Lusine Kamikyan
Homework 5
BME 595A
Collaborated with Emma Reid

Part A

train():
This trains on the MNIST dataset.
My Batch_size is 500. I use the MSE error and SGD optimizer with learning rate 0.5. I run through 10 epochs.

The following is the architecture of CNN:

- conv2d(1, 6, kernel_size=5,stride = 1, padding = 2). It takes in an image of 1x28x28 and makes it into 6x28x28 (added padding to make it 28x28)
- MaxPool(kernel_size=2, stride =2) - makes the 6x28x28 into a size of 6x14x14
- Tanh() - activation function
- conv2d(6, 16, kernel_size=5,stride = 1). It takes in an image of 6x14x14 and makes it into 16x10x10 (0 padding)
- MaxPool(kernel_size=2, stride =2) - makes the 16x10x10 into a size of 16x5x5
- Tanh() - activation function
- I use view(-1,16*5*5) to change into 1 dimensional vector (1x16*5*5) for Fully connected layers
- Linear(16*5*5,120) layer to make the 16*5*5 into a vector of 120 size
- Tanh() - activation function
- Linear(120,84) layer to make the 120 into a vector of 84 size
- Tanh() - activation function
- Linear(84,10) layer to make the 84 into the final output vector of size 10

To be able to write the code for CNN in python (couldn't change the dim of the image from 2d to 1d for FC layers if only used nn.Sequence), I added a private function __forward_train() that takes an image and moves it through the layers of the network.

forward([28x28 ByteTensor] img):
Takes in an image of size 28x28 byte tensor, converts it into a float tensor, changes the dimension from 28x28 into 1x1x28x28, then calls the forward

function to get the the prediction, returns an integer corresponding to the prediction.

Here are the plots for the training and test errors:
As we can see the losses go down and we can see convergence

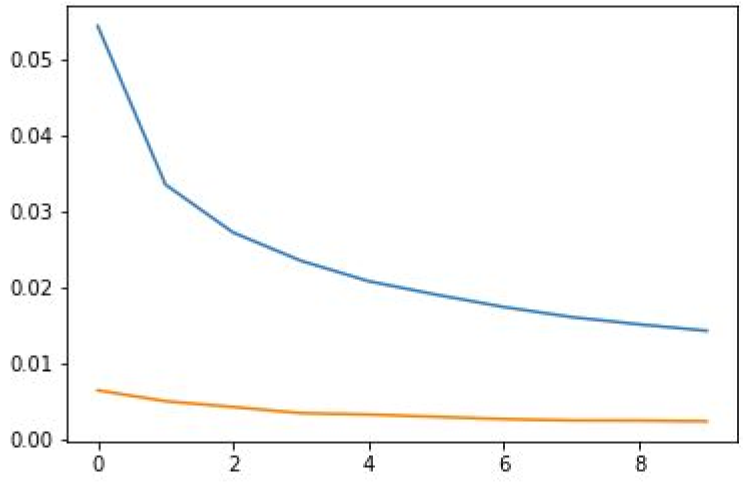


Figure 1: Epoch vs Train Loss
Epoch vs Test Loss

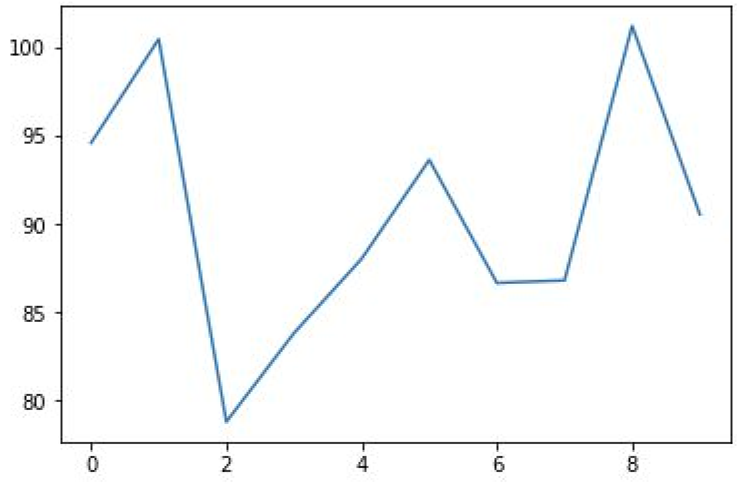


Figure 2: Epoch vs time taken to run the train and validation

The accuracy of CNN is much better than the Fully Connected NN. I run the FC in HW 4 for 10 epochs and got accuracy of 85.61%, however, running LeNet for only 10 epochs gave 96.83% accuracy. The error function, optimizer and learning rate were the same for both cases

Part B:

train():

The architecture of CNN is the same as above, except the first convolution layer has 0 padding, the number of input channels is 3 instead of 1, and the number of final output is 100 instead of 10.

I have 25 epochs. Batch size is 500. I used Adam for optimizer with learning rate 0.001. My loss is CrossEntropy loss.

I get 30.56% accuracy

Here are the results for the losses:

The losses go down and converge

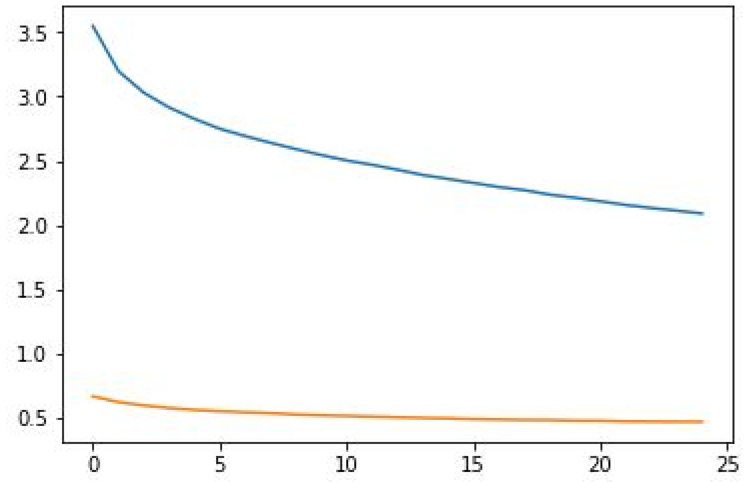


Figure 3: **Epochs vs train loss**
Epochs vs test loss

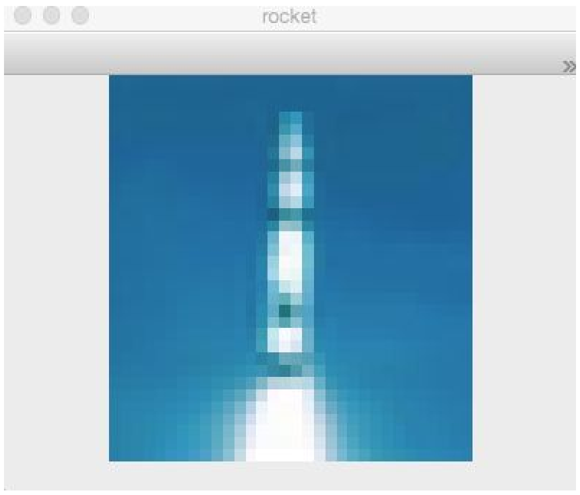
forward([3x32x32 ByteTensor] img):

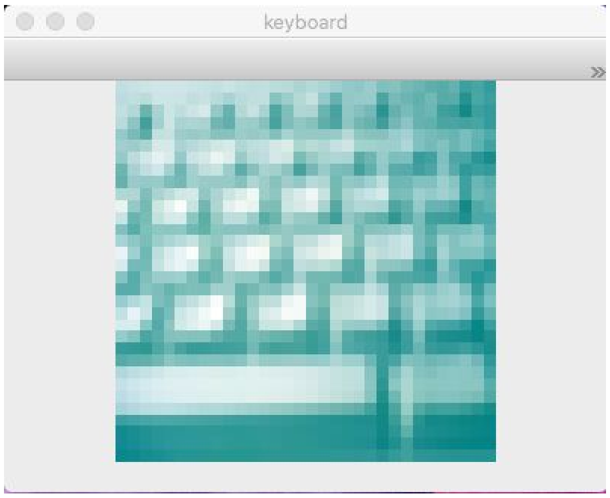
Takes in a 3x32x32 Byte tensor, converts to a float tensor, moves the image through the network to get the prediction and returns the label corresponding to the prediction

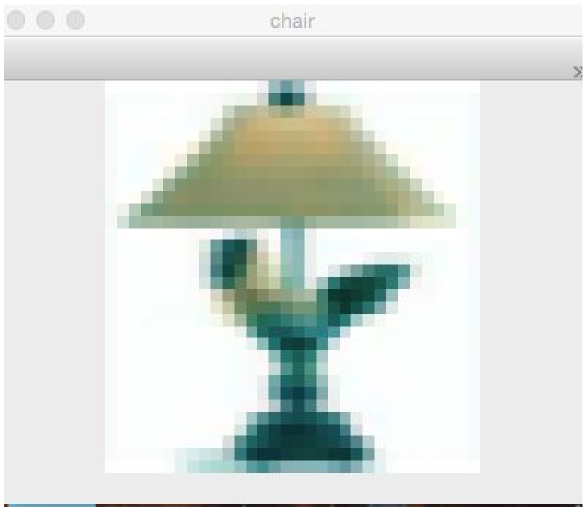
view([3x32x32 ByteTensor] img):

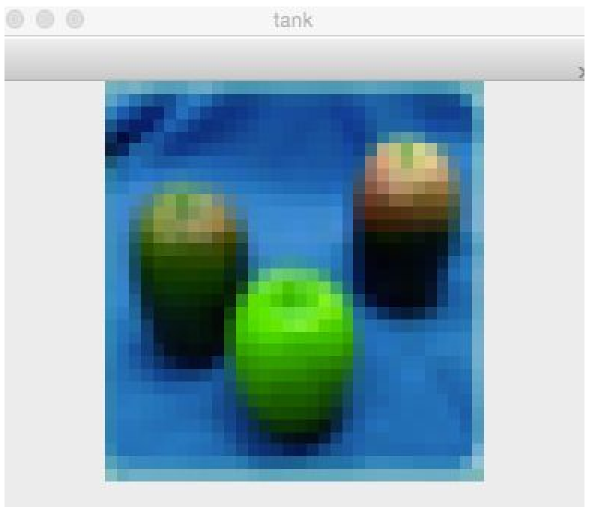
NOTE: The last minute I realized view takes in bytetensor and not float tensor, for some reason when I used a byte tensor image, the image would come either grey or black with a few pixels colored. So I kept my view to accept float tensor of size 3x32x32

Takes in a 3x32x32 FloatTensor, gets the prediction of the image label, converts the image into numpy, converts the image into the correct dimension, changes the colors from BRG to RGB, and uses cv2 to show the image with its predicted label.
Here are some correct and incorrect predictions:





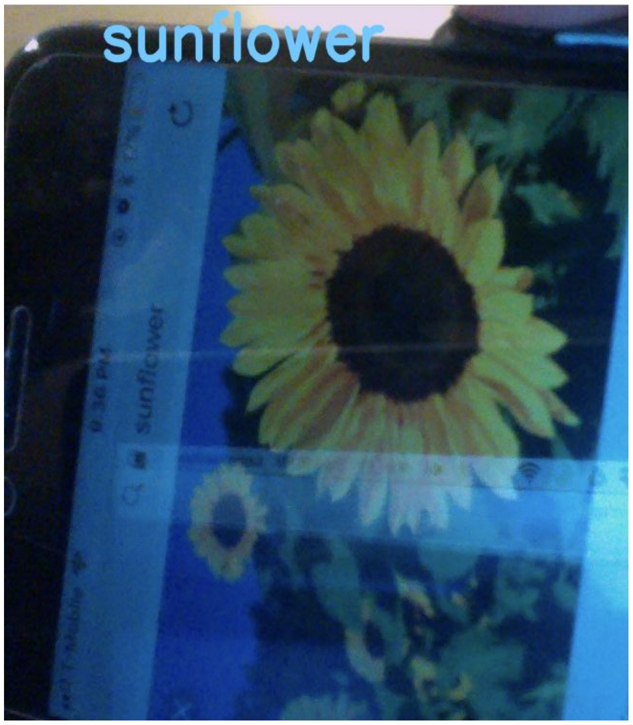


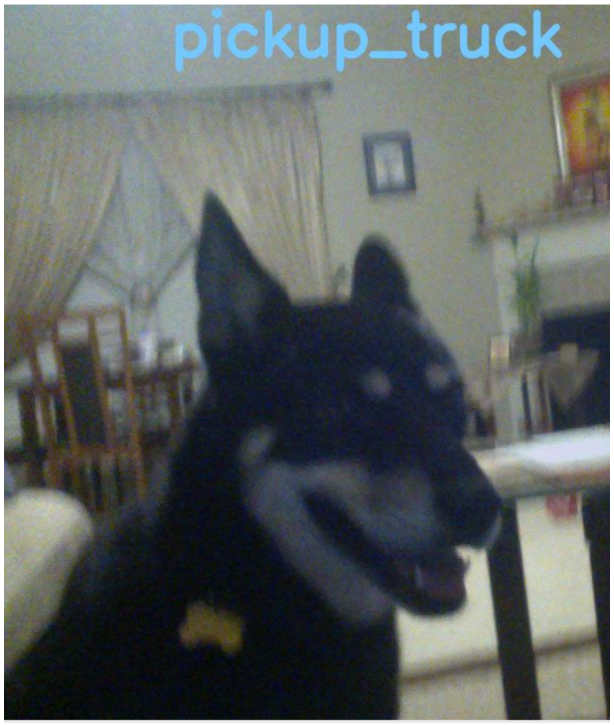


cam([int] /idx/):

The cam(index = 0) starts the camera feed, sets the camera widow parameters, in a while loop starts getting continuous frames from the camera (video), resizes the images to 32x32, transfers to tensor, normalizes, makes into the correct dimension, then calls the forward function, gets the predicted label, and shows the label on the camera feed live. To stop the camera, press ‘q’

Here are some correct and incorrect predictions:


sunflower
9:36 PM
sunflower


pickup_truck