Homework 6
Lusine Kamikyan
BME 595
Collaborated with Emma Reid

Part A: Consists of train.py
train.py

At the beginning I use *argparse* to read the command line and get the directory where the data is and the directory where the trained model will be saved

I define ***alexnet*** class:
The following are the functions inside alexnet class:

__init__(nn.module):
I define the alexnet architecture here:
Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
ReLU(inplace=True),
MaxPool2d(kernel_size=3, stride=2),
Conv2d(64, 192, kernel_size=5, padding=2),
ReLU(inplace=True),
MaxPool2d(kernel_size=3, stride=2),
Conv2d(192, 384, kernel_size=3, padding=1),
ReLU(inplace=True),
Conv2d(384, 256, kernel_size=3, padding=1),
ReLU(inplace=True),
Conv2d(256, 256, kernel_size=3, padding=1),
ReLU(inplace=True),
MaxPool2d(kernel_size=3, stride=2),
Dropout(),
Linear(256 * 6 * 6, 4096),
ReLU(inplace=True),
Dropout(),
Linear(4096, 4096),
ReLU(inplace=True),
Linear(4096, 200)

forward_train(self, image):
Takes in the data and moves it through the layers of the neural network

Train(self):

My batch size is 100, both for training and validation.
We need to put the validation images in the right folder formats, similar to how the training data is using the val_annotations.txt file.
I also create a dictionary of validation classes that I will use to output correct word for the image instead of the n########## name.
I load the training and validation data. I have used a code from https://www.kaggle.com/carloalbertobarbano/vgg16-transfer-learning-pytorch. I normalize the images, resize them. Train data gets shuffled, the validation data doesn't get shuffled.
I then load the pretrained alexnet model, and save the weights on my model, except the last linear layer.
I also freeze the weight on all the layers except the last linear layer

I used Adam optimizer with learning rate 10^-3 and my loss function is cross entropy.
Then I iterate through the epochs for both train and validation (similar to the previous hw) and calculate the loss functions for both testing and validation, then plot them.
I save the trained model as alexnet_model.pth.tar using torch.save in the directory provided in the command line (after each epoch)

Here are the results:
As we can see the loss functions are going down as epochs increase.
I run it for 10 epochs. However, the part A plots are for 5 epochs since I originally run for 5 epochs since it takes very long time to run it and save the plots. Later I decided to run 10 epochs, however, at the end of the last epoch when it was supposed to plot the figures, it didn't. So unfortunately I can't show the loss plots for 10 epochs, however the results for part B are based on the trained model as a result of running for 10 epochs

As we can see the loss for both test and validation go down with each epoch

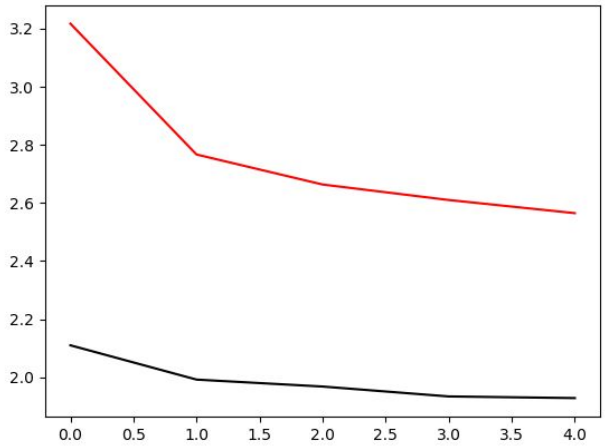


Figure1: Training Loss
Validation Loss

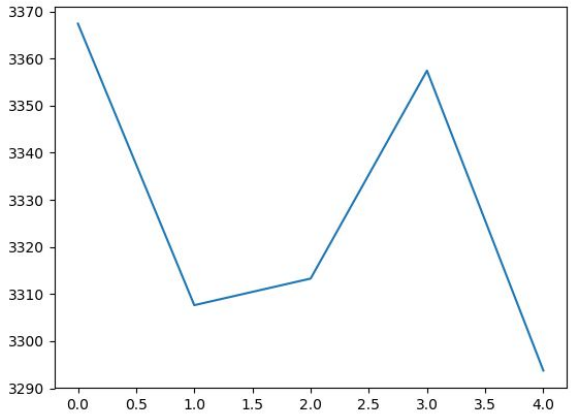


Figure2: Epoch vs Time

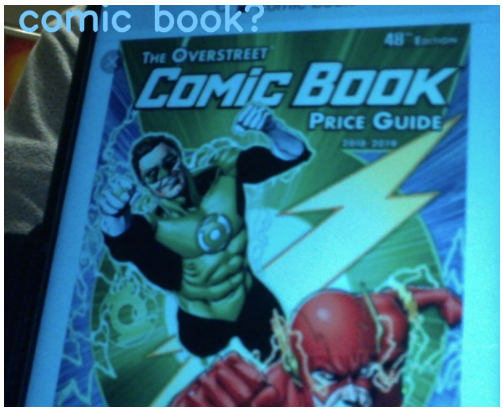
Part B: Consists of test.py
I again use the argparse to obtain the directory where the saved model is.

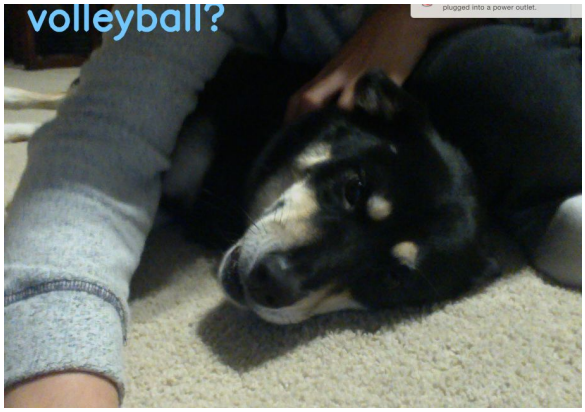
*Class **test*** has the following functions:
__init__(self):
I call the alexnet model and load the trained model from part A

forward(self, image):
Takes in an image from the camera feed, converts to a torch tensor, resizes it using view and passes it through the forward pass of trained model. Returns the predicted name corresponding to the image (we use the validation classes dictionary we created in part A to obtain the names)

cam(index = 0):
The same as previous hw, only change the sizes of the images and the normalization to make it match with the data we have this time.

Here are the results. Some predictions that are correct and some that are not:




volleyball?
plugged into a power outlet.