**Project Overview**

Image recognition has become a mainstream technology for many developers in today’s era of computer science. Neural networks are being used to decipher and interpret pixelated images so that the subtle patterns in images can be translated by an algorithm into something meaningful. Naturally occurring problems that human beings ask themselves every day require complex thought and vision analysis.

Take images of animals for example. Neural networks can be used in a variety of ways to interpret these types of images, solving problems such as is this animal a cat or dog, or is there an animal outlined in this frame but is it camouflaged? Are there people’s faces in the frame, and if so, do they look like any particular individuals? Regular people can answer these questions; however the real magic of neural networks is getting a computer to also answer the same questions.

**Problem Statement**

The goal of this project is to determine what kind of dog breed a picture of a dog or human most likely resembles. In order to do this, a type of neural network known as a “convolution neural network” will be used to analyze the data and make reasonable inferences as to what kind of dog breed is in the image. Images stored as portable network graphics (PNG) or JPEG files will be used as input. The application will determine based on the image if the image is a picture of a dog, human, or neither. Depending on the answer, it will change the text of the final output window. The algorithm will then take these image files and transform them into normalized three-dimensional data structures known as “tensors”. These tensors will contain data for each pixel in the image corresponding to numerical values for the red, green, and blue shading that make up the image.

The tensors will then be passed as input into a series of “convolutional” layers of a neural network. A convolutional layer is a way of condensing data structures into smaller structures that summarize a portion of the original data structure. This is done by applying calculations based on each set of values in a tensor. In this case, sections of pixels will be condensed into smaller images and segmented into a large amount of different “channels” or output images. The final output images will then be used as the input into a fully-connected network, where the input will be the flattened nodes that will feed into the remaining neural network. The output will be a “class” that has scored the highest, which will contain the breed that the network thinks the image portrays.

**Metrics**

There are 133 different dog breeds, and some dog breeds have a variety of different looking animals (color can be different, fur can be different). Therefore the training method will loop through this process until the algorithm achieves at least 60% accuracy on the testing data, considering with how complex the dataset is, this seems like a reasonable benchmark (randomly guessing 1 out of 133 breeds would yield less than 1% accuracy). Hyperparameters such as the learning rate, fully-connected architecture, and the loss function can be tuned to help reach this benchmark. A pre-trained VGG16 convolutional network will be used to achieve this benchmark. Pre-trained networks are great to use because these models have been designed for image classification and have been trained on millions of images. They can handle large datasets very well, and in this case will be valuable tools at maintaining higher accuracy when interpreting the images.

**Data Exploration and Visualization**

The datasets will consist of large dog and human datasets. The datasets are provided by Udacity and can be located here:

*Dog dataset:*

[*https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip*](https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip)

*Human dataset:*

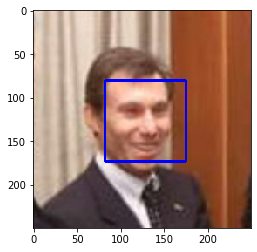
[*https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip*](https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip)

There are 13233 total human images and 8351 total dog images. The human images will be processed using a module known as “OpenCV” which makes use of Haar feature-based cascade classifiers.



Figure 1: Haar Cascade Classifiers, taken from https://docs.opencv.org/trunk/db/d28/tutorial\_cascade\_classifier.html

Like the convolutional networks, the cascade classifiers will filter the incoming images into different types which will place an emphasis on either edges, line features, rectangles, etc… The OpenCV module provides a pre-trained model and framework for detecting human faces.



Visualizing the python module with matplotlib on our dataset yields an example image as pictured above. Notice there is a bounding box. Our algorithm will count the number of human facial bounding boxes to determine if the picture is a picture of a dog or a person.

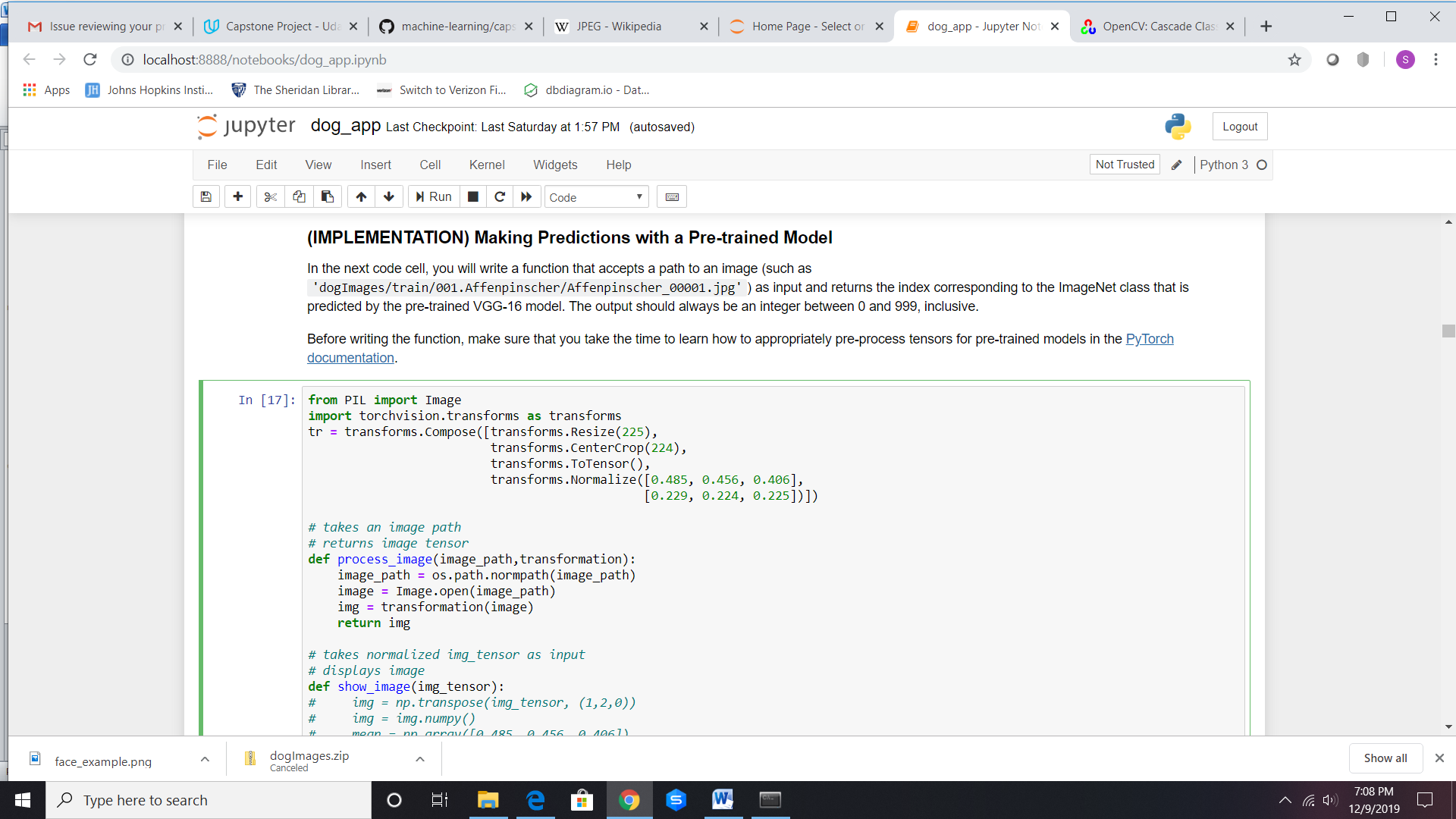
In order to interpret dog images, a pre-trained VGG16 neural network from ImageNet will be used. This network requires that RGB (red, green, blue scaled) images are pre-processed and converted into tensors based on specific normalized values. In this case, it will require that the pixel values in these tensors are weighted with a specific mean and standard deviation. For each RGB value for each pixel in an image, this calculation will be done:

mean = np.array([0.485, 0.456, 0.406])

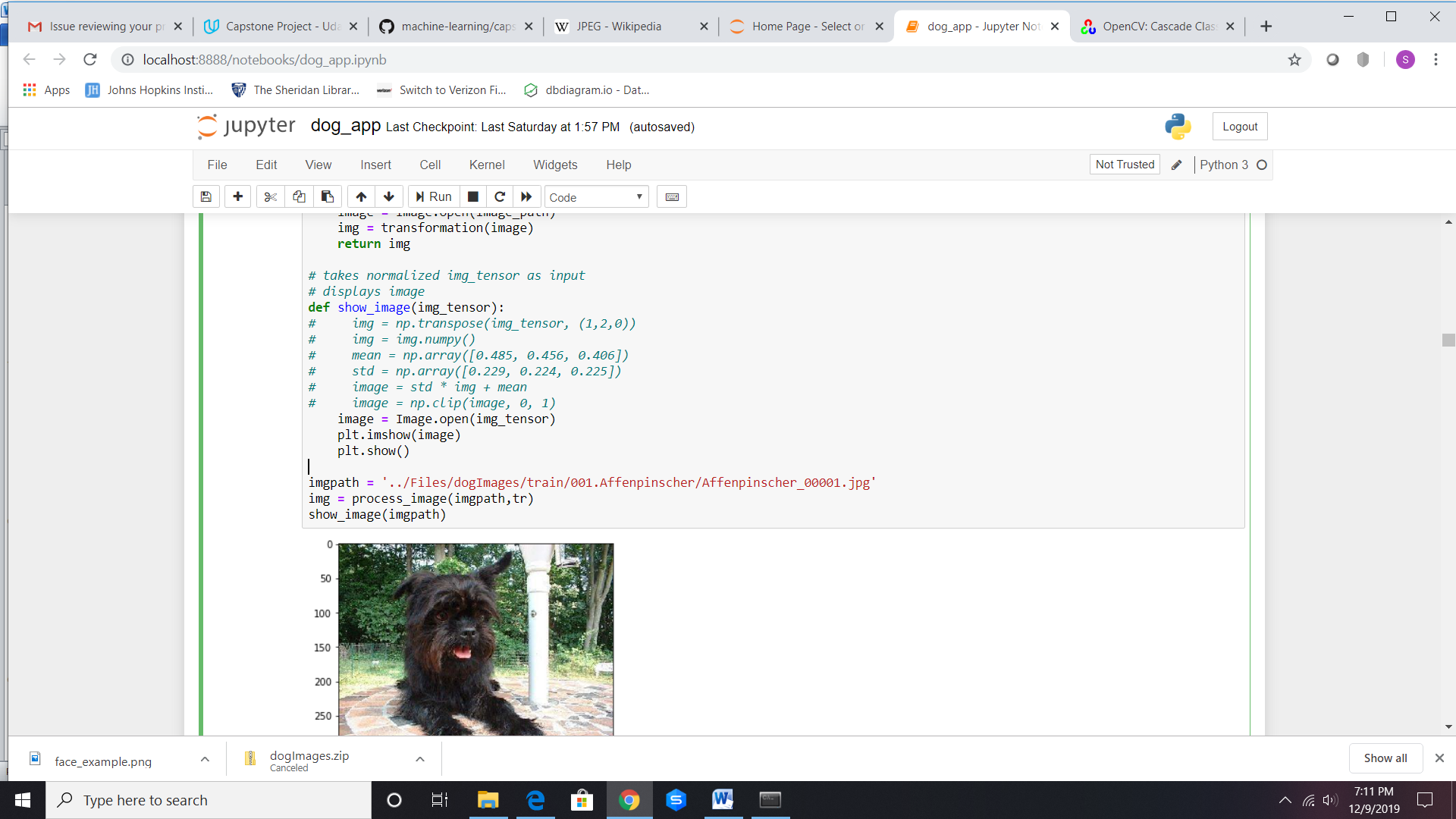
# std = np.array([0.229, 0.224, 0.225])

# img\_tensor = (image – mean) / std

This can also be achieved by transforming the images with the built-in modules of torchvision:



The result is an image tensor which can be used to feed into the neural network. In order to verify that the appropriate transformations are done on the image, the same calculations can be applied to the tensor but in the reverse order to show the image, however in order to keep things simple, the PIL module is used to show images for data visualization:





**Algorithms and Techniques:**

There are many different kinds of pre-trained convolutional neural networks, along with the associated hyperparameters for each one. When trying to figure out how exactly to go about it, it is important to look at the actual network architecture for each pre-trained model. In this case, the VGG16 convolutional architecture was recommended for its simplicity.



Figure 2: VGG16 architecture taken from https://neurohive.io/en/popular-networks/vgg16/

Based on the diagram, each layer involves several filters before it is condensed (pooled) and split into its respective channels and layers. After making its way through all the convolutional layers, the fully connected layer makes sense of the final convoluted input. Notice that the term “ReLU” is used frequently in this diagram. This is known as the “Rectified Linear Unit” and it is an important transformation that is applied to the final filtered output to give the results a form of “non-linearity”[1]. This is because the features are scaled based on shading values, however because each value contains an extremely unique weighted value, the filtered images carry a somewhat linear structure to how they are composed. Applying this type of transformation helps generalize the pixels and “blend” similar features to take away from some of that linearity:

Figure 3: ReLU function applied to convolutional layer to introduce non-linearity[1]

After passing our data through this pre-trained network, a classifying component based on our dataset will be used to actually train the model on our data. In order to do this, our own subset architecture must be defined along with a loss function and learning rate. For our classifier, 2 hidden layers of scalable values relative to the amount of classes are used for balancing purposed. A negative-likelihood loss function is also used to calculate the amount of loss the algorithm outputs when making inferences about the correct class. This is useful for 1D-tensors with large final output layers (in our case 133)[2]. The optimizer that will be used will be an Adam optimizer with a learning rate of 0.0005. The smaller the learning rate, the more computationally expensive the training will be. However this is a trade-off, as the models tend to be more accurate.

The benchmark for this model is an accuracy of greater than at least 60%. We want our model to perform well, however due to the amount of similarities between dog breeds, expecting an accuracy of close to 100% is not realistic. 60% is a good benchmark because it assumes that at our algorithm can make correct inferences on at least over half our dataset, or around 80 dog breeds.