Carnegie Mellon Unversity
17-644: Applied Deep Learning
Assignment# 2 Report

A. Visualizing Data

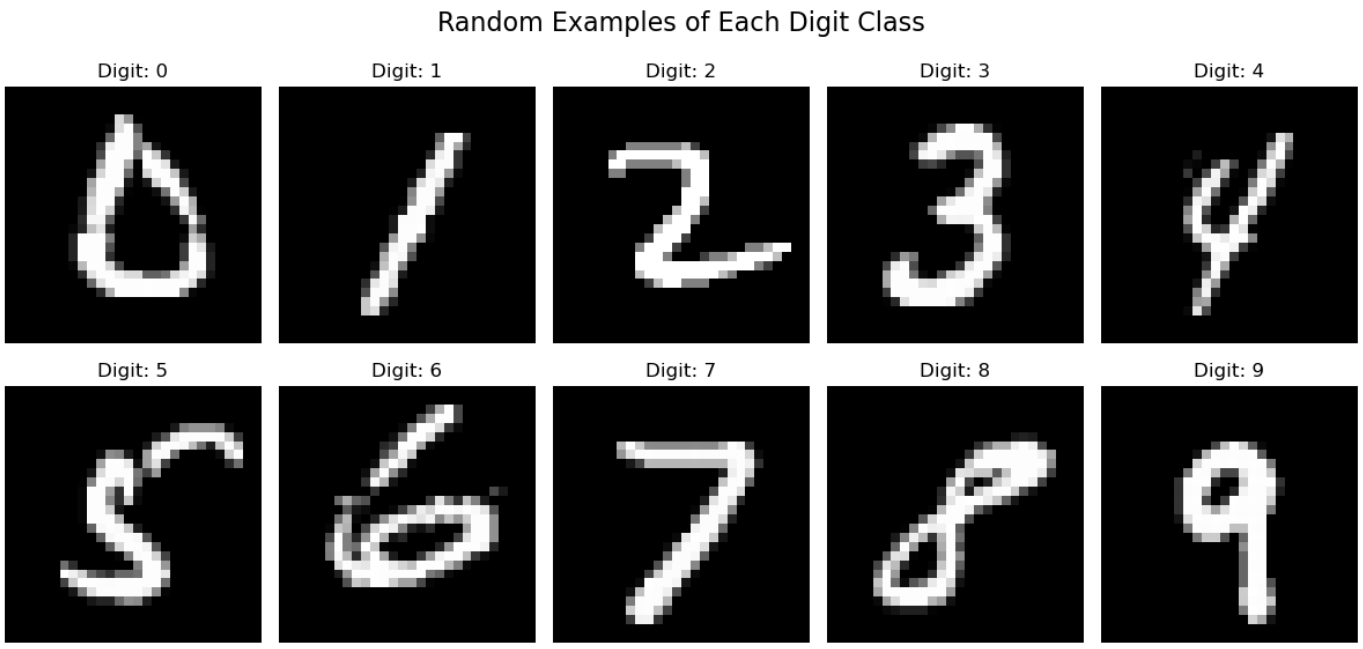


Figure 1 Random examples of each digit class in *MNIST Handwritten Digits* Dataset

```
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

fig, axes = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle('Random Examples of Each Digit Class', fontsize=16)
for i in range(10):
    ax = axes[i // 5, i % 5]
    class_indices = np.where(y_train == i)[0]
    random_index = np.random.choice(class_indices)
    ax.imshow(X_train[random_index], cmap='gray')
    ax.axis('off')
    ax.set_title(f"Digit: {i}")

plt.tight_layout()
plt.show()
```

Code 1 To get the number plots of a sample from one digit

B. Grayscale Histogram Analysis

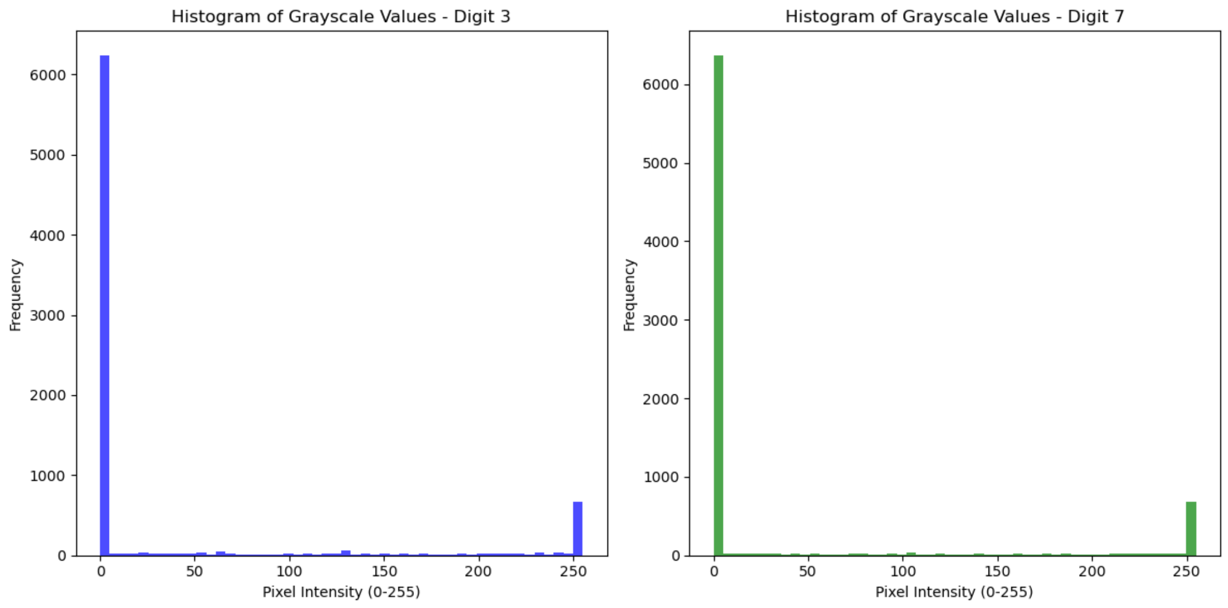


Figure 2 Histogram of a pair of numbers grayscale values (Digits 3 & 7)

Histograms look almost the same as each other. I wondered and tried with the other pairs, as well. The results were almost the same. Therefore, I ensured that all the digit pictures were somehow standardized with a majority of black and white colors.

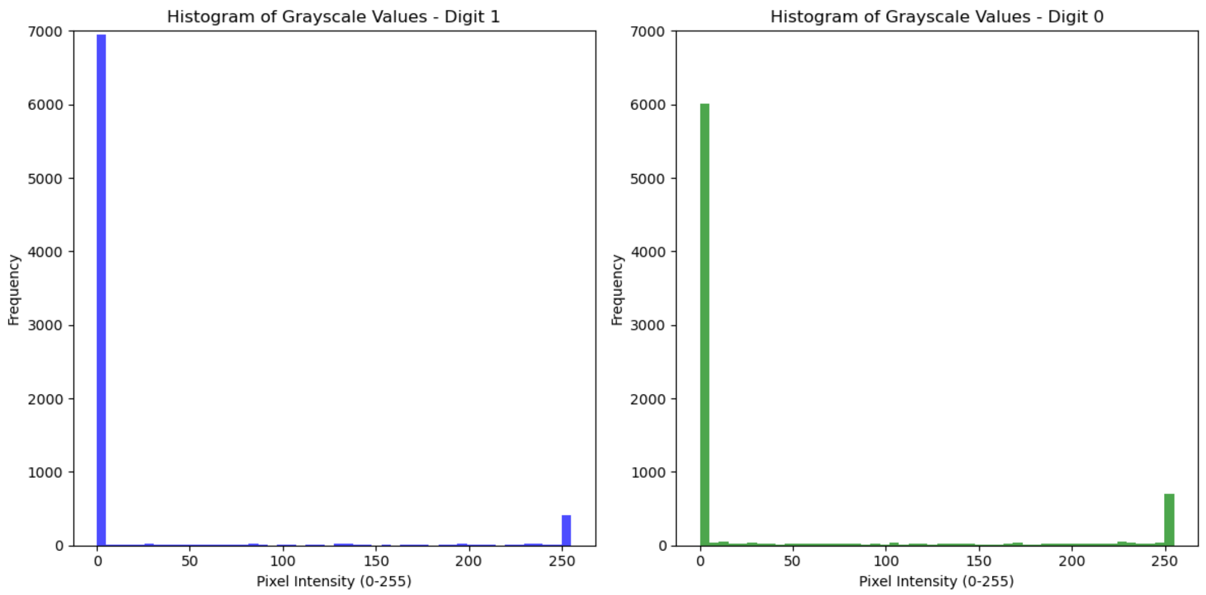


Figure 3 Random examples of each digit class in MNIST Handwritten Digits Dataset with a different digit pair (Digits 0 & 1)

However, I spotted a difference while comparing 0 and 1 (Figure 3). I assume the difference in black and white derives from their graphical structure. Digits like 0 might have more white pixels (high intensity), while 1 may have a simpler structure with fewer white pixels.

C. Building the Neural Network

Network Architecture selected:
- **Input**: 784 neurons (28x28 flattened image)
- **Hidden Layer 1**: 128 neurons, ReLU activation
- **Hidden Layer 2**: 64 neurons, ReLU activation
- **Output Layer**: 10 neurons, Softmax activation

At this step, the goal was to classify the MNIST images (28x28 pixels) into one of 10 classes (digits 0 to 9). I imagined the neural network would learn to map grayscale pixel values to the correct digit class.

I went with dense (fully connected) hidden layers, using ReLU as my activation function. I started with flattening all the pixels (784 in count) into a 1D vector. As the loss function, I chose "Sparse Categorical Crossentropy" after a quick research. I didn't know about this loss function but I learned that it's preferable when the labels are integers. As the last step, I decided to use ADAM as my optimizer.

Before I ran the model, I calculated the total number of learnable parameters *by hand* as instructed in the assignment brief.

$$Q = \sum_{i=1}^{L} n_i(n_{i-1} + 1)$$

In my case;

$$Q = \sum_{i=1}^{2} n_i(n_{i-1} + 1)$$

For the first hidden layer;

$$Q1 = (784\ x\ 128) + 128 = 100{,}480$$

For the second hidden layer;

$$Q2 = (128\ x\ 64) + 64 = 8{,}256$$

For the output layer;

$$Q3 = (64\ x\ 10) + 10 = 650$$

In total;

$$Q = Q1 + Q2 + Q3$$
$$\boldsymbol{Q = 109{,}386}$$

```
# normalize daya via scaling pixel values to 0-1 from 0-255
X_train_normalized = X_train / 255.0
X_test_normalized = X_test / 255.0

model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)), # convert 28*28 to 1d vector
    layers.Dense(128, activation='relu'), # 1st hidden layer
    layers.Dense(64, activation='relu'), # 2nd hidden layer
    layers.Dense(10, activation='softmax') # output layer for 10 classes (all digits classified)
])

model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

Code 2 Neural network model initialization code and output log

D. Training the Model

In my first attempt, I tried the parameters mentioned in the code below:

```
history = model.fit(X_train_normalized, y_train, epochs=20, validation_data=(X_test_normalized,
y_test),batch_size=64)
```

```
Epoch 1/20
2025-03-31 11:44:10.720826: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to
get CPU frequency: 0 Hz
938/938 [==============================] - 2s 1ms/step - loss: 0.2845 - accuracy: 0.9178 -
val_loss: 0.1386 - val_accuracy: 0.9569
Epoch 2/20
938/938 [==============================] - 1s 1ms/step - loss: 0.1130 - accuracy: 0.9657 -
val_loss: 0.1033 - val_accuracy: 0.9672
Epoch 3/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0770 - accuracy: 0.9760 -
val_loss: 0.0857 - val_accuracy: 0.9731
Epoch 4/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0572 - accuracy: 0.9817 -
val_loss: 0.0825 - val_accuracy: 0.9743
Epoch 5/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0448 - accuracy: 0.9857 -
val_loss: 0.0787 - val_accuracy: 0.9775
Epoch 6/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0360 - accuracy: 0.9881 -
val_loss: 0.0831 - val_accuracy: 0.9750
Epoch 7/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0292 - accuracy: 0.9905 -
val_loss: 0.0776 - val_accuracy: 0.9771
Epoch 8/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0232 - accuracy: 0.9926 -
val_loss: 0.0776 - val_accuracy: 0.9781
Epoch 9/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0207 - accuracy: 0.9931 -
val_loss: 0.0878 - val_accuracy: 0.9761
Epoch 10/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0182 - accuracy: 0.9939 -
val_loss: 0.0935 - val_accuracy: 0.9757
Epoch 11/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0149 - accuracy: 0.9950 -
val_loss: 0.0880 - val_accuracy: 0.9772
Epoch 12/20
```

```
938/938 [==============================] - 2s 2ms/step - loss: 0.0134 - accuracy: 0.9958 -
val_loss: 0.0939 - val_accuracy: 0.9779
Epoch 13/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0136 - accuracy: 0.9952 -
val_loss: 0.0968 - val_accuracy: 0.9788
Epoch 14/20
938/938 [==============================] - 1s 2ms/step - loss: 0.0120 - accuracy: 0.9959 -
val_loss: 0.0972 - val_accuracy: 0.9805
Epoch 15/20
938/938 [==============================] - 1s 2ms/step - loss: 0.0098 - accuracy: 0.9968 -
val_loss: 0.1012 - val_accuracy: 0.9791
Epoch 16/20
938/938 [==============================] - 1s 1ms/step - loss: 0.0106 - accuracy: 0.9964 -
val_loss: 0.1036 - val_accuracy: 0.9793
Epoch 17/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0105 - accuracy: 0.9964 -
val_loss: 0.1126 - val_accuracy: 0.9779
Epoch 18/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0087 - accuracy: 0.9970 -
val_loss: 0.1181 - val_accuracy: 0.9783
Epoch 19/20
938/938 [==============================] - 1s 2ms/step - loss: 0.0082 - accuracy: 0.9971 -
val_loss: 0.1258 - val_accuracy: 0.9762
Epoch 20/20
938/938 [==============================] - 2s 2ms/step - loss: 0.0092 - accuracy: 0.9970 -
val_loss: 0.1181 - val_accuracy: 0.9770
```

Code 2 Neural network model training code and the output log

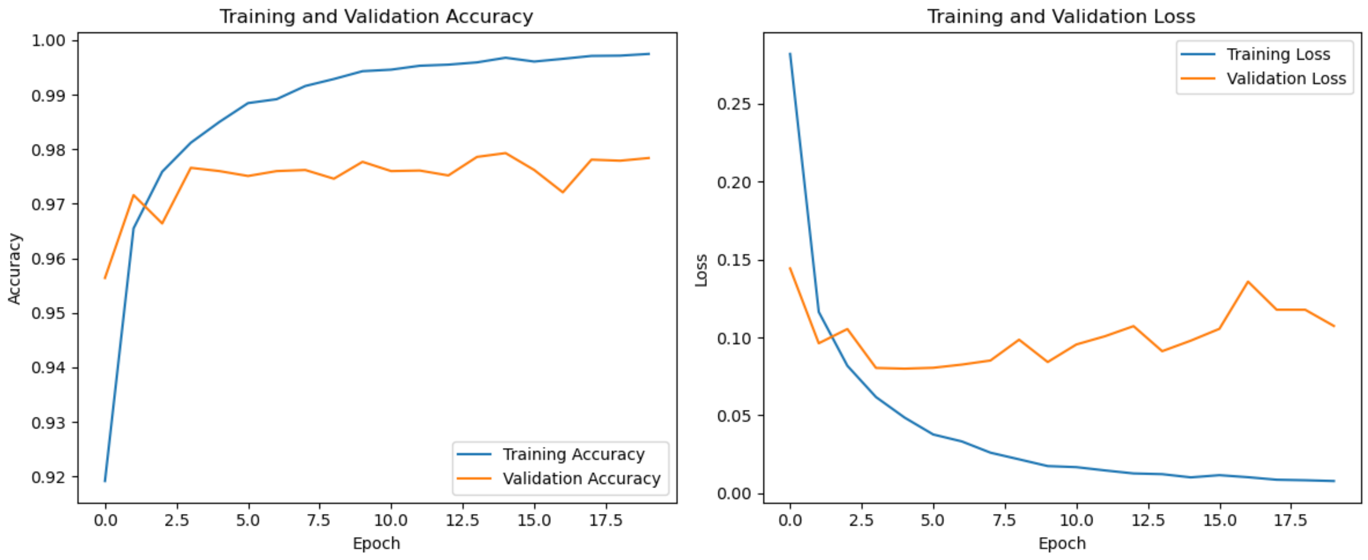


Figure 4 Training and validation accuracy and validation loss plot

Accuracy Plot
As visible on the accuracy plot and the output log, my training accuracy reaches nearly 100% while my validation accuracy improves and stabilizes at around 97-98%, but remains consistently lower than the training accuracy. This might signal a potential overfitting? Since it learns my training data very well but struggles to implement the logic to the validation data. We learned during the lectures that the widening gap between training and validation accuracy is a common indicator of overfitting.

Loss Plot
In this plot, training loss decreases steadily and approached to zero while validation loss initiallly decreases but then begins to rise around epoch 5 and fluctuates slightly. This is probably a great sign of overfitting. After a few epochs, I infer that the model memorizes the training data instead of learning meaningful general patterns. The increasing validation loss indicates that further training no longer improves generalization.

To improve the model, some options I can think of, deriving from the lecture notes, are early stopping, regularization, or dropout.

Since it's the easiest, I decided to conduct early stopping with 3 epochs (Figure 5). Hence, I achieved a good training and validation accuracy result, at the same time decreased the difference between them.

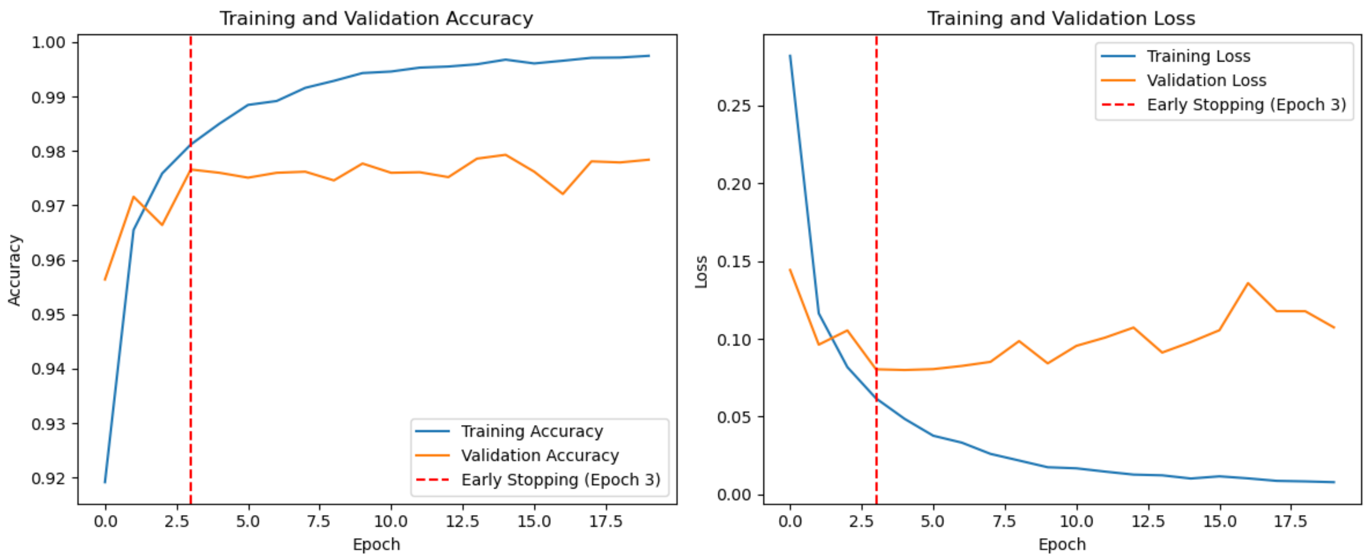
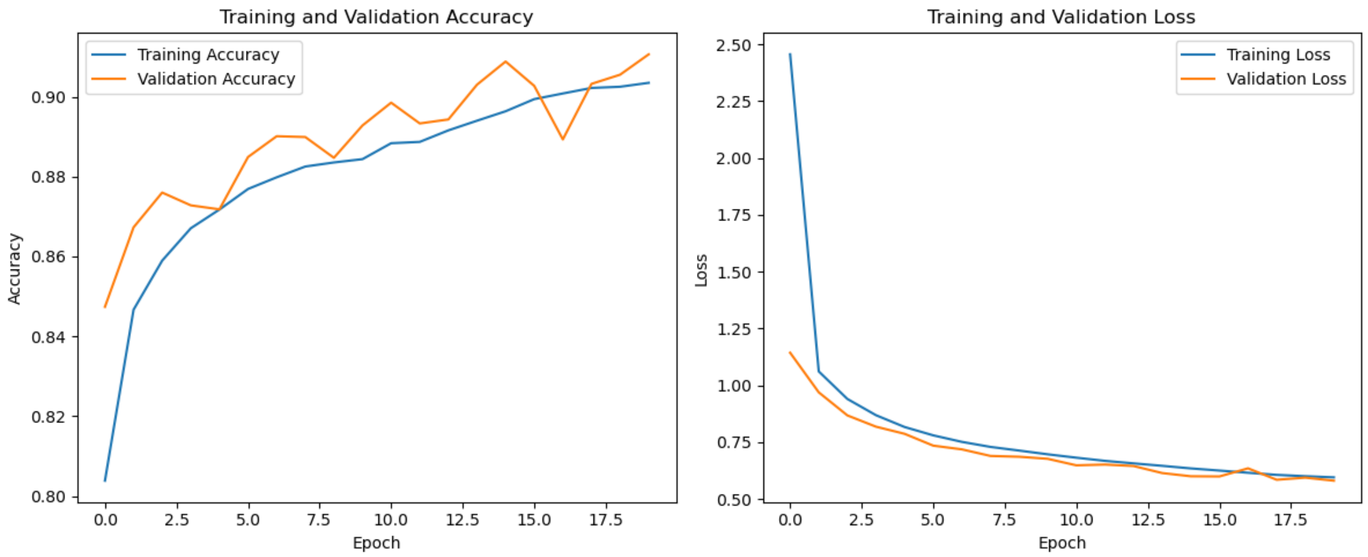


Figure 5 Early stopping results with 3 epochs

Then, out of curiosity, I wanted to try out using L2 regularization lambda value 0.01. Even though I trained my network with more epochs (20), I was able to prevent overfitting compared to the model without regularization.



E. Semester Project – Initial Ideas

For my semester project, I plan to explore multi-task deep learning for building façade classification and -if possible- metadata prediction using real-world imagery. The core deep learning tasks will include image classification (e.g., building construction type: brick, wood, glass façade) and -if possible- structured attribute prediction (e.g., number of stories, height, roof type, built year). My research question is: *Can a transfer learning-based CNN architecture extract useful features from façade images to predict both visual and non-visual metadata of buildings?*

Given this is my first hands-on deep learning project, I plan to initially focus on using pre-trained CNNs (e.g., ResNet50 or MobileNetV2) with a single-output classifier. As I gain confidence, I will explore a multi-task architecture that includes a secondary output for structured metadata.

Rather than generating my own dataset from Google Street View, I explore the possibility of using the dataset described in "Bounding Boxes Are All We Need" (https://arxiv.org/abs/2010.01305) as recommended by our TA Tushar, which is specifically curated for street-level image classification of buildings. This will allow me to avoid early bottlenecks in dataset collection and ensure the training images are of consistent quality. If time permits, I'd like to revisit the idea of integrating local metadata from the Allegheny County Real Estate Portal to simulate a small-scale multi-label learning setup. The outcome of this project will hopefully serve as a foundation for my master's thesis, which explores the use of deep learning in large-scale building stock analysis and energy-efficient urban planning.