png')
buf.seek(0)
return send_file(buf, mimetype='image/png')
```

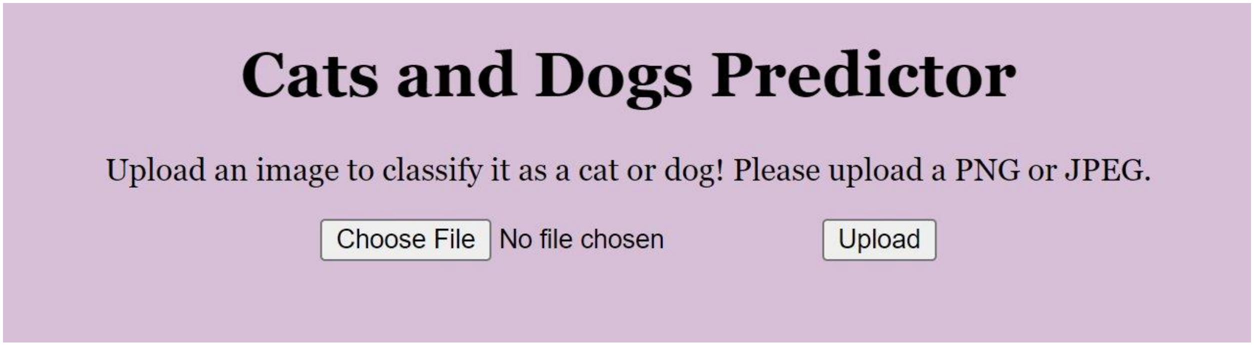
The output of the web app is configured here. The classification of the image is determined by the predicted class index: 0 for cat and 1 for dog. Also, the prediction probability and uploaded image are given.



```
return '''
<!doctype html>
<style>
body {
    font-family: Georgia, serif;
    background-color:thistle;
    text-align:center;
}
</style>
<title>Cats and Dogs Predictor</title>
<h1>Cats and Dogs Predictor</h1>
<p>Upload an image to classify it as a cat or dog! Please upload a PNG or JPEG.</p>
<form method=post enctype=multipart/form-data>
  <input type=file name=file>
  <input type=submit value=Upload>
</form>
</body>
'''
```

HTML is used to create the interface for the web app. Customizations chosen include the font, background color, and verbiage used. This step ended up being surprisingly easy and fun! It was a new experience but provided insight into what the basics of web app design may entail.

When running the app, there are two screens that the user encounters: one where an image is selected and submitted, and a second where the output of the model and photo are displayed:



Cats and Dogs Predictor

Upload an image to classify it as a cat or dog! Please upload a PNG or JPEG.

Figure 9: Main page of web app

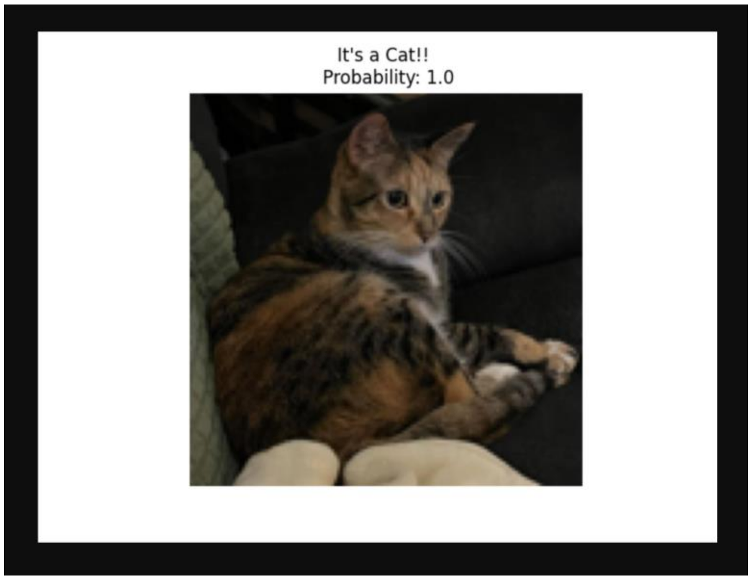


Figure 10: Output of web app after processing photo

The easy-to-use interface of the web app made testing the model on a singular photo very easy and quick. It does not require the user to run any code to receive an output. This made testing new photos relatively easy and an entertaining task at that.

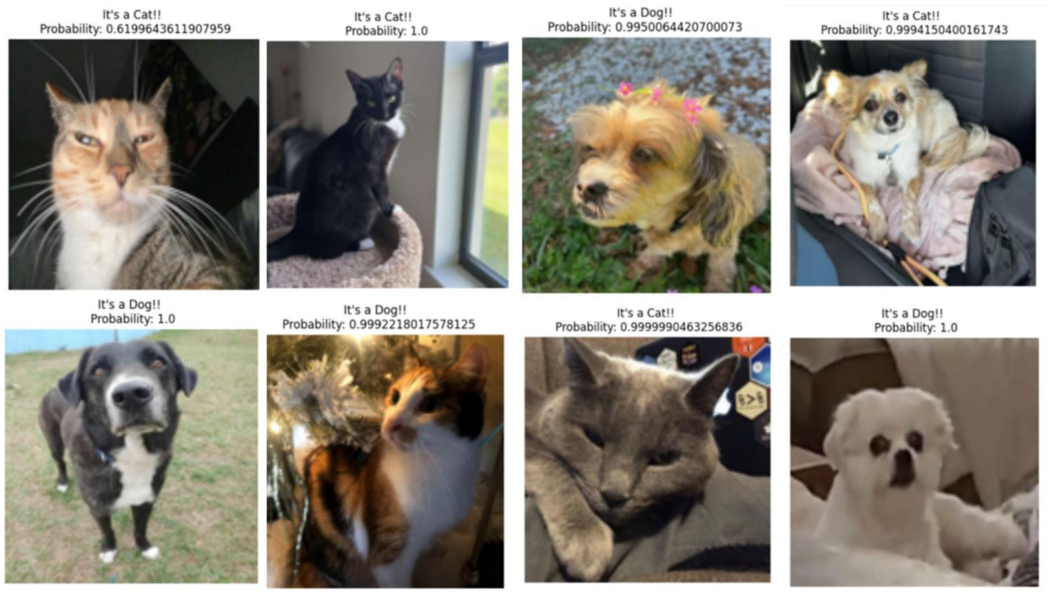

It's a Cat!!
Probability: 0.6199643611907959
It's a Cat!!
Probability: 1.0
It's a Dog!!
Probability: 0.9950064420700073
It's a Cat!!
Probability: 0.9994150400161743
It's a Dog!!
Probability: 1.0
It's a Dog!!
Probability: 0.9992218017578125
It's a Cat!!
Probability: 0.9999990463256836
It's a Dog!!
Probability: 1.0

Figure 11: Results of the model tested on photos from friends

Conclusion

A custom network was created from two Kaggle examples that were found since these examples had previously been ran with a high accuracy probability. The neural network model was able to perform at 84 percent accuracy. The lighting, angle, shape, and additional noise affected the model's view of the animal. For fun and to make our project more interactive a flask app implemented in python was used. There were more resources on working with flask which is why we switched from a Shiny app.

Future Work

In terms of future work, finding a way to host the web app off the local device would be a fun experience to attempt. Currently, the flask app is run locally. The main issue is that the model file that is used for classifying a photo is over 255 MB and finding somewhere to store this proved difficult (as it exceeds the capacity that GitHub can hold). A possible exploration would be to use Shiny or Google Drive and see if a workaround could be found. Additionally, we would like to view different kinds of models and neural networks to compare the results. With this comparison, we could continue to improve upon and adjust the current model. As we improve on the model, we wish to add more data as this is only a very small subset of the full dataset of three million images. We eventually wish to explore more image pre-processing techniques to see if it works better with the model. And finally, we could add in an "other" option if the probability of an image being a cat or a dog is close (for example 55 - 45) as this image could be hard to view even for humans let alone this model.

References

ASIF , A. A. (2023). Exploring CNN models: Cats and dogs classification. https://www.kaggle.com/code/asif00/exploring-cnn-models-cats-and-dogs-classification

Acsany, Philipp. (2023, December 13). Build a Scalable Flask Web Project From Scratch. Real Python. https://realpython.com/flask-project/

Brownlee, J. (2021, December 7). How to classify photos of dogs and cats (with 97% accuracy). MachineLearningMastery.com. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

Lendio, L. (2023, September 6). Cats vs dogs image classification model. Kaggle. https://www.kaggle.com/code/orensa/cats-vs-dogs-image-classification-model

Padhiar, K. (2020, May 1). Cat vs dog dataset. Kaggle. https://www.kaggle.com/datasets/karakaggle/kaggle-cat-vs-dog-dataset