HW5 Yaroslav Popryho
ypopry2@uic.edu

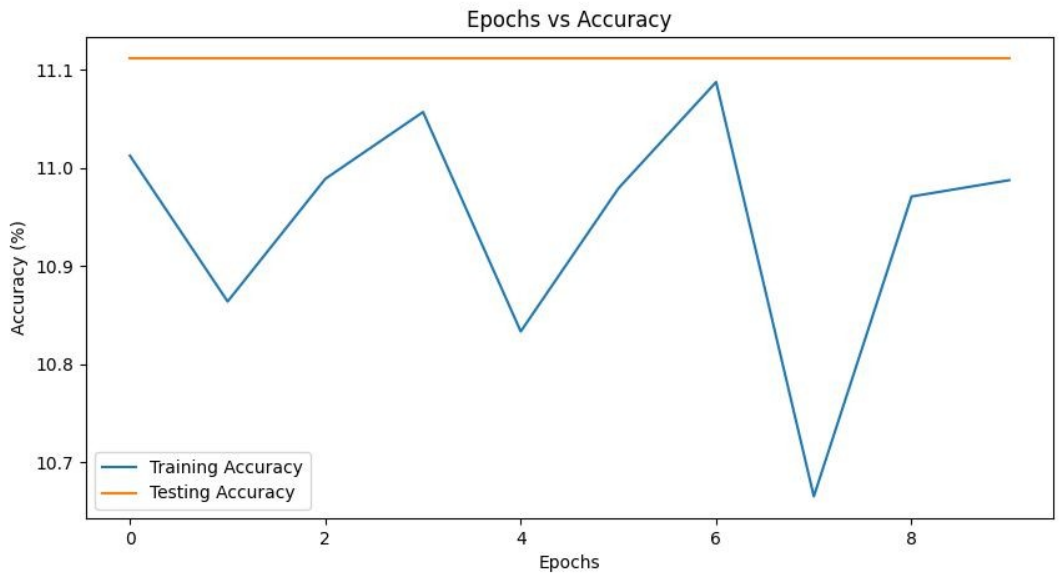
– The details of your neural network design process, what did you try, what worked, what did not... The optimizer, the loss function, and other hyperparameters that you utilized.

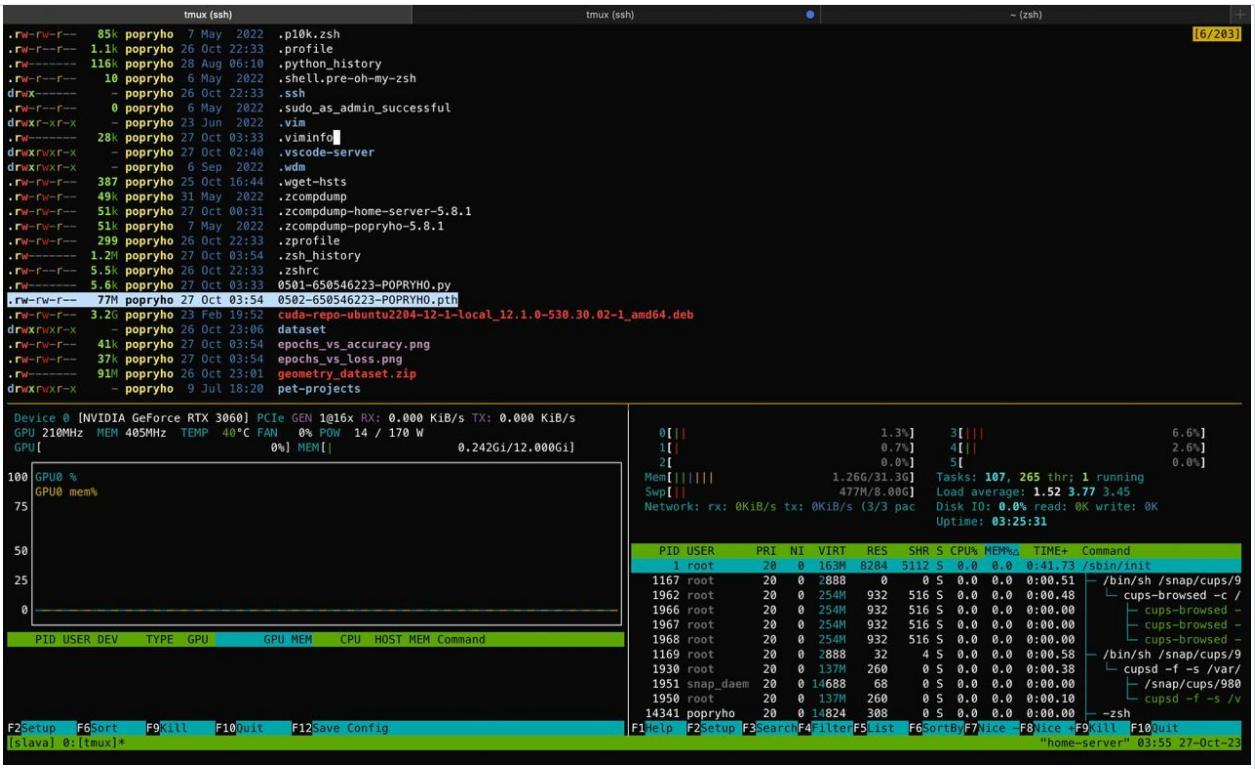
In the first experiment, I utilized the following architecture:

- 4 convolutional layers with channels increasing from 32 to 256.
- 3 max-pooling layers.
- 2 fully connected layers, with a dropout of 50% before the final layer.

Hyper parameters:

- Learning rate: 0.001.
- Batch size: 132.
- Optimizer: Adam
- Loss function: Cross Entropy Loss



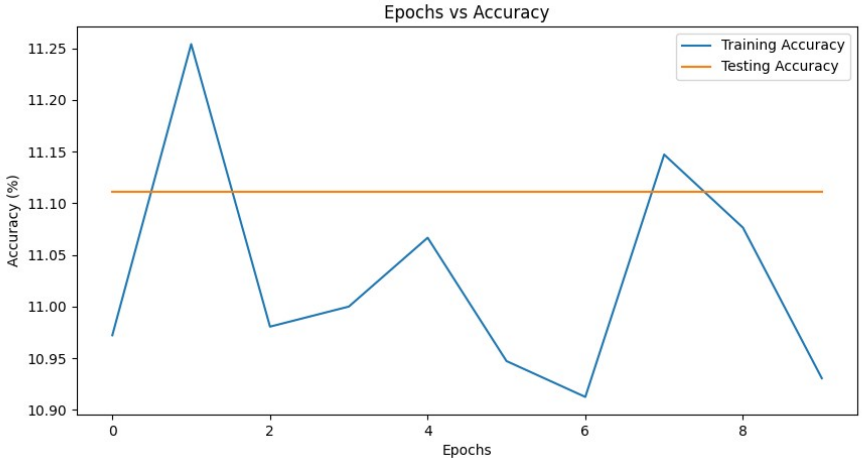


Result: The loss on the test set was higher than on the training set, indicating overfitting. Surprisingly, the training accuracy was also lower than expected. Additionally, the model size (77 MB) exceeded the acceptable limit.

For the 2nd experiment, I was using the same `model` architecture from experiment 1, but just modified the training parameters:

- Set learning rate to 0.01.

Result: The testing accuracy remained constant across epochs, suggesting that the model wasn't learning effectively or had reached a local minimum.



I then opted for a simpler model with following changes:

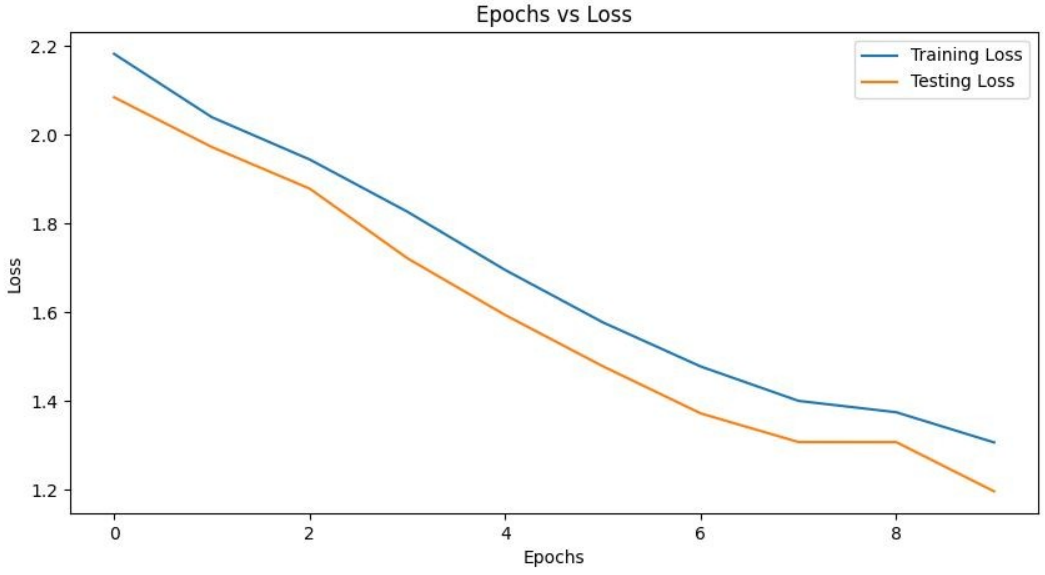
- Reduced the number of channels in convolutional layers.
- Removed one convolutional layer.
- Adjusted the size of the fully connected layers.

Hyper parameters:

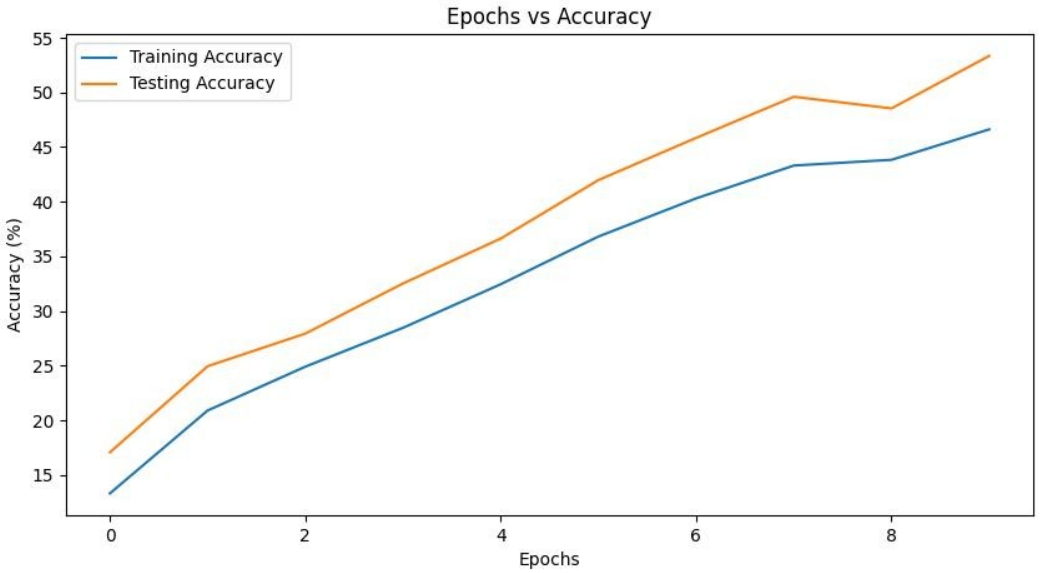
- Batch size: 32.
- Learning rate: 0.001.

Result: This model showed the expected behavior with increasing accuracy and decreasing loss across epochs for both training and testing sets, signifying improved generalization and performance.

- A graph that shows epochs vs loss on training set, and on the same graph epoch vs loss on the set set.

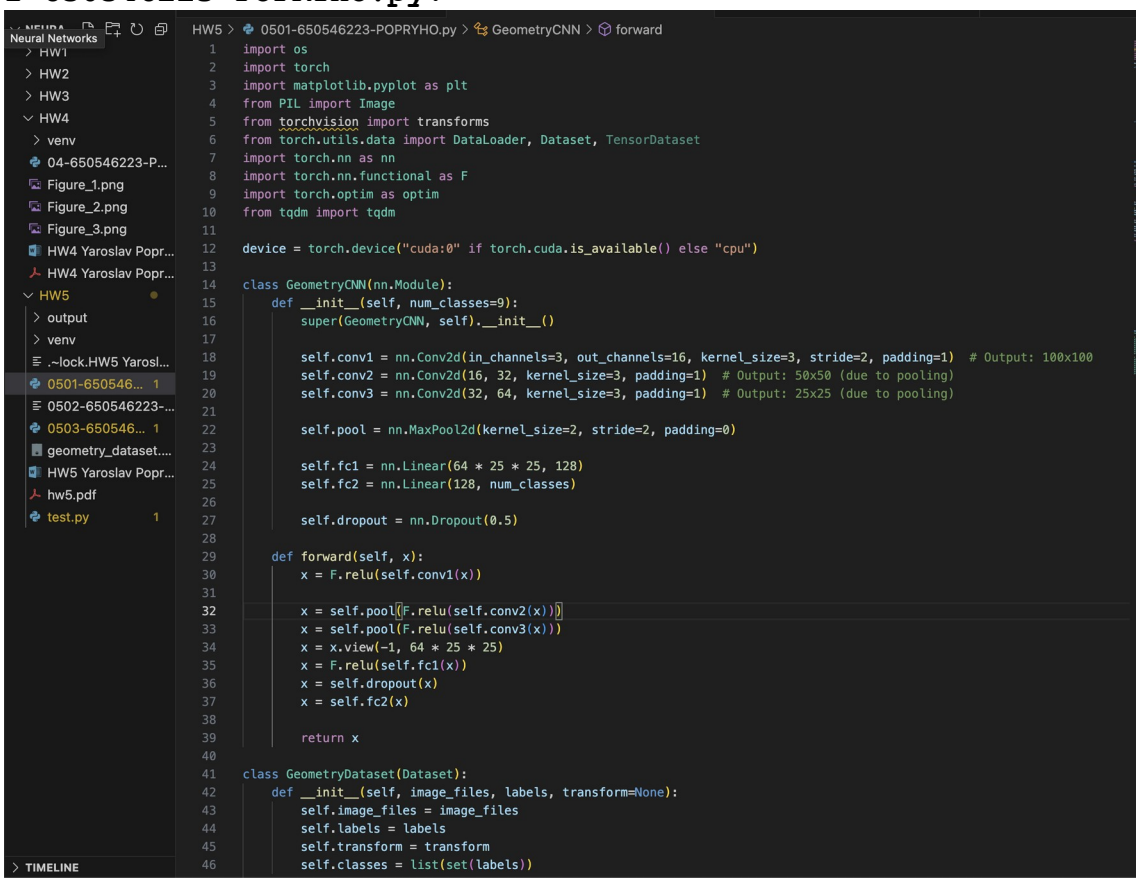


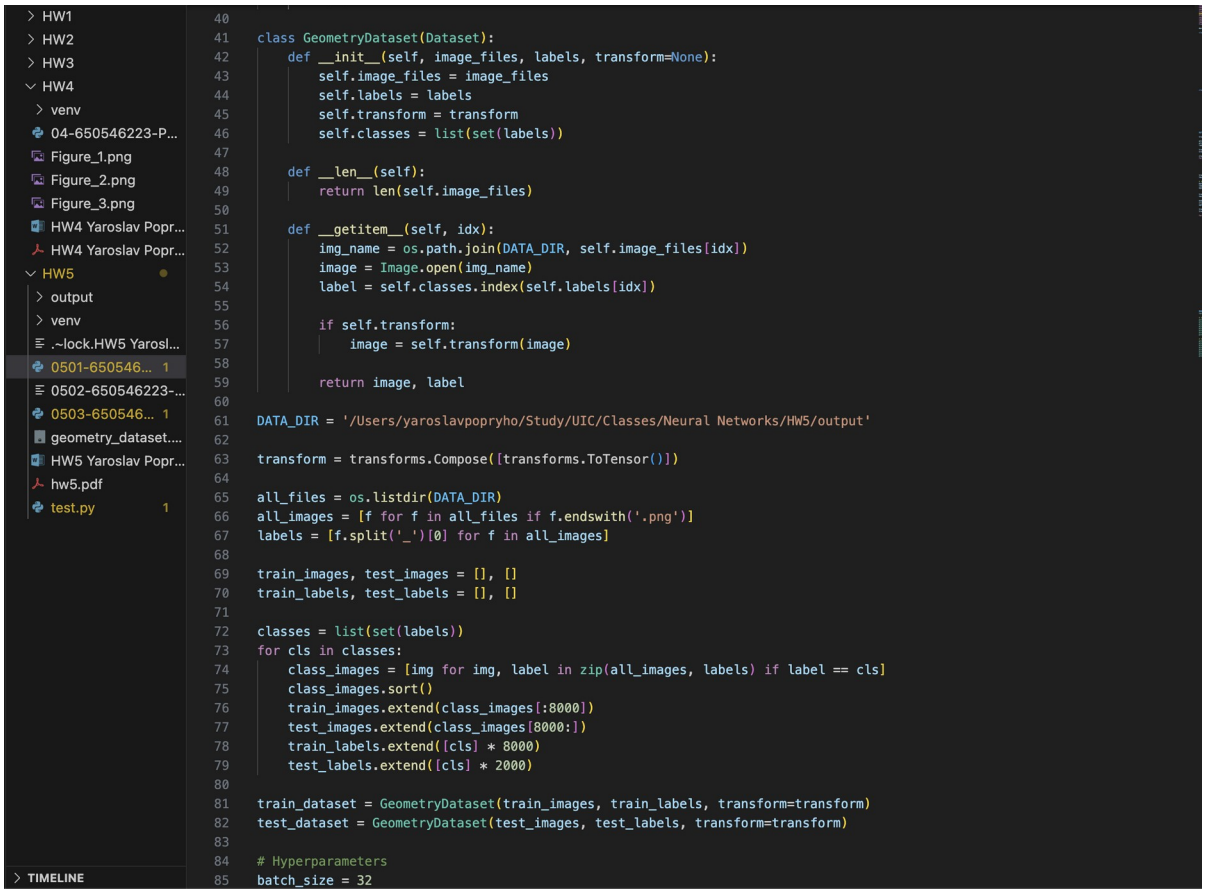
- A graph that shows epochs vs accuracy on training set, and on the same graph epoch vs accuracy on the set set. - All your codes

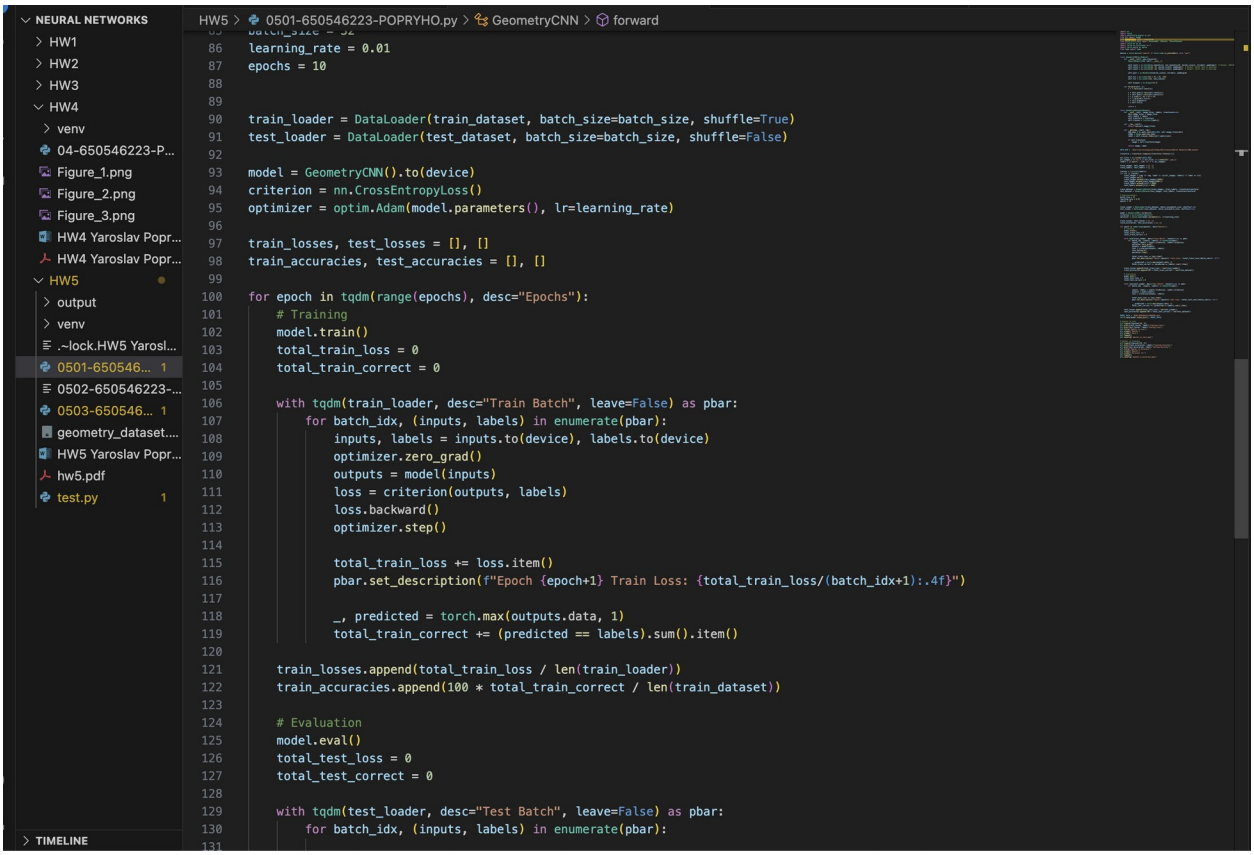


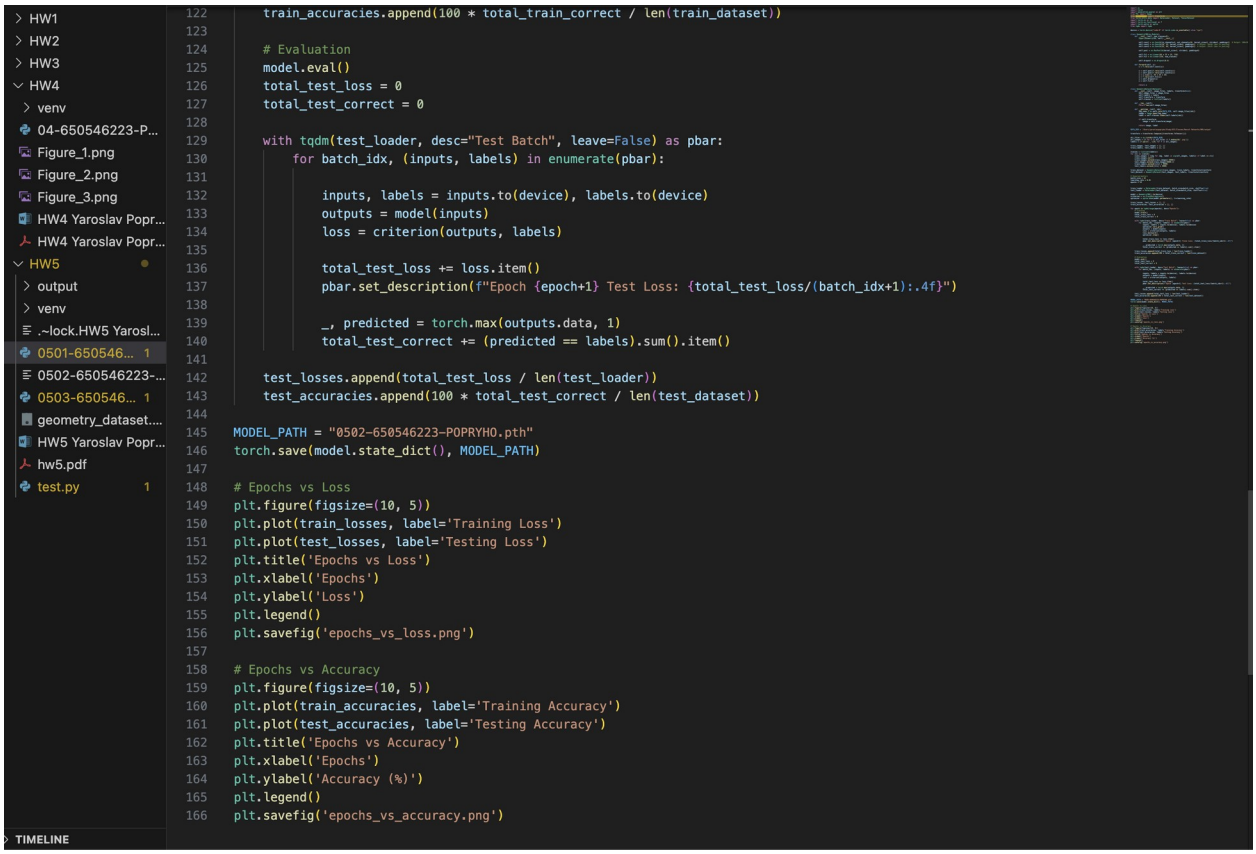
- All your codes.

0501-650546223-POPRYHO.py:

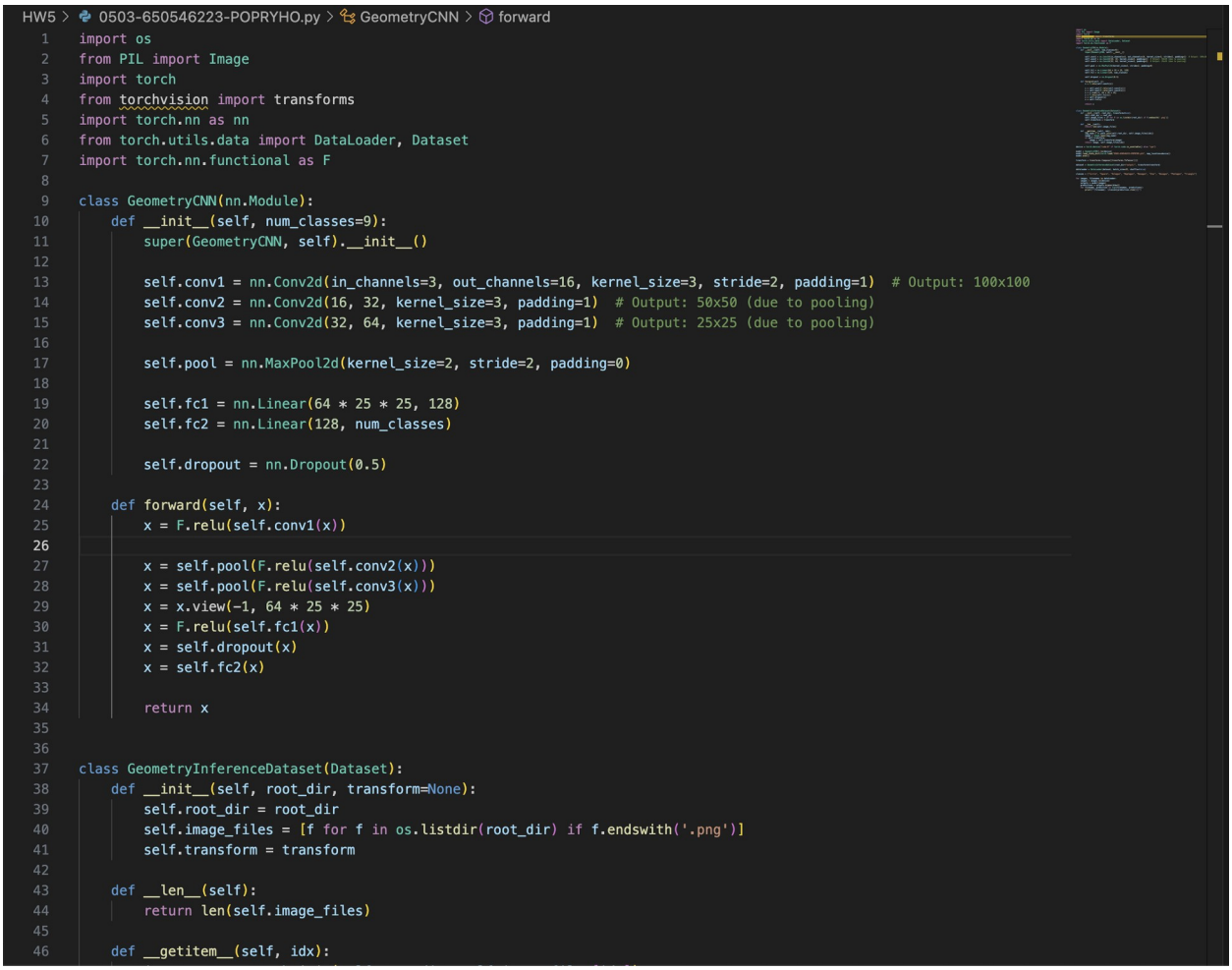



HW1
HW2
HW3
HW4
venv
04-650546223-P...
Figure_1.png
Figure_2.png
Figure_3.png
HW4 Yaroslav Popr...
HW4 Yaroslav Popr...
HW5
output
venv
.~lock.HW5 Yarosl...
0501-650546... 1
0502-650546223-...
0503-650546... 1
geometry_dataset....
HW5 Yaroslav Popr...
hw5.pdf
test.py 1
TIMELINE
40
41 class GeometryDataset(Dataset):
42 def __init__(self, image_files, labels, transform=None):
43 self.image_files = image_files
44 self.labels = labels
45 self.transform = transform
46 self.classes = list(set(labels))
47
48 def __len__(self):
49 return len(self.image_files)
50
51 def __getitem__(self, idx):
52 img_name = os.path.join(DATA_DIR, self.image_files[idx])
53 image = Image.open(img_name)
54 label = self.classes.index(self.labels[idx])
55
56 if self.transform:
57 image = self.transform(image)
58
59 return image, label
60
61 DATA_DIR = '/Users/yaroslavpopryho/Study/UIC/Classes/Neural Networks/HW5/output'
62
63 transform = transforms.Compose([transforms.ToTensor()])
64
65 all_files = os.listdir(DATA_DIR)
66 all_images = [f for f in all_files if f.endswith('.png')]
67 labels = [f.split('_')[0] for f in all_images]
68
69 train_images, test_images = [], []
70 train_labels, test_labels = [], []
71
72 classes = list(set(labels))
73 for cls in classes:
74 class_images = [img for img, label in zip(all_images, labels) if label == cls]
75 class_images.sort()
76 train_images.extend(class_images[:8000])
77 test_images.extend(class_images[8000:])
78 train_labels.extend([cls] * 8000)
79 test_labels.extend([cls] * 2000)
80
81 train_dataset = GeometryDataset(train_images, train_labels, transform=transform)
82 test_dataset = GeometryDataset(test_images, test_labels, transform=transform)
83
84 # Hyperparameters
85 batch_size = 32


NEURAL NETWORKS
HW1
HW2
HW3
HW4
venv
04-650546223-P...
Figure_1.png
Figure_2.png
Figure_3.png
HW4 Yaroslav Popr...
HW4 Yaroslav Popr...
HW5
output
venv
.~lock.HW5 Yarosl...
0501-650546... 1
0502-650546223-...
0503-650546... 1
geometry_dataset....
HW5 Yaroslav Popr...
hw5.pdf
test.py 1
TIMELINE
HW5 > 0501-650546223-POPRYHO.py > GeometryCNN > forward
batch_size = 32
learning_rate = 0.01
epochs = 10
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
model = GeometryCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
train_losses, test_losses = [], []
train_accuracies, test_accuracies = [], []
for epoch in tqdm(range(epochs), desc="Epochs"):
# Training
model.train()
total_train_loss = 0
total_train_correct = 0
with tqdm(train_loader, desc="Train Batch", leave=False) as pbar:
for batch_idx, (inputs, labels) in enumerate(pbar):
inputs, labels = inputs.to(device), labels.to(device)
optimizer.zero_grad()
outputs = model(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
total_train_loss += loss.item()
pbar.set_description(f"Epoch {epoch+1} Train Loss: {total_train_loss/(batch_idx+1):.4f}")
_, predicted = torch.max(outputs.data, 1)
total_train_correct += (predicted == labels).sum().item()
train_losses.append(total_train_loss / len(train_loader))
train_accuracies.append(100 * total_train_correct / len(train_dataset))
# Evaluation
model.eval()
total_test_loss = 0
total_test_correct = 0
with tqdm(test_loader, desc="Test Batch", leave=False) as pbar:
for batch_idx, (inputs, labels) in enumerate(pbar):


HW1
HW2
HW3
HW4
venv
04-650546223-P...
Figure_1.png
Figure_2.png
Figure_3.png
HW4 Yaroslav Popr...
HW4 Yaroslav Popr...
HW5
output
venv
.~lock.HW5 Yarosl...
0501-650546... 1
0502-650546223-...
0503-650546... 1
geometry_dataset....
HW5 Yaroslav Popr...
hw5.pdf
test.py 1
TIMELINE
train_accuracies.append(100 * total_train_correct / len(train_dataset))
# Evaluation
model.eval()
total_test_loss = 0
total_test_correct = 0
with tqdm(test_loader, desc="Test Batch", leave=False) as pbar:
for batch_idx, (inputs, labels) in enumerate(pbar):
inputs, labels = inputs.to(device), labels.to(device)
outputs = model(inputs)
loss = criterion(outputs, labels)
total_test_loss += loss.item()
pbar.set_description(f"Epoch {epoch+1} Test Loss: {total_test_loss/(batch_idx+1):.4f}")
_, predicted = torch.max(outputs.data, 1)
total_test_correct += (predicted == labels).sum().item()
test_losses.append(total_test_loss / len(test_loader))
test_accuracies.append(100 * total_test_correct / len(test_dataset))
MODEL_PATH = "0502-650546223-POPRYHO.pth"
torch.save(model.state_dict(), MODEL_PATH)
# Epochs vs Loss
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(test_losses, label='Testing Loss')
plt.title('Epochs vs Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('epochs_vs_loss.png')
# Epochs vs Accuracy
plt.figure(figsize=(10, 5))
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(test_accuracies, label='Testing Accuracy')
plt.title('Epochs vs Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.savefig('epochs_vs_accuracy.png')

0503-650546223-POPRYHO.py



```
        self.root_dir = root_dir
        self.image_files = [f for f in os.listdir(root_dir) if f.endswith('.png')]
        self.transform = transform

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_name = os.path.join(self.root_dir, self.image_files[idx])
        image = Image.open(img_name)
        if self.transform:
            image = self.transform(image)
        return image, self.image_files[idx]

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = GeometryCNN().to(device)
model.load_state_dict(torch.load("0502-650546223-POPRYHO.pth", map_location=device))
model.eval()

transform = transforms.Compose([transforms.ToTensor()])

dataset = GeometryInferenceDataset(root_dir="output/", transform=transform)

dataloader = DataLoader(dataset, batch_size=32, shuffle=False)

classes = ["Circle", "Square", "Octagon", "Heptagon", "Nonagon", "Star", "Hexagon", "Pentagon", "Triangle"]

for images, filenames in dataloader:
    images = images.to(device)
    outputs = model(images)
    predictions = outputs.argmax(dim=1)
    for filename, prediction in zip(filenames, predictions):
        print(f"{filename}: {classes[prediction.item()]}")
```