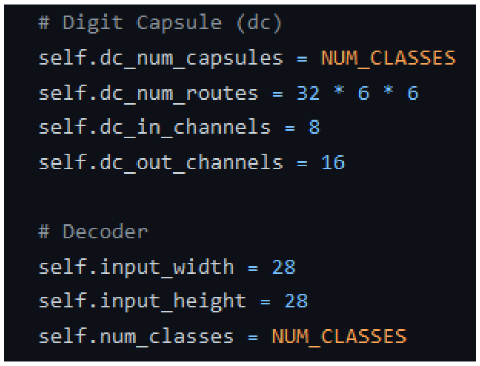
Individual Report

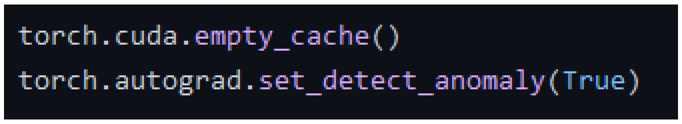
In this project, I worked on the following tasks:

In this project, I carried out coding tasks related to getting it to work on our dataset. I obtained the code implementation of capsule networks from capsule networks and modified the parameters to run it for the 43 classes relevant to our dataset instead of the original ten classes implemented for the MNIST dataset.

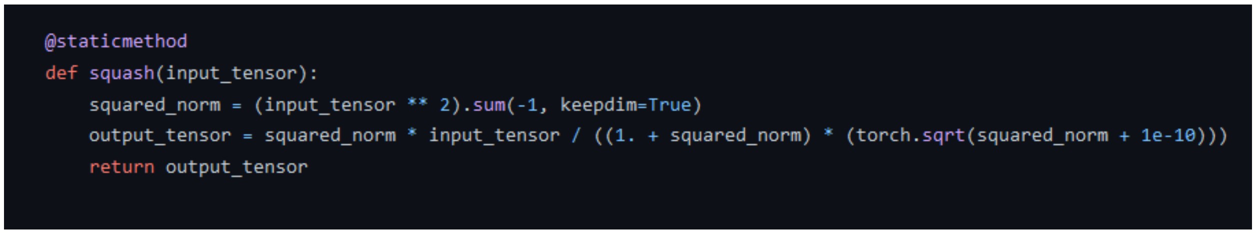


```
# Digit Capsule (dc)
self.dc_num_capsules = NUM_CLASSES
self.dc_num_routes = 32 * 6 * 6
self.dc_in_channels = 8
self.dc_out_channels = 16

# Decoder
self.input_width = 28
self.input_height = 28
self.num_classes = NUM_CLASSES
```

There was an issue with the loss function becoming NaN after a few iterations once I got the code to run. I spent about a week trying to figure out how to debug the issue and reached out to the Professor as well. Finally, what seemed to work was adding a Pytorch parameter to the script for anomaly detection which pointed out the exact point which led to the generation of NaNs. The problem was with the implementation of the sqrt function in torch, it is unstable while calculating the square root of zero. Adding a small value to the input helped fix the issue.

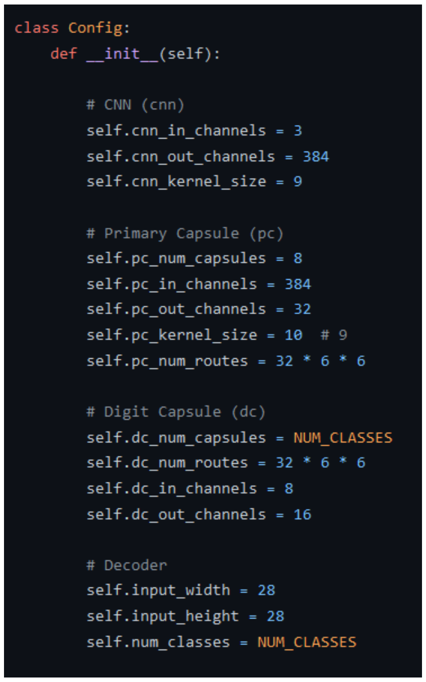


```
torch.cuda.empty_cache()
torch.autograd.set_detect_anomaly(True)
```



```
@staticmethod
def squash(input_tensor):
    squared_norm = (input_tensor ** 2).sum(-1, keepdim=True)
    output_tensor = squared_norm * input_tensor / ((1. + squared_norm) * (torch.sqrt(squared_norm + 1e-10)))
    return output_tensor
```

I also tuned the parameters in the config file to get the best-optimized results for the capsule network.

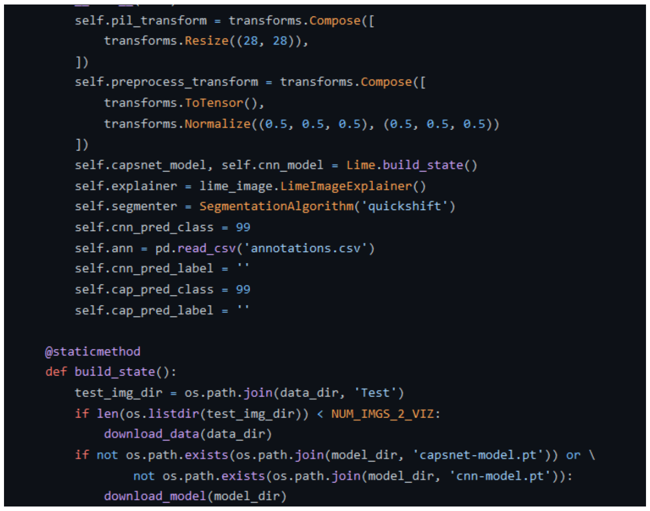


```python
class Config:
    def __init__(self):

        # CNN (cnn)
        self.cnn_in_channels = 3
        self.cnn_out_channels = 384
        self.cnn_kernel_size = 9

        # Primary Capsule (pc)
        self.pc_num_capsules = 8
        self.pc_in_channels = 384
        self.pc_out_channels = 32
        self.pc_kernel_size = 10  # 9
        self.pc_num_routes = 32 * 6 * 6

        # Digit Capsule (dc)
        self.dc_num_capsules = NUM_CLASSES
        self.dc_num_routes = 32 * 6 * 6
        self.dc_in_channels = 8
        self.dc_out_channels = 16

        # Decoder
        self.input_width = 28
        self.input_height = 28
        self.num_classes = NUM_CLASSES
```

I added a utils script to create helper functions like downloading the pretrained model and data directory from Kaggle. I also created a plotting function to graph the loss, accuracy, and F1 scores.

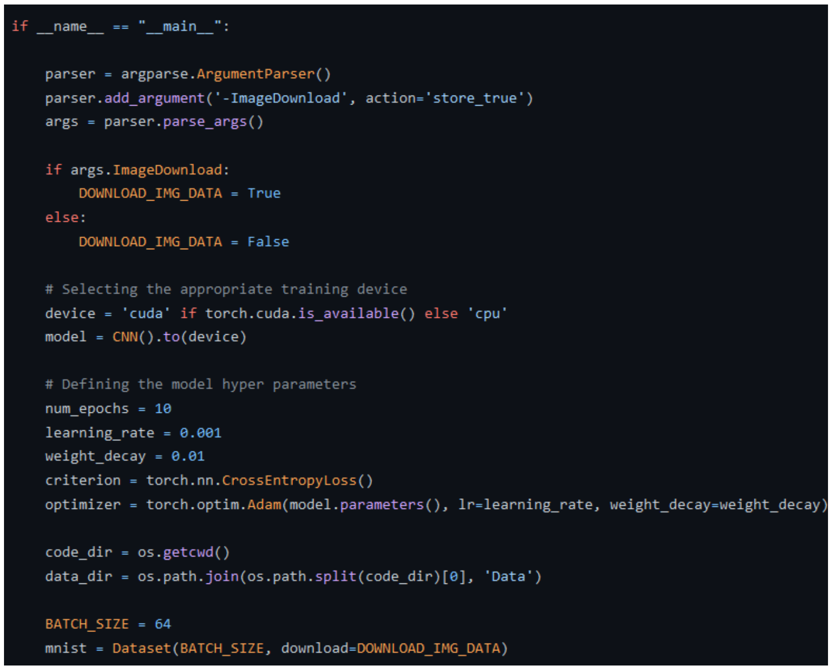
Utils code link

I converted the explain.py script for LIME into a class object to make it more streamlined and easy to use for the two different models.



```python
        self.pil_transform = transforms.Compose([
            transforms.Resize((28, 28)),
        ])
        self.preprocess_transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])
        self.capsnet_model, self.cnn_model = Lime.build_state()
        self.explainer = lime_image.LimeImageExplainer()
        self.segmenter = SegmentationAlgorithm('quickshift')
        self.cnn_pred_class = 99
        self.ann = pd.read_csv('annotations.csv')
        self.cnn_pred_label = ''
        self.cap_pred_class = 99
        self.cap_pred_label = ''

    @staticmethod
    def build_state():
        test_img_dir = os.path.join(data_dir, 'Test')
        if len(os.listdir(test_img_dir)) < NUM_IMGS_2_VIZ:
            download_data(data_dir)
        if not os.path.exists(os.path.join(model_dir, 'capsnet-model.pt')) or \
                not os.path.exists(os.path.join(model_dir, 'cnn-model.pt')):
            download_model(model_dir)
```

I converted two out of the three executable scripts: Baseline_CNN and test_capsnet into executable scripts so they can be run from the command line.



```
if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    parser.add_argument('-ImageDownload', action='store_true')
    args = parser.parse_args()

    if args.ImageDownload:
        DOWNLOAD_IMG_DATA = True
    else:
        DOWNLOAD_IMG_DATA = False

    # Selecting the appropriate training device
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    model = CNN().to(device)

    # Defining the model hyper parameters
    num_epochs = 10
    learning_rate = 0.001
    weight_decay = 0.01
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

    code_dir = os.getcwd()
    data_dir = os.path.join(os.path.split(code_dir)[0], 'Data')

    BATCH_SIZE = 64
    mnist = Dataset(BATCH_SIZE, download=DOWNLOAD_IMG_DATA)
```

Finally, I learned about the capsnet architecture from online sources so I can effectively work with the code and make changes where required, and understand the complete workflow.

For a list of all of my contributions to the project, please access my GitHub profile at this link: GitHub repo