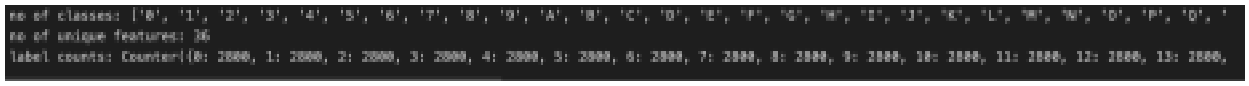
EMNIST_MNIST_CLASSIFICATION

1. Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? How many entries and variables does the dataset comprise? Provide the main statistics about the entries of the dataset.

EMNIST data is the extended version of the MNIST data mostly it contains the handwritten digits from (0 -9) and the Upper case letter from (A-Z) and the lower case letters from (a-z) and entire dataset it is int format of grayscale images which has the image shape of 28 * 28 and 3 channel dimension.

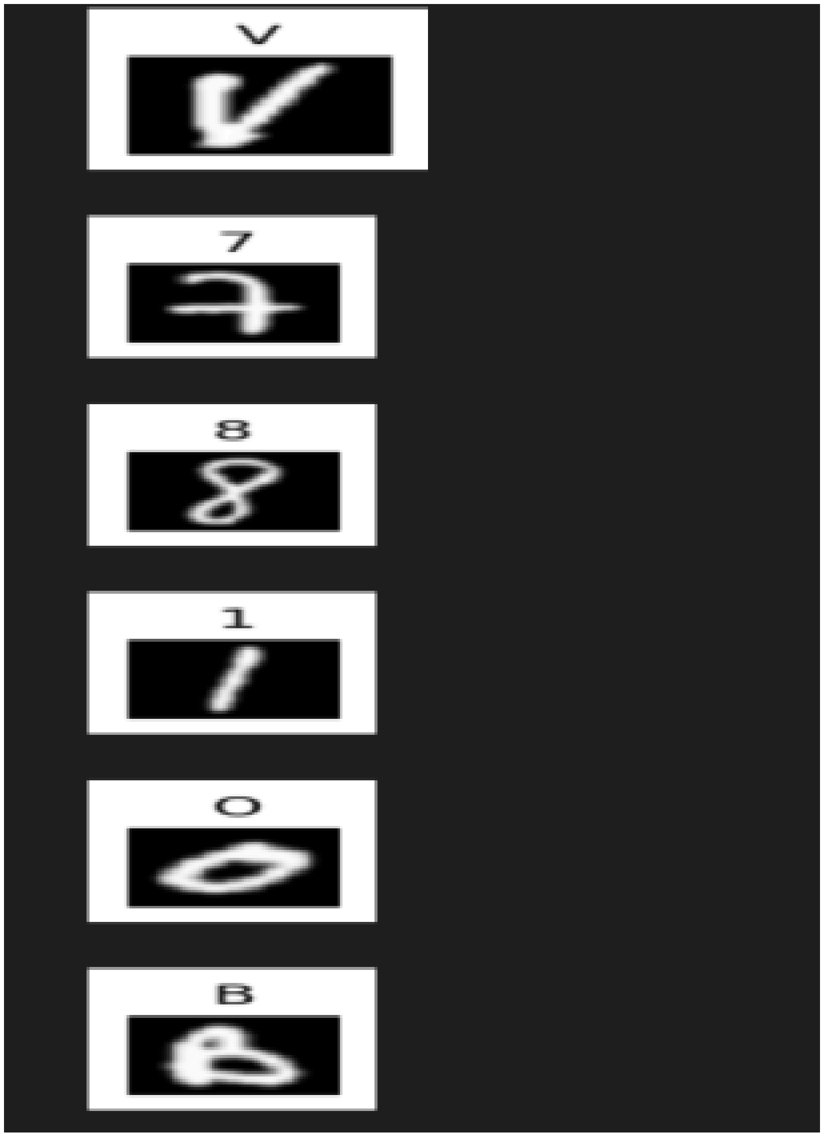


```
no of classes: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', '
no of unique features: 36
label counts: Counter({0: 2800, 1: 2800, 2: 2800, 3: 2800, 4: 2800, 5: 2800, 6: 2800, 7: 2800, 8: 2800, 9: 2800, 10: 2800, 11: 2800, 12: 2800, 13: 2800,
```

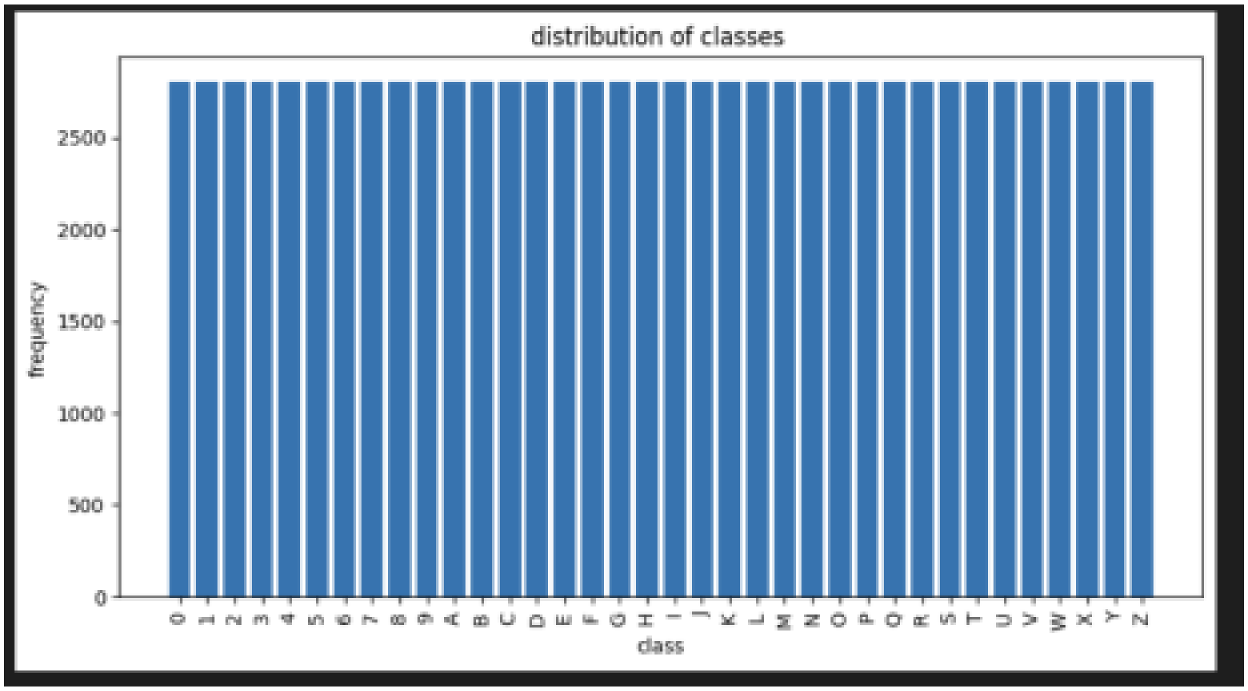
From this we have we have 36 unique classes and has total length of the dataset 100800 samples of the data

Data Visualization:

In this we will know how the images are in the dataset

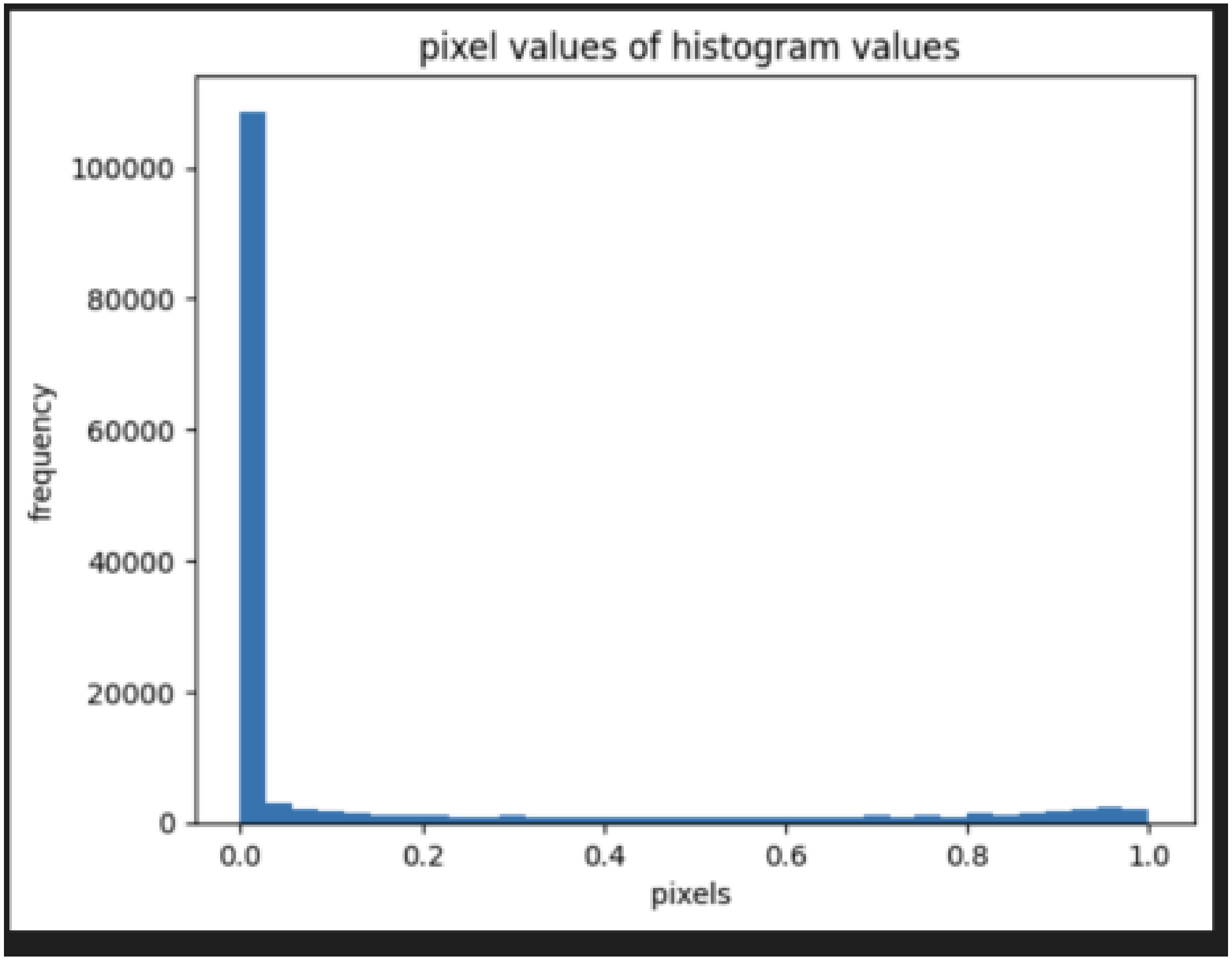


Representation of how the classes are distributed

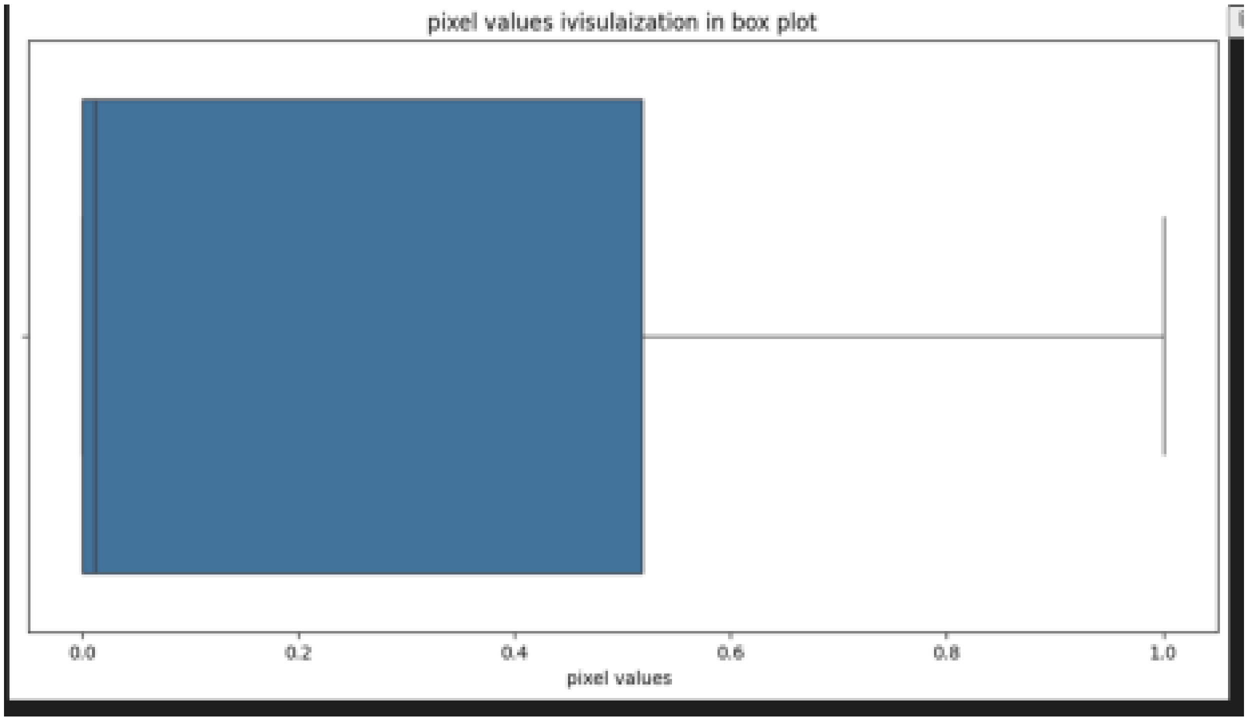


2. Provide at least 3 visualization graphs with short description for each graph.

Below image is a representation of randomly selected image pixel representation from the dataset



Now below image is an similar kind of image where the pixel values are represented in the box plot visulaization



Data augmentation

From the data set We have selected a random image and we have applied data augmentation methods and below is the representation of how the images are flipped horizontally and random rotation by 10 here is the visualization.

data augmentation methods

Provide the model details of your CNN.

** In this building the architecture I have built the Custom CNN architecture where I have used the 5 convolution layers and the Max pool , and the activation function RELU. Before that we are resize the entire image dataset to the shape of 224 * 224 * 3

Layer 1 : In my CNN architecture I have built the first convolution layer with the input channel 3 and the output channel 16 , stride 1 padding 1.
and applying the activation function after this i am applying the max pool to the first convolution layer in this to reduce the spatial dimension by a factor of 2 after applying this my I wil get the image shape to the 122 * 122 * 16
Layer 2 : I have built the second convolution layer with the input channel 16 and the output channel 32 , stride 1 padding 1 . and applying the activation function and then After this i am applying the max pool to the first convolution layer in this to reduce the spatial dimension by a factor of 2 after applying this my I wil get the image shape to the 61 * 61 * 32

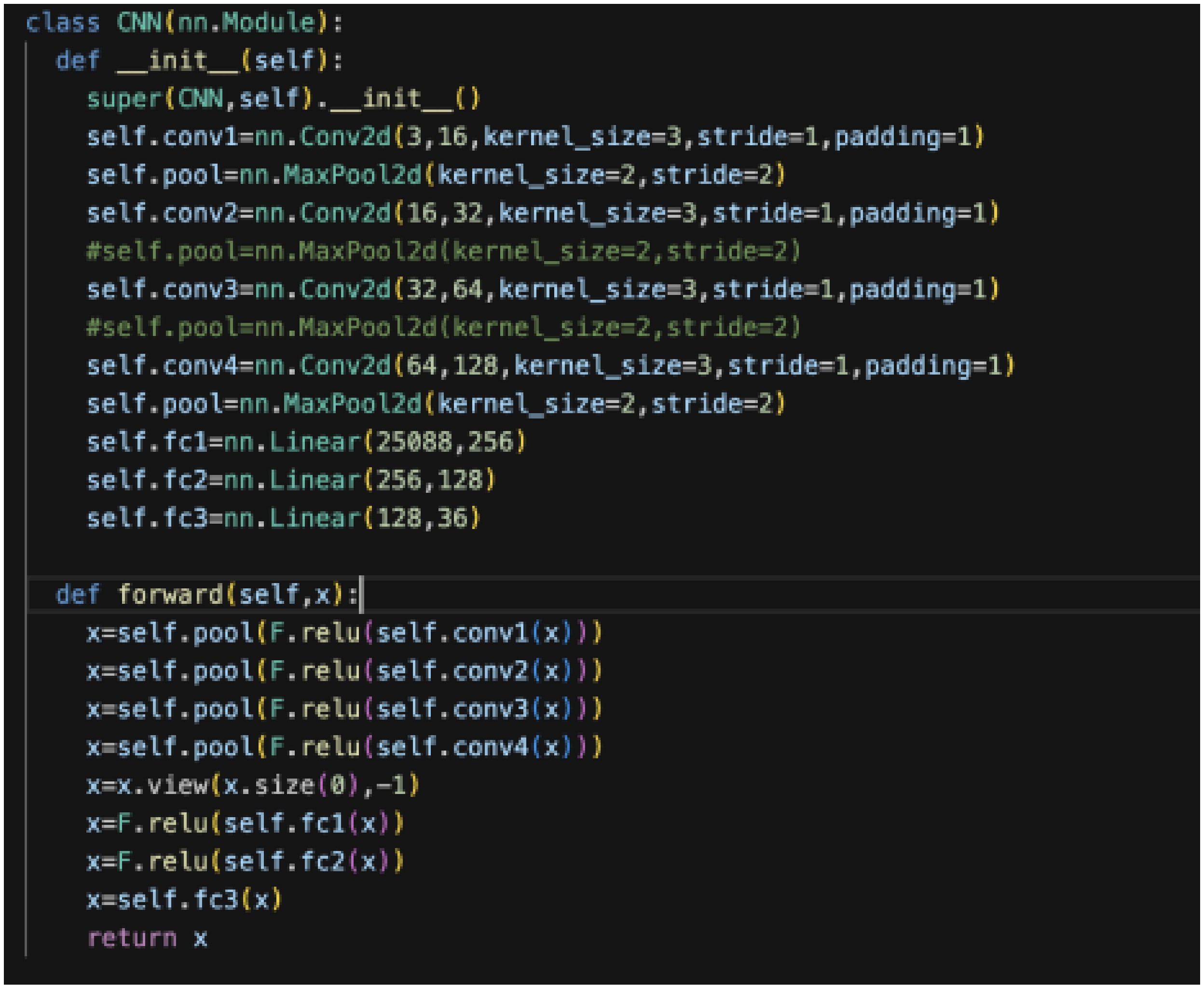
Layer 3 : I have built the third convolution layer with the input channel 32 and the output channel 64 , stride 1 padding 1 . and applying the activation function and then After this i am applying the max pool to the first convolution layer in this to reduce the spatial dimension by a factor of 2 after applying this my I wil get the image shape to the 30.5 * 30.5 * 64

Layer 4 : I have built the third convolution layer with the input channel 64 and the output channel 128 , stride 1 padding 1 . and applying the activation function and then After this i am applying the max pool to the first convolution layer in this to reduce the spatial dimension by a factor of 2 after applying this my I wil get the image shape to the 14* 14 * 128

In the architecture after that I have built the Fully connected layer for the model.The first fully connected layer 1 takes input of size 25088 and outputs 256 features. The input size is calculated based on the output size of the convolutional layers after flattening. Fully connected layer 2 takes the input size of 256 features as an input from the fully connected layer 1 and the outputs are the 128
Fully connected layer 3 takes the input features of 128 from the second fully connected layer and outputs 36 features.

After everything the model is flatten the output tensor into a 2D tensor with shape .

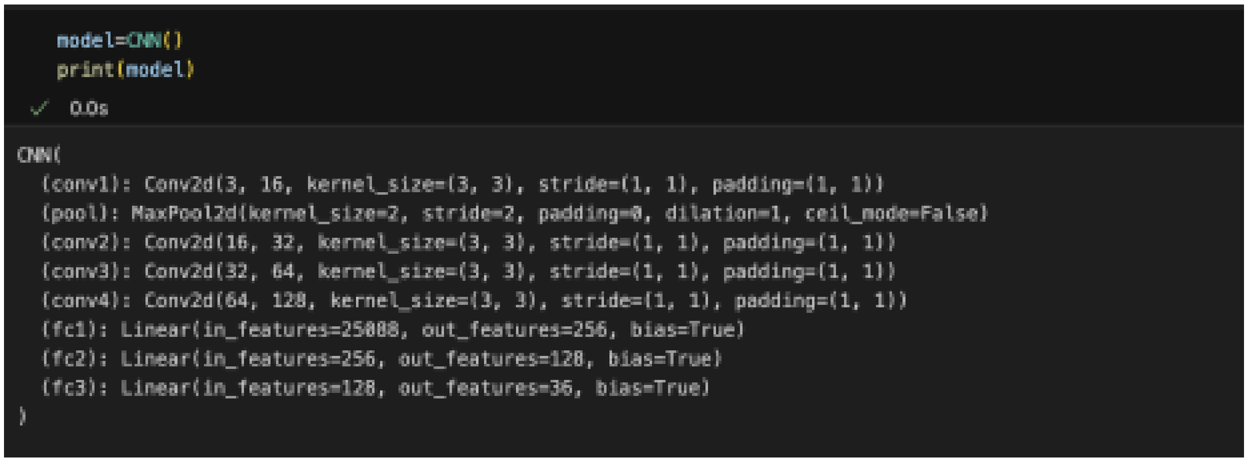
The code architecture would be like this



```python
class CNN(nn.Module):
  def __init__(self):
    super(CNN,self).__init__()
    self.conv1=nn.Conv2d(3,16,kernel_size=3,stride=1,padding=1)
    self.pool=nn.MaxPool2d(kernel_size=2,stride=2)
    self.conv2=nn.Conv2d(16,32,kernel_size=3,stride=1,padding=1)
    #self.pool=nn.MaxPool2d(kernel_size=2,stride=2)
    self.conv3=nn.Conv2d(32,64,kernel_size=3,stride=1,padding=1)
    #self.pool=nn.MaxPool2d(kernel_size=2,stride=2)
    self.conv4=nn.Conv2d(64,128,kernel_size=3,stride=1,padding=1)
    self.pool=nn.MaxPool2d(kernel_size=2,stride=2)
    self.fc1=nn.Linear(25088,256)
    self.fc2=nn.Linear(256,128)
    self.fc3=nn.Linear(128,36)

  def forward(self,x):
    x=self.pool(F.relu(self.conv1(x)))
    x=self.pool(F.relu(self.conv2(x)))
    x=self.pool(F.relu(self.conv3(x)))
    x=self.pool(F.relu(self.conv4(x)))
    x=x.view(x.size(0),-1)
    x=F.relu(self.fc1(x))
    x=F.relu(self.fc2(x))
    x=self.fc3(x)
    return x
```

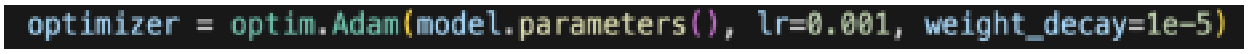
The model architecture would be like this

.



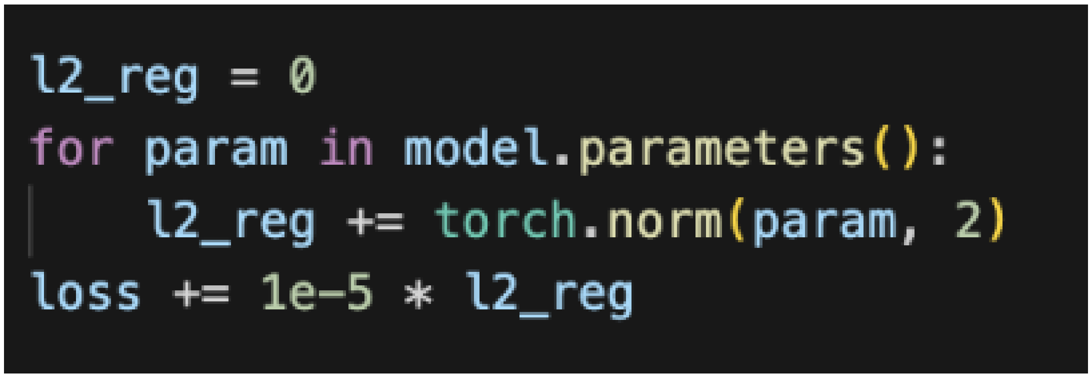
```
model=CNN()
print(model)
✓ 0.0s

CNN(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=25088, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=36, bias=True)
)
```

Discuss how the improvement tools work on CNN architectures.
L2 Regularization :



```python
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```

I used weight decay with a decay value of 1e-5 in the Adam optimizer. By punishing large weights during optimization and managing the model's complexity, this assisted in preventing overfitting. It additionally enhanced the extrapolation of unknown data.

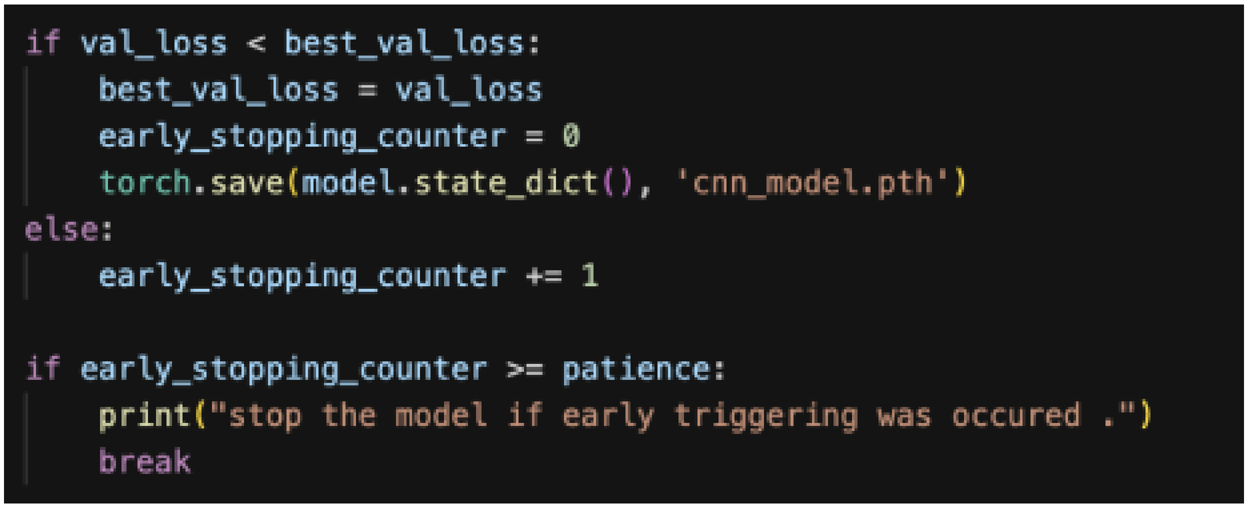


```
l2_reg = 0
for param in model.parameters():
    l2_reg += torch.norm(param, 2)
loss += 1e-5 * l2_reg
```

adds the regularization penalty to the original loss, this penalizing large weights, and helps prevent overfitting by encouraging the model to keep the weights small.

Early Stopping :

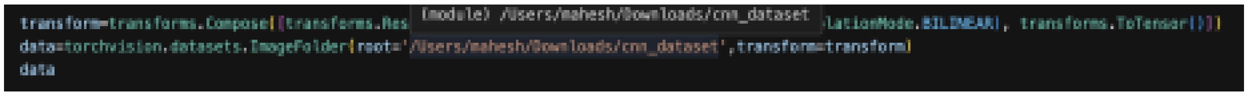
In this code i have implemented the early stopping in the training code itself the main use of this early stopping is if the loss validation or validation loss has no improvement for consecutive 5 iterations and automatically training has to stop to prevent from the overfitting issues and this help us to avoid the overfitting and it will give us the best model and it will ensures the performance of the model also will be good and give better results



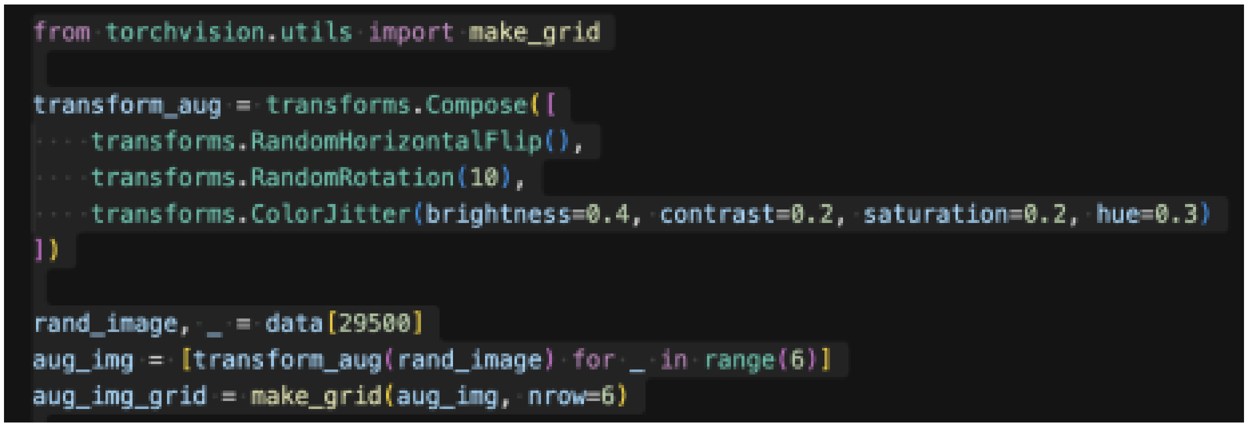
```
if val_loss < best_val_loss:
    best_val_loss = val_loss
    early_stopping_counter = 0
    torch.save(model.state_dict(), 'cnn_model.pth')
else:
    early_stopping_counter += 1

if early_stopping_counter >= patience:
    print("stop the model if early triggering was occured .")
    break
```

Image Augmentation :

The input images are converting or resizing to the shape of 224 * 224 . and the images are passed to random rotations up to 10 degrees in both clockwise and counterclockwise directions, as well as random horizontal flips. Additionally, we use color jittering to alter hue = 0.3,saturation =0.2 brightness of 0.4 and contrast 0.2.



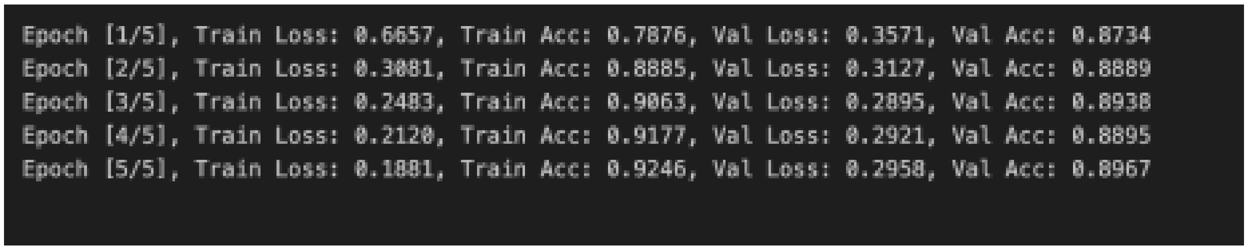
```
transform=transforms.Compose([transforms.Res (module) /Users/mahesh/Downloads/cnn_dataset lationMode.BILINEAR), transforms.ToTensor()])
data=torchvision.datasets.ImageFolder(root='/Users/mahesh/Downloads/cnn_dataset',transform=transform)
data
```



```
from torchvision.utils import make_grid

transform_aug = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.4, contrast=0.2, saturation=0.2, hue=0.3)
])

rand_image, _ = data[29500]
aug_img = [transform_aug(rand_image) for _ in range(6)]
aug_img_grid = make_grid(aug_img, nrow=6)
```

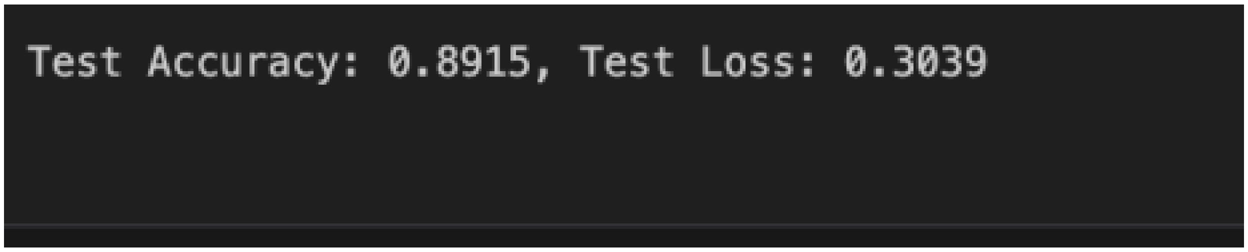
.Provide the performance metrics and analyze the results.

Training accuracy , loss and Validation accuracy and loss



```
Epoch [1/5], Train Loss: 0.6657, Train Acc: 0.7876, Val Loss: 0.3571, Val Acc: 0.8734
Epoch [2/5], Train Loss: 0.3081, Train Acc: 0.8885, Val Loss: 0.3127, Val Acc: 0.8889
Epoch [3/5], Train Loss: 0.2483, Train Acc: 0.9063, Val Loss: 0.2895, Val Acc: 0.8938
Epoch [4/5], Train Loss: 0.2120, Train Acc: 0.9177, Val Loss: 0.2921, Val Acc: 0.8895
Epoch [5/5], Train Loss: 0.1881, Train Acc: 0.9246, Val Loss: 0.2958, Val Acc: 0.8967
```

Testing accuracy and test loss



```
Test Accuracy: 0.8915, Test Loss: 0.3039
```

When compared to the accuracy on the EMNIST data after the training process I have got over 92 % accuracy this indicates that model has learned the training data in good manner and after that we have got the validation accuracy over 89 % and this accuracy is very close to the training and and the model performs in a generalized manner and it avoids the problem of the overfitting.while coming to the test accuracy this is near to the validation accuracy and it indicates that the model performs very well on the data .

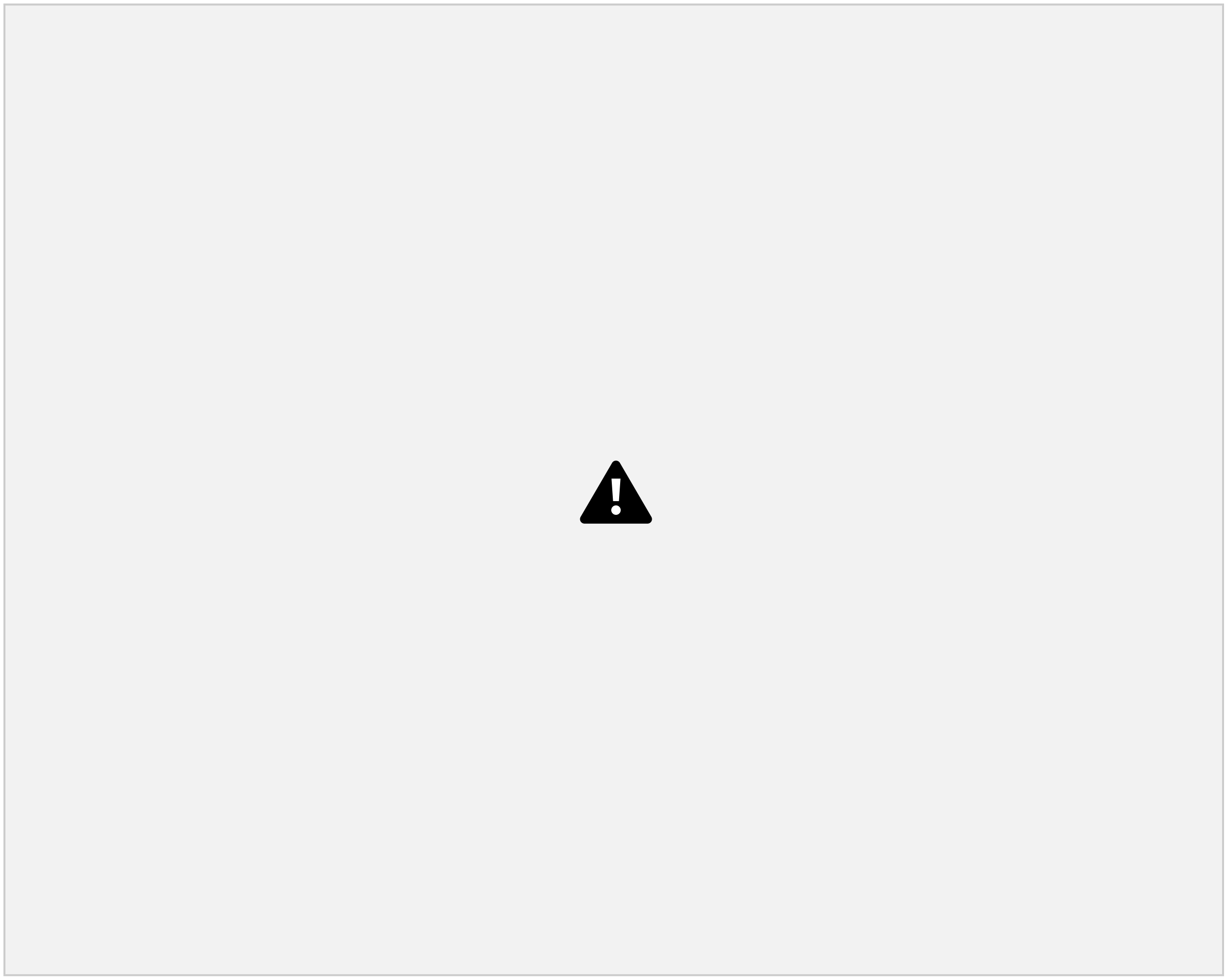
Confusion matrix

The diagonal elements represents the number of points for which the predicted label equals the true label, while off-diagonal elements are those that mislabeled by classifier. The higher diagonal values of confusion matrix the better, indicating correct predictions many.

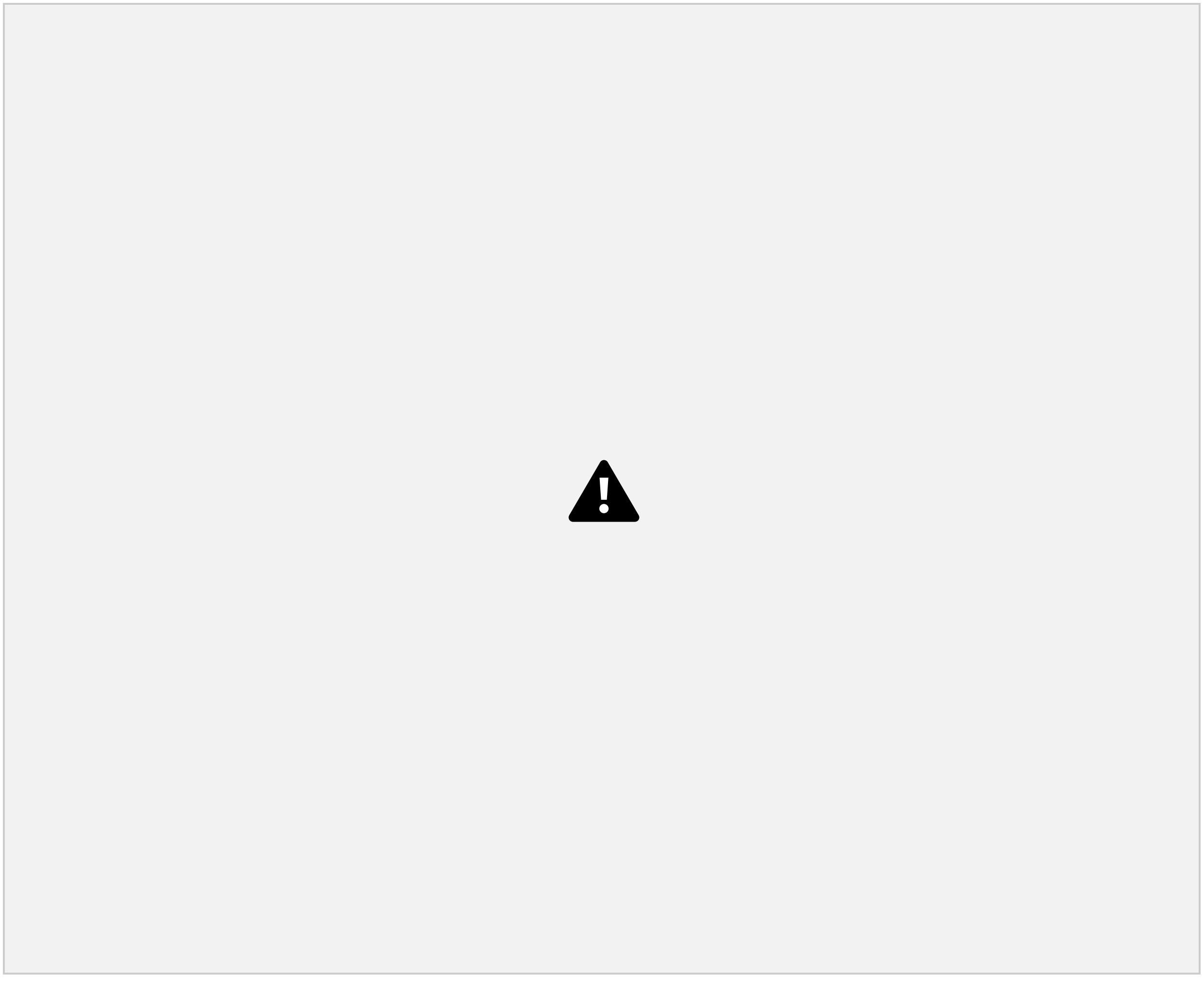
F1 score , precision , recall:

Provide graphs and analyze the results.

Training accuracy vs validation accuracy



Training loss vs validation Loss

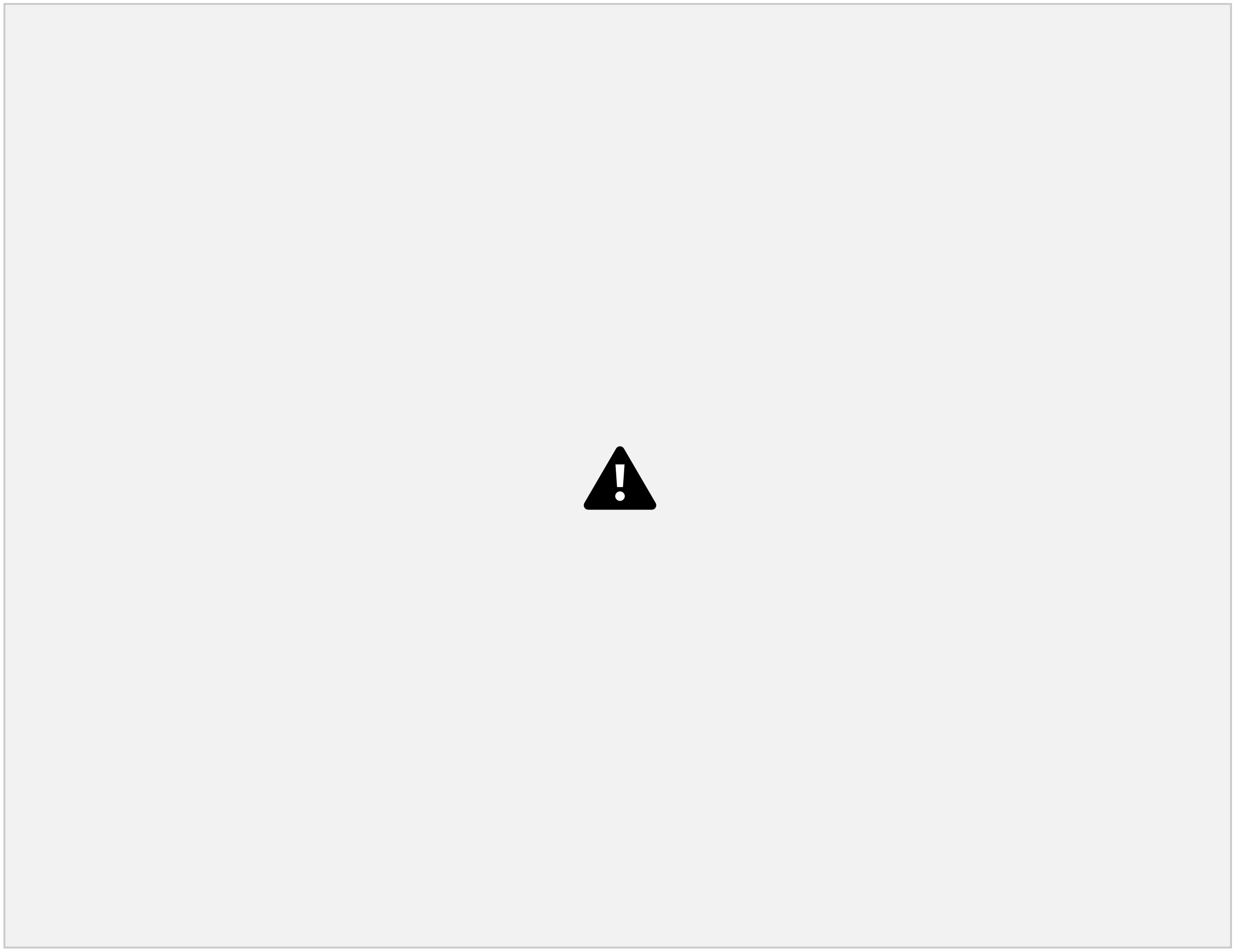


ROC Curve

X-axis represents the false positive rate and y - axis represents the true positive rates and these lie in between 0 - 1. Orange line in the graph will represent the ROC curve. And the curve will start from the origin (0,0) and it will increase towards the left corner . The shape of the graph will indicate that the classifier performs well and it correctly classifies a high amount of the positive samples while keeping false positive rate low.Diagonal purple dashed line represents a random classifier and the good classifier stays as far away from this line as possible. An AUC of 0.94 suggests that the classifier is very good at distinguishing between positive and negative classes.

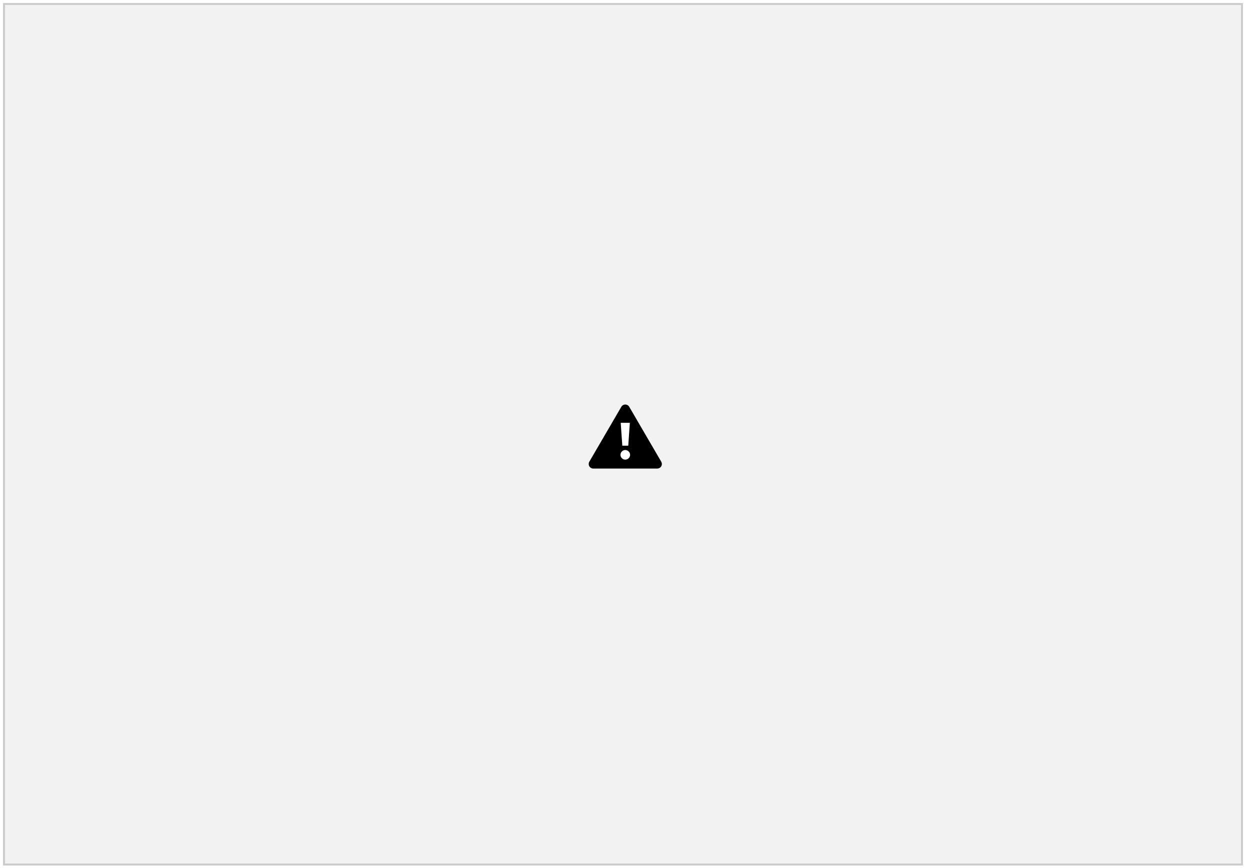
Box plot representation Of Training accuracy , validation accuracy , testing accuracy:

Down below is the box plot representation of the accuracy of the model and we can compare each of the results



Box plot representation of training loss , validation loss , testing loss

In this graph down below is the representation of the losses of train , validation , test loss of the model



DEEP LEARNING

ASSIGNMENT1_part_2_report

CNN Model:

- The CNN model defined in the code is the VGG13 (Version B) architecture. ● First, there are max pooling layers, following a sequence of convolutional layers with ReLU activation.
- With a kernel size of 3x3 and a padding of 1, the convolutional layers contain 64, 128, 256, and 512 filters.
- An adaptive average pooling layer comes after the convolutional layers, and then fully connected layers.
- ReLU activation and dropout regularization are features of each of the 4096 units in the fully linked layers.
- The final output layer has 3 units, corresponding to the number of classes.

Impact of Regularization, Dropout, and Early Stopping:

- **Regularization:** Training involves applying L2 regularization to the model parameters. For the purpose of calculating the loss function, the algorithm multiplies the weights' L2 norm by a regularization strength of 1e-5.
- **Dropout:** Dropout regularization is used in the fully connected layers, as evident from the nn.Dropout() layers in the classifier part of the model.
- **Early stopping:** The validation loss is monitored in order to apply early stopping. Training ends after a certain amount of epochs (patience=5) if the validation loss does not improve.
- **learning rate:** The learning rate is adjusted using the `ReduceLROnPlateau` scheduler based on the validation loss.

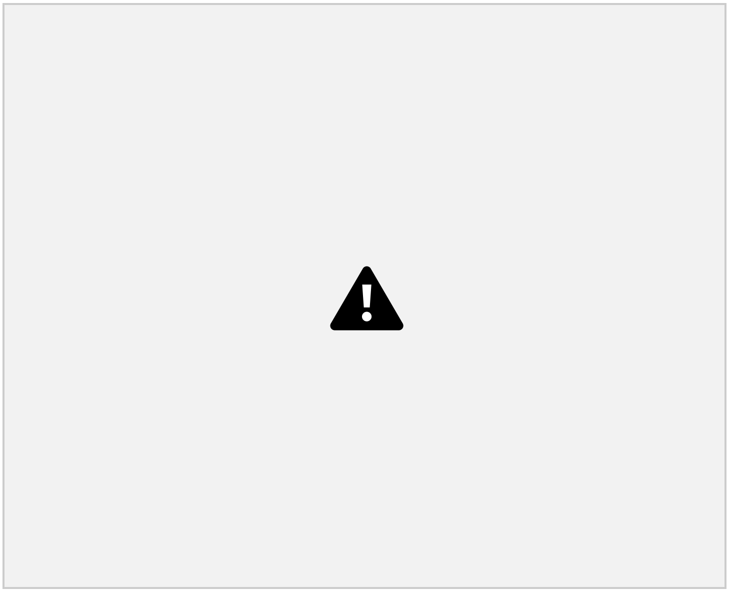
Results and Graphs:

a. Training Accuracy, Training Loss, Validation Accuracy, Validation Loss, Testing Accuracy, and Testing Loss:

- Over the course of five epochs, the CNN model's training and validation accuracy increased considerably, reaching 90.94% and 89.52%, respectively. Testing accuracy on unseen data was high at 90.98%, confirming the model's effectiveness in image classification

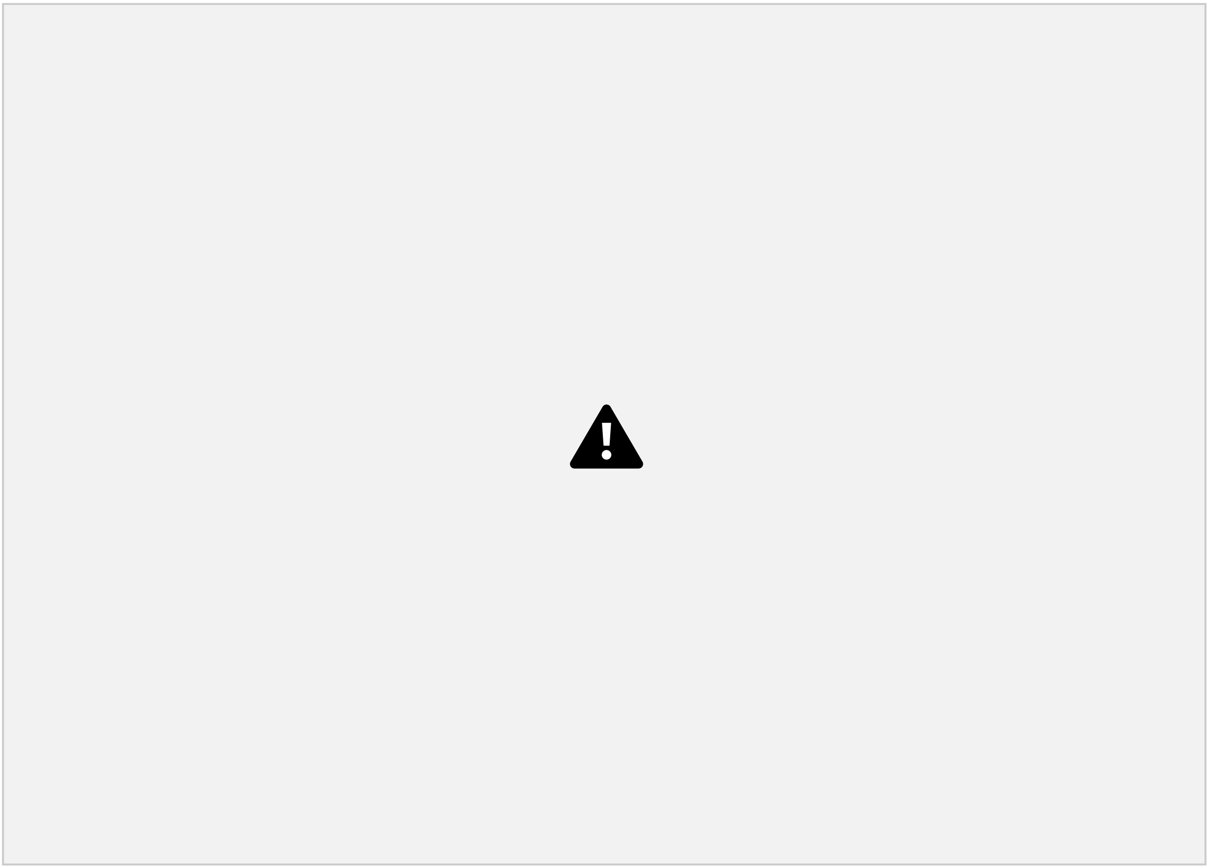
b. Training and Validation Accuracy over Epochs:

- The training and validation accuracy graphs are plotted using `plt.plot()` and `plt.show()`.
- The graph shows the progression of training and validation accuracy over the epochs.
- It helps to visualize the model's performance and identify any overfitting or underfitting.

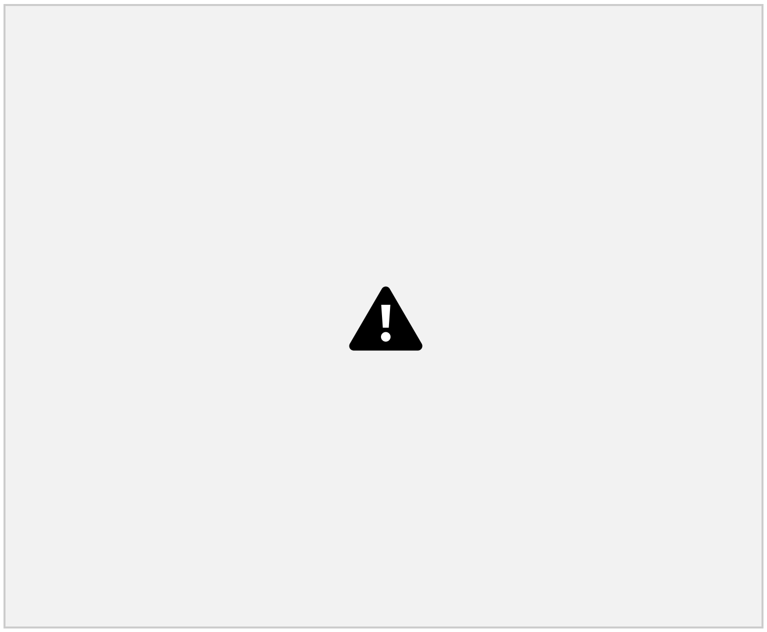


Training and Validation Loss over Epochs:

- Similar to the accuracy graph, the training and validation loss graphs are plotted using `plt.plot()` and `plt.show()`.
- The graph displays the change in training and validation loss over the epochs. ● It provides insights into the model's learning process and convergence.



Confusion Matrix:



The confusion matrix offers information about how well each class is performing with the model.

Analyzing the confusion matrix might be useful in order to figure out which particular classes the model finds difficult. If certain classes have high amounts of misclassification, this could mean that the model needs to be improved in order to better identify those classes.

The true labels and predicted labels from the test set are used to compute the confusion matrix.

Other Evaluation Metrics:

- The code calculates precision, recall, and F1 score using the `precision_score()`, `recall_score()`, and `f1_score()` functions from scikit-learn.
- The true labels and predicted labels from the test set are used to compute these metrics.
- The output shows a precision of 0.9104, recall of 0.9098, and F1 score of 0.9094.

In conclusion, the CNN model that we developed with the VGG13 architecture is performing incredibly well when it comes to picture classification. Our model achieves excellent accuracy and minimizes loss by stopping early based on validation loss, dropout and L2 regularization, among other strategies.