Part IV: OCTMNIST Classification [20 points]

Preprocess the dataset by normalizing the pixel values to a standardized range, typically between 0 and 1.

The below converts the pixel values to standardized values between 0 and 1 which is performed using transforms.compose library

```
batch_size=128
info = INFO["octmnist"]
data_transform_valtest = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize(mean=[.5], std=[.5])
            ])
data_transform_train = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize(mean=[.5], std=[.5])
            ])
channels = info["n_channels"]
classes = len(info["label"])
DataClass = getattr(medmnist, info["python_class"])
train_dataset = DataClass(split="train", transform=data_transform_train, download=True)
val_dataset = DataClass(split="val", transform=data_transform_valtest, download=True)
test_dataset = DataClass(split="test", transform=data_transform_valtest, download=True)
train_loader = data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = data.DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=False)
test_loader = data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

Build a neural network:

```
class CNN(nn.Module):
  def __init__(self, in_channels, num_classes):
    super(CNN, self).__init__()
    self.features = nn.Sequential(
        nn.Conv2d(in_channels, 16, kernel_size=3),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.Conv2d(16, 16, kernel_size=3),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(16, 64, kernel_size=3),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.Conv2d(64, 64, kernel_size=3),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.Conv2d(64, 64, kernel_size=3, padding=1),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2)
    )
    self.fc = nn.Sequential(
        nn.Linear(64 * 4 * 4, 128),
        nn.ReLU(),
        nn.Linear(128, 128),
        nn.ReLU(),
        nn.Linear(128, num_classes)
    )
    self.dropout = nn.Dropout(p=0.5)
  def forward(self, x):
    x = self.features(x)
    x = self.dropout(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)
    return x
```

Architecture - 5 Conv2D layers, Linear Layers, Batch Normalization
Activation function -RELU

how they impact the performance:

a. Regularization: Apply regularization techniques, such as L1 or L2 regularization, to the model's parameters.

L2 regularization adds the squared magnitudes of the weights.
Here we apply L2 regularization as parameter to the SGD optimizer

```
self.model = CNN(in_channels=n_channels, num_classes=n_classes).cuda()
self.criterion = nn.CrossEntropyLoss()
self.optimizer = optim.SGD(self. model.parameters(), lr=learning_rate, momentum=0.9,weight_decay=l2_lambda)
self.train_accuracy_per_epoch = []
self.train_loss_per_epoch = []
self.val_accuracy_per_epoch = []
self.val_loss_per_epoch = []
self.test_accuracy_per_epoch = []
```

L2 regularization didnt significantly increase the accuracy or reduce the loss

b. Dropout: Introduce dropout layers between the fully connected layers.

```
    self.dropout = nn.Dropout(p=0.5)
def forward(self, x):
    x = self.features(x)
    x = self.dropout(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)
    return x
```

I introduced a dropout layer in the middle with the dropout value being 0.5

c. Early stopping: Monitor the performance of the model on a validation set

```
        val_acc,val_loss = self.test(val,0)
        self.val_accuracy_per_epoch.append(val_acc)
        self.val_loss_per_epoch.append(val_loss.cpu().numpy())
        if val_loss < self.best_val_loss:
          self.best_val_loss = val_loss
          self.counter = 0
        else:
          self.counter += 1
          if self.counter >= self.patience:
              print("Early stopping!")
              break
        test_accuracy, y_true,y_pred , loss = self.test(test=test_loader)
        self.test_accuracy_per_epoch.append(test_accuracy)
        self.test_loss_per_epoch.append(loss.cpu().numpy())
        cm = confusion_matrix(y_true.tolist(), y_pred.tolist())
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        plt.figure(figsize=(5,5))
        disp.plot()
        plt.xticks(rotation=90)
        plt.show()
```

Early Stopping code can be seen in the above picture:
I gave the patience value to be 5 here. Validation set accuracy didnt improve for 5 iterations consecutively, hence Early stopping is enabled and training stops at the epoch number 5

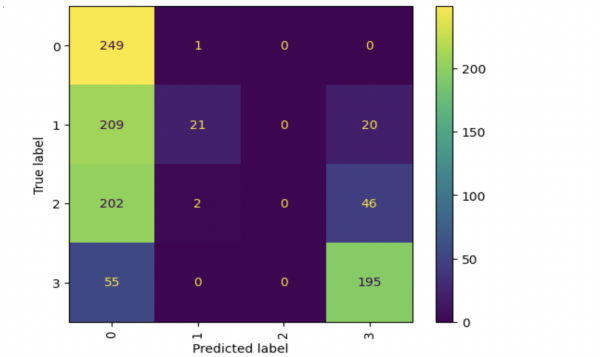
```
17 epoch

100%|██████████| 762/762 [00:30<00:00, 24.97it/s]

train accuracy 0.915364650122593
train loss tensor(0.0019, device='cuda:0', grad_fn=<DivBackward0>)
val accuracy 0.9078655834564254
val loss tensor(0.2830, device='cuda:0')
Early stopping!
```

We can see that the train and validation accuracies are both above 90

Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.

a) The training, testing, validation accuracies and losses looked like below the figure after the first epoch

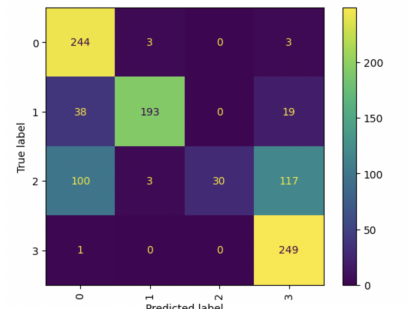
```
0 epoch

100%|██████████| 762/762 [00:30<00:00, 24.71it/s]

train accuracy 0.7302850928937082
train loss tensor(0.0057, device='cuda:0', grad_fn=<DivBackward0>)
val accuracy 0.767817577548006
val loss tensor(0.6941, device='cuda:0')
test accuracy 0.465
test loss tensor(1.4994, device='cuda:0')

<Figure size 500x500 with 0 Axes>
```



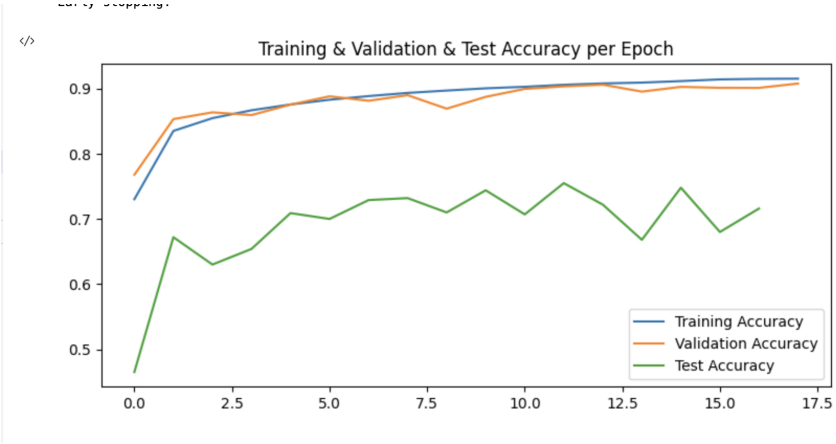
Training accuracy was 73. Testing accuracy is 46.5

b) I ran the program for 20 epochs but the model stopped after the 17th epoch as I enabled early stopping.
The below are the results for the last epoch:

```
16 epoch

100%|██████████| 762/762 [00:30<00:00, 25.19it/s]

train accuracy 0.9150876617048124
train loss tensor(0.0019, device='cuda:0', grad_fn=<DivBackward0>)
val accuracy 0.901218611521418
val loss tensor(0.3006, device='cuda:0')
test accuracy 0.716
test loss tensor(0.8030, device='cuda:0')

<Figure size 500x500 with 0 Axes>
```



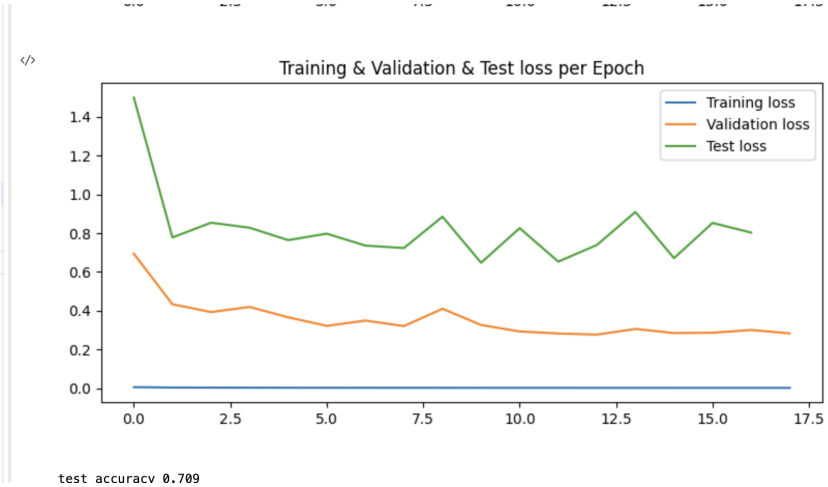
Plot the training and validation accuracy over time (epochs). & Plot the testing accuracy (epochs).



The above graph represents training , testing, and validation accuracies over time. You can see that training and validation have reached 90 almost in the end.

Testing data is struggling but is still above 70 by the end.

Plot the training and validation loss & testing loss over time (epochs).



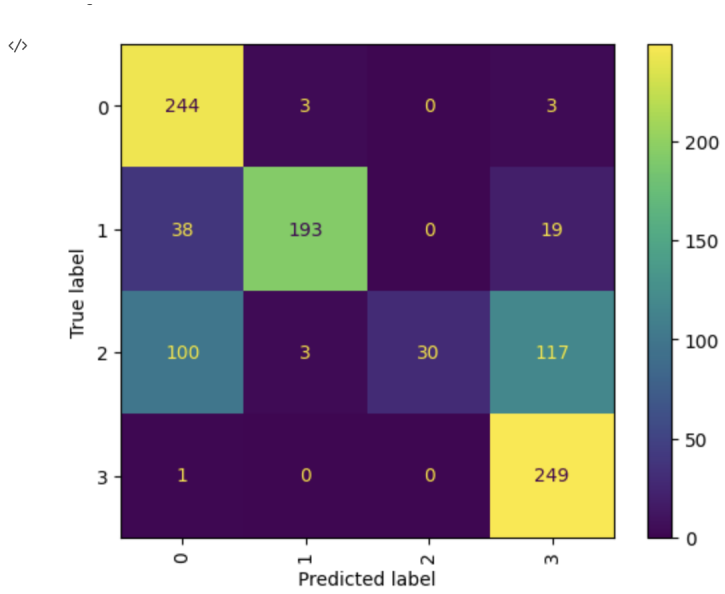
test accuracv 0.709

Training loss was almost zero from the beginning hence there isnt much difference.

Testing and Validation loss have surely gone down over time indicating the model has learned really well.

Generate a confusion matrix using the model's predictions on the test set.

We have got 4 classes and hence 4x4 matrix



Calculate and report other evaluation metrics such as precision, recall (scikit-learn), and F1 score (scikit-learn) to further analyze the model's performance on the test set.

Precision:

Precision measures the accuracy of a model in identifying positive instances. It focuses on minimizing false positives, indicating how often the model is correct when it predicts something as positive.

Recall:

Recall measures a model's ability to find all positive instances. It focuses on minimizing false negatives, indicating how often the model correctly identifies positive instances out of all actual positives.

F1 Score:

F1 score is the harmonic mean of precision and recall. It provides a single metric that balances the trade-off between precision and recall.

F1 score is useful when you want to consider both precision and recall.

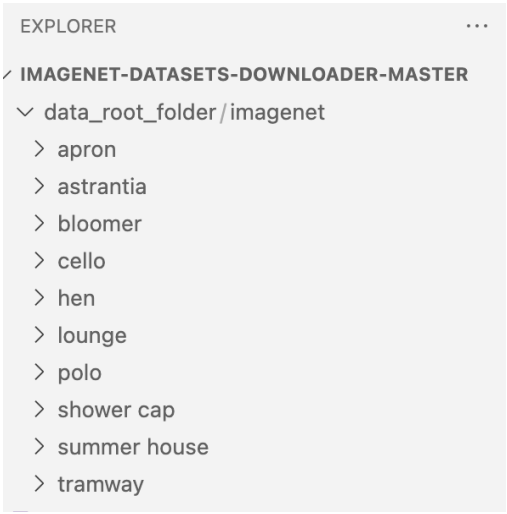
The below are the precision, recall, and F1 scores of the model:

```
Precision: 0.7681848196424106
Recall: 0.7090000000000001
F1 Score: 0.6760119683544141
```

The model has a precision of 76%, which means that when it predicts a positive instance, it is correct 76% of the time. The recall is 70%, indicating that the model is able to find 70% of the actual positive instances. The F1 score, which considers both precision and recall, is 67%. These metrics suggest that the model has relatively good accuracy in identifying positives, but it may miss some positive instances.

Part V: ImageNet Classification [30 points]

Choose at least 10 classes from ImageNet. Each class has to contain at least 500 images.



Randomly 10 classes were selected each containing 500 classes using https://github.com/skaldek/ImageNet-Datasets-Downloader

Preprocess the dataset for training (e.g. removing missing images, normalizing, split between training/testing/validation)

The below code shows the normalization that's been done on the images:

```python
from PIL import Image
import os
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torchvision import models


def normalize_images(root_folder):
    for folder_name in os.listdir(root_folder):
        folder_path = os.path.join(root_folder, folder_name)
        if not os.path.isdir(folder_path):
            continue

        print(f"Processing folder: {folder_name}")
        for image_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image_name)
            if os.path.isfile(image_path):
                # Normalize the image (e.g., resize, convert to grayscale, etc.)
                img = Image.open(image_path)

                # Replace the following lines with your desired normalization techniques

                img = img.resize((224, 224))  # Resize the image to a specific size
                img = img.convert('L')  # Convert the image to grayscale

                # Save the normalized image, overwrite the original image if desired
                img.save(image_path)


root_folder = "/Users/anurimavaishnavikumar/Desktop/ImageNet-Datasets-Downloader-master/data_root_folder/imagenet"
normalize_images(root_folder)
```

Split between test, train, validation is done as follows:

```python
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
root_dir = "/Users/anurimavaishnavikumar/Desktop/ImageNet-Datasets-Downloader-master/data_root_folder/image
dataset = ImageFolder(root=root_dir, transform=transform)
train_ratio = 0.8
val_ratio = 0.1
test_ratio = 0.1
num_samples = len(dataset)
train_size = int(train_ratio * num_samples)
val_size = int(val_ratio * num_samples)
test_size = num_samples - train_size - val_size
training_accuracy , testing_accuracy , validation_accuracy = [], [], []
training_loss , testing_loss , validation_loss = [], [], []
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])
batch = 256
train_loader = DataLoader(train_dataset, batch_size=batch, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch, shuffle=False)
```

Build a CNN classifier with atleast 3 convolutional layers to train on an ImageNet dataset that you have collected. You may follow AlexNet CNN architecture to build a CNN model.

I used Alexnet Architecture for my model

```
class AlexNet(nn.Module):
    def __init__(self, output_dim):
        super().__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 3, 2, 1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 192, 3, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 384, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True)
        )

        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, output_dim),
        )

    def forward(self, x):
        x = self.features(x)
        h = x.view(x.shape[0], -1)
        x = self.classifier(h)
        return x, h
```

a) The model contains two main parts: self.features for extracting image features and self.classifier for classification.
b) The self.features part includes several convolutional layers with ReLU activation and max pooling operations.
c) The self.classifier part consists of fully connected layers with dropout and ReLU activation.

d) The model takes RGB images as input and produces output logits based on the specified output_dim.
e) During the forward pass, input images are processed through the feature extraction layers, reshaped, and passed through the classification layers to obtain logits. Intermediate features are also returned.

Regularization: L1 and L2

L2 regularization -> weight_decay = 0.001

```
OUTPUT_DIM = 10
model = AlexNet(OUTPUT_DIM)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.8, weight_decay=0.001)
num_epochs = 100
transform = transforms.Compose([
       transforms.RandomHorizontalFlip(0.2),
                           transforms.RandomCrop(32, padding=2),
                           transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
root_dir = "/content/ImageNet-Datasets-Downloader-master/ImageNet-Datasets-Downloader-master/data_root_folder/imagenet"
dataset = ImageFolder(root=root_dir, transform=transform)
train_ratio = 0.8
val_ratio = 0.1
test_ratio = 0.1
num_samples = len(dataset)
```

Dropout: Introduce dropout layers between the fully connected layers.

```
        )

        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, output_dim),
        )
```

We have 3 dropout layers each with value 0.5. This line defines a nn.Sequential container for the classifier part of the model. It consists of dropout layers, linear layers, and ReLU activation functions sequentially. The last linear layer produces the output logits with a size matching output_dim.

Image Augmentation: I didnt perform any image augmentation in this task

Early stopping: Monitor the performance of the model on a validation set and stop training when the validation loss stops improving.

```
EPOCHS = 100
best_valid_loss = float('inf')
patience = 5
no_improvement_count = 0
model.load_state_dict(torch.load('/content/ImageNet-Datasets-Downloader-master/ImageNet-Datasets-Downloader-master/image.pt'))

for epoch in trange(EPOCHS, desc="Epochs"):
    start_time = time.monotonic()
    train_loss, train_acc = train(model, train_loader, optimizer, criterion)
    valid_loss, valid_acc = evaluate(model, val_loader, criterion,0)
    test_loss, test_acc = evaluate(model, test_loader, criterion,0)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        no_improvement_count = 0
        torch.save(model.state_dict(), '/content/ImageNet-Datasets-Downloader-master/ImageNet-Datasets-Downloader-master/image.pt')
    else:
        no_improvement_count += 1
    if no_improvement_count >= patience:
        print(f'Validation loss has not improved for {patience} epochs. Early stopping.')
        break
    end_time = time.monotonic()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. Acc: {valid_acc*100:.2f}%')
    print(f'\t Test. Loss: {test_loss:.3f} |  Test. Acc: {test_acc*100:.2f}%')
    training_accuracy.append(train_acc)
    testing_accuracy.append(test_acc)
    validation_accuracy.append(valid_acc)
    training_loss.append(train_loss)
    testing_loss.append(test_loss)
    validation_loss.append(valid_loss)
```

The above code shows early stopping with the patience value being 5.

Expected min accuracy: 70% -> Validation loss did not improve after 5 epochs showing us that the early stopping has been enabled.

```
print("F1 Score:", f1)
```

```
Epoch: 01 | Epoch Time: 0m 40s
        Train Loss: 0.171 | Train Acc: 70.07%
         Val. Loss: 1.546 |  Val. Acc: 65.80%
         Test. Loss: 1.541 |  Test. Acc: 65.67%
Epoch: 02 | Epoch Time: 0m 40s
        Train Loss: 0.207 | Train Acc: 64.89%
         Val. Loss: 1.602 |  Val. Acc: 63.64%
         Test. Loss: 1.545 |  Test. Acc: 64.38%
Epoch: 03 | Epoch Time: 0m 40s
        Train Loss: 0.190 | Train Acc: 66.61%
         Val. Loss: 1.604 |  Val. Acc: 65.37%
         Test. Loss: 1.616 |  Test. Acc: 65.24%
Epoch: 04 | Epoch Time: 0m 42s
        Train Loss: 0.201 | Train Acc: 64.50%
         Val. Loss: 1.659 |  Val. Acc: 62.77%
         Test. Loss: 1.654 |  Test. Acc: 62.66%
Epoch: 05 | Epoch Time: 0m 40s
        Train Loss: 0.210 | Train Acc: 64.76%
         Val. Loss: 1.651 |  Val. Acc: 62.34%
         Test. Loss: 1.528 |  Test. Acc: 65.67%
Validation loss has not improved for 5 epochs. Early stopping.
```

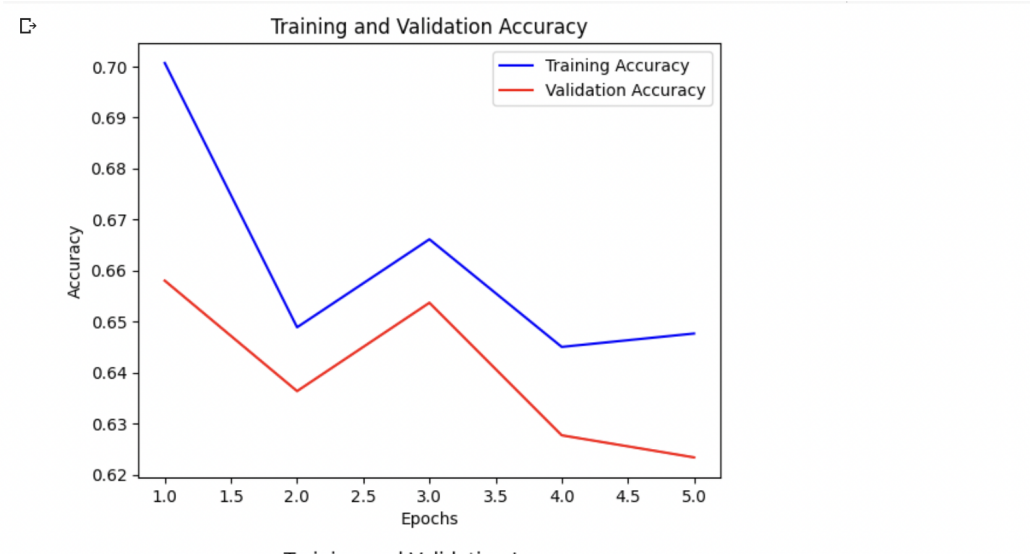
Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.

```
print("F1 Score:", f1)
Epochs: 5%
Epoch: 01 | Epoch Time: 0m 40s
        Train Loss: 0.171 | Train Acc: 70.07%
         Val. Loss: 1.546 |  Val. Acc: 65.80%
         Test. Loss: 1.541 |  Test. Acc: 65.67%
Epoch: 02 | Epoch Time: 0m 40s
        Train Loss: 0.207 | Train Acc: 64.89%
         Val. Loss: 1.602 |  Val. Acc: 63.64%
         Test. Loss: 1.545 |  Test. Acc: 64.38%
Epoch: 03 | Epoch Time: 0m 40s
        Train Loss: 0.190 | Train Acc: 66.61%
         Val. Loss: 1.604 |  Val. Acc: 65.37%
         Test. Loss: 1.616 |  Test. Acc: 65.24%
Epoch: 04 | Epoch Time: 0m 42s
        Train Loss: 0.201 | Train Acc: 64.50%
         Val. Loss: 1.659 |  Val. Acc: 62.77%
         Test. Loss: 1.654 |  Test. Acc: 62.66%
Epoch: 05 | Epoch Time: 0m 40s
        Train Loss: 0.210 | Train Acc: 64.76%
         Val. Loss: 1.651 |  Val. Acc: 62.34%
         Test. Loss: 1.528 |  Test. Acc: 65.67%
Validation loss has not improved for 5 epochs. Early stopping.
```

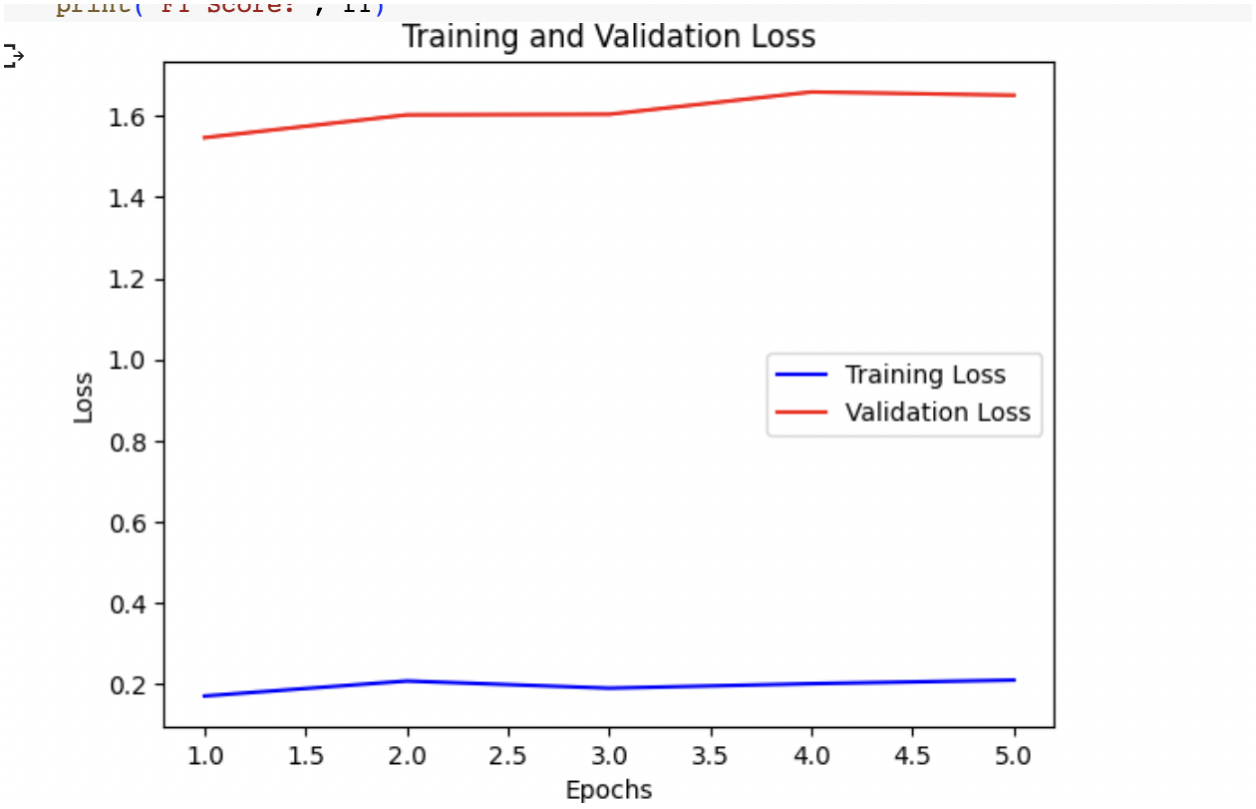
Training and Validation Accuracy

Plot the training and validation accuracy over time (epochs).

The training and validation accuracy lie between 0.62-0.70. The maximum value of training accuracy is 0.70 which is the minimum expected accuracy

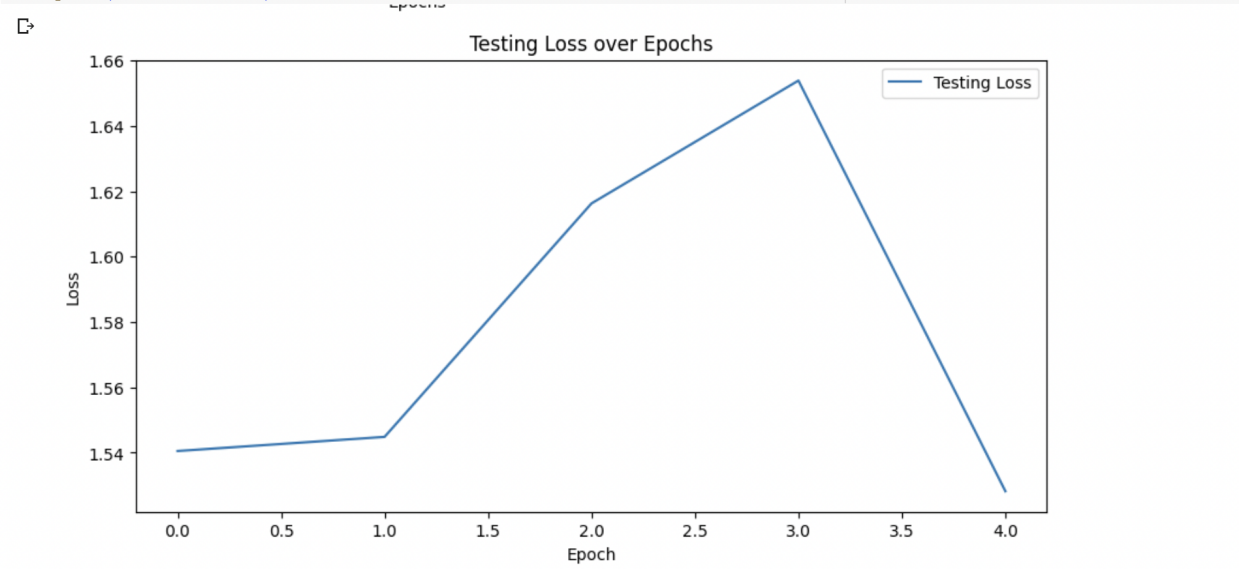


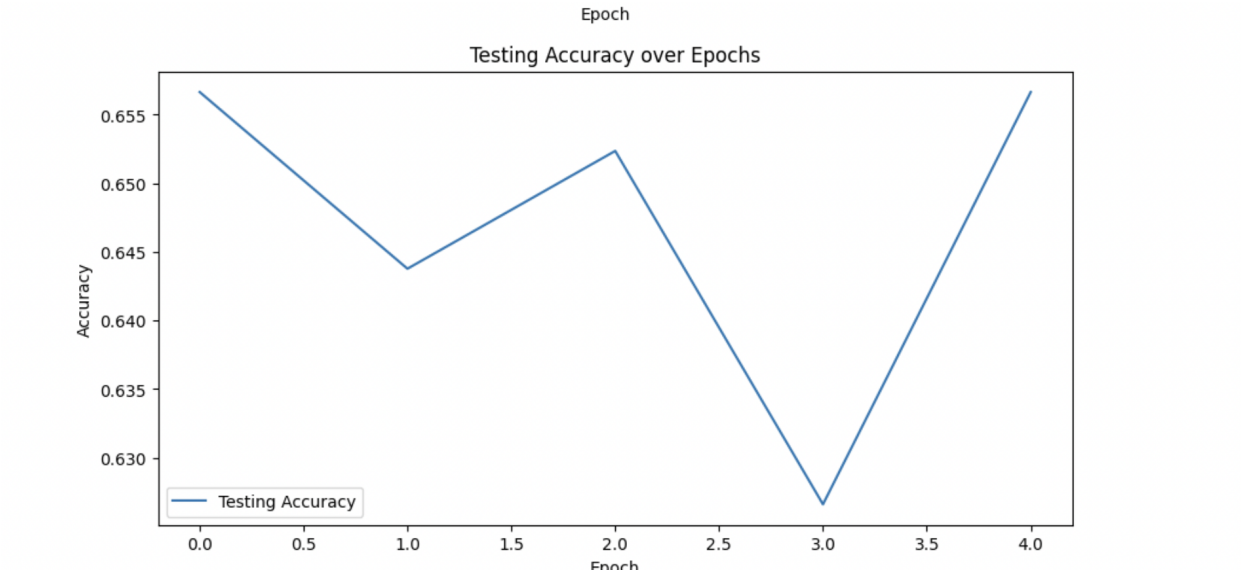
Plot the training and validation loss over time (epochs).



Plot the testing accuracy and testing loss over time (epochs).

The testing loss can be seen lying between 1.54 and 1.66

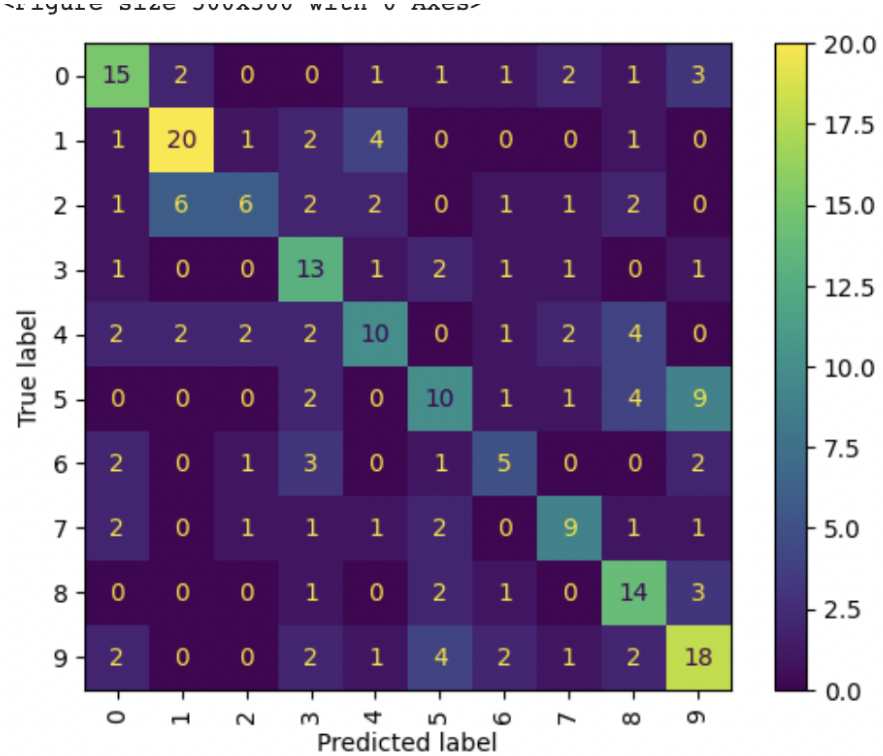




The testing accuracy is lying between 0.630 and 0.655.

Generate a confusion matrix using the model's predictions on the test set.

We have a 10x10 matrix - 10 classes.



Calculate and report other evaluation metrics such as precision, recall, and F1 score to further analyze the model's performance on the test set.

All the precision,recall, and f1 score have values 0.50 indicating the model has performed significantly well.

```
Precision: 0.5091147714372867
Recall: 0.505897242923105
F1 Score: 0.4990558770126093
<Figure size 500x500 with 0 Axes>
```