Report

Methodology :

Results :

	Model	Number of Conv Layers	Batch Normalization	Weight Decay	Learning Rate Scheduler	Learning Rate	optimizer	Loss Function	Accuracy	precision	recall	f1-score	Nuumber of Epochs
0	First Model	3	No	No	No	0.001	Adam	CrossEntropyLoss	0.79	0.82	0.83	0.83	20
1	Second Model	4	Yes	Yes	Yes	0.001	Adam	CrossEntropyLoss	0.86	0.88	0.90	0.89	50
2	Third Model	4	Yes	Yes	Yes	0.001	Adam	CrossEntropyLoss	0.87	0.91	0.92	0.91	100

Preprocessing :

- **normalized** the data using `transforms.Normalize` **by 0.5**
- **scaled** using `transforms.ToTensor()` scales the pixel values from the range [0, 255] to [0, 1].
- **Agumention** using **Random Affine Transformation,Random Horizontal Flip,Random Affine Transformation**

```
[17] transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomAffine(degrees=0, translate=(0.05, 0.05)),
        transforms.RandomAffine(degrees=0, scale=(0.95, 1.05)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])

    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])
    dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
```

Loading and splitting the data by 0.2 validation split with Applying The preprocessing steps

```
validation_split = 0.2
dataset_size = len(dataset)
split = int(validation_split * dataset_size)
train_size = dataset_size - split
```

```
[ ] train_data, valid_data = torch.utils.data.random_split(dataset, [train_size, split])
    train_loader = torch.utils.data.DataLoader(train_data, batch_size=128, shuffle=True, pin_memory=True)
    valid_loader = torch.utils.data.DataLoader(valid_data, batch_size=128, shuffle=False, pin_memory=True)

    test_data = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
    test_loader = torch.utils.data.DataLoader(test_data, batch_size=128, shuffle=True, pin_memory=True)
```

```
print(dataset)
```

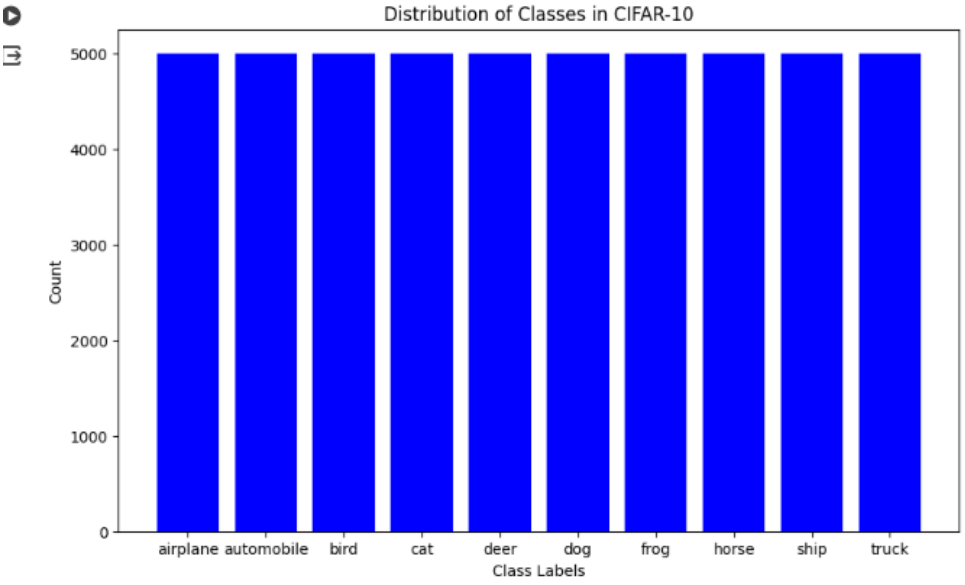
```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: Compose(
               RandomHorizontalFlip(p=0.5)
               RandomAffine(degrees=[0.0, 0.0], translate=(0.05, 0.05))
               RandomAffine(degrees=[0.0, 0.0], scale=(0.95, 1.05))
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

```
print(test_data)
```

```
Dataset CIFAR10
    Number of datapoints: 10000
    Root location: ./data
    Split: Test
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

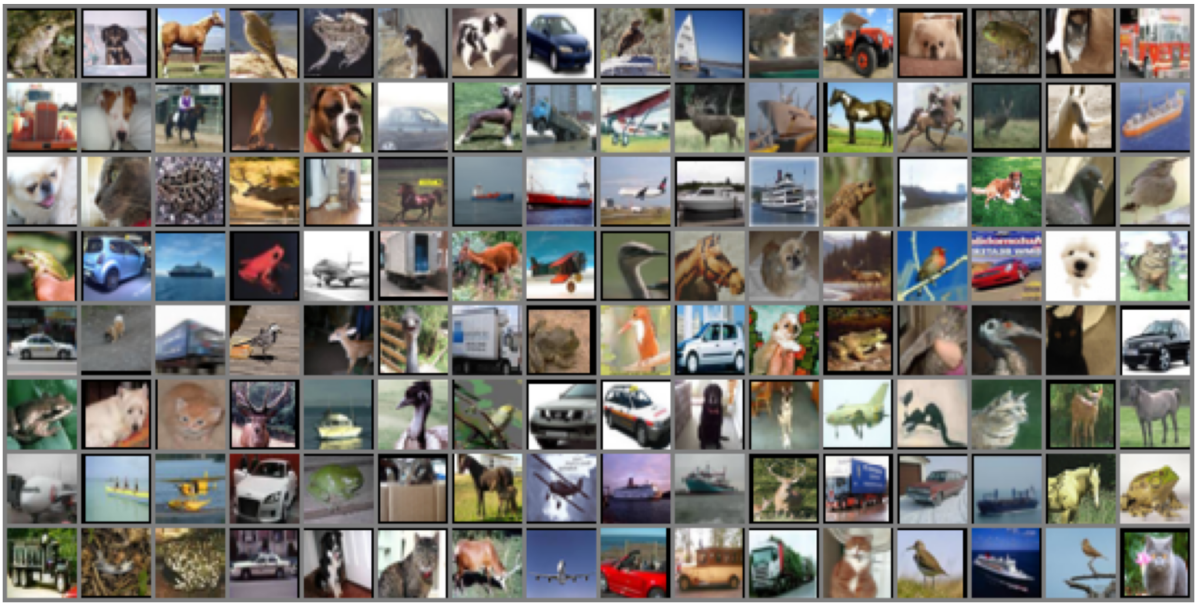
Classes of the dataset

- we have 10 balanced classes

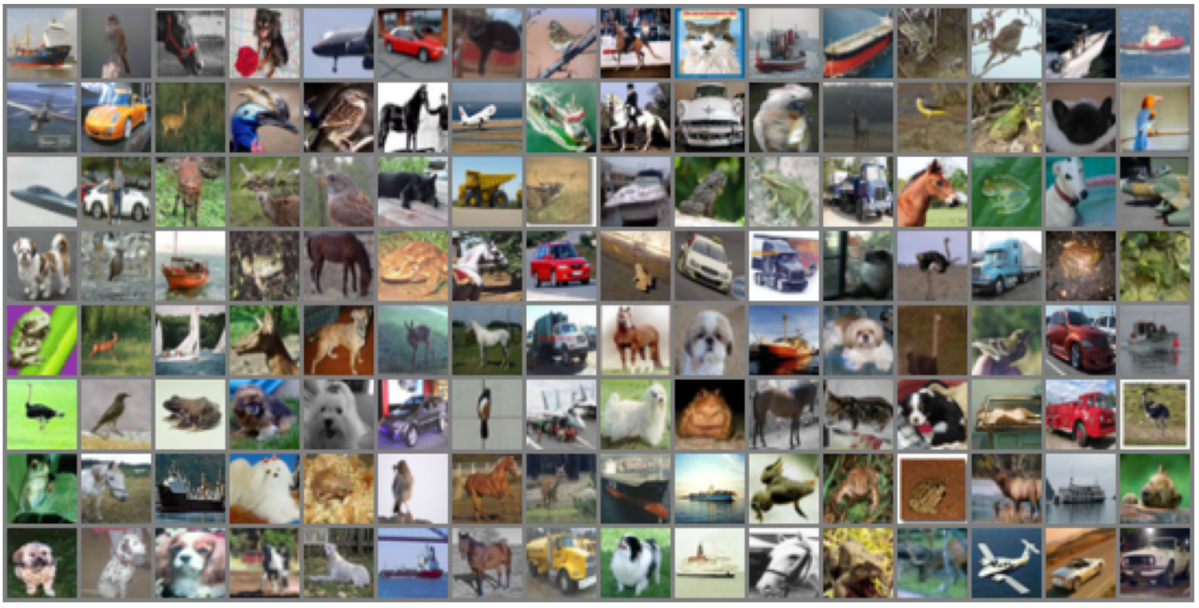


Visualization:

images of the train dataset

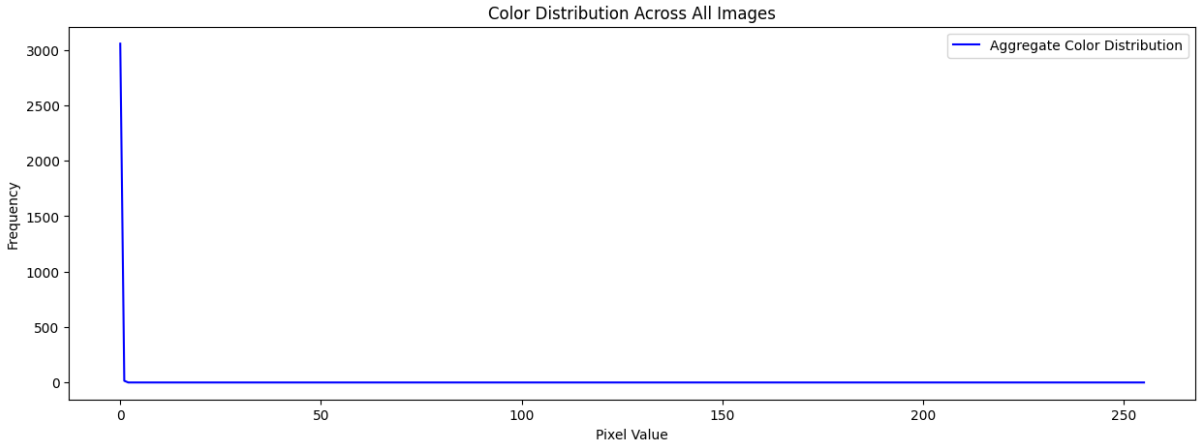


images of the test dataset :



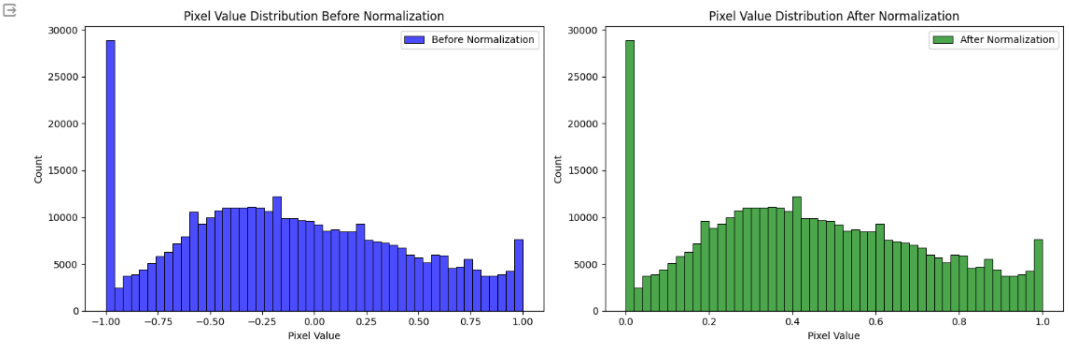
Color distribution across all images in a dataset Graph :

It helps to understand the prevalent colors and intensities present in the images as I was trying to know more about the images in the dataset



Pixel Value Distribution before and after normalization :

Comparing distributions show the impact of normalization on the data range and to be sure that the normalization is applied Correctly



Training and Evaluating and testing Function

```
def train_and_evaluate(model, train_loader, valid_loader, tes
    train_losses = []
    test_losses = []
    valid_accuracies = []
    all_labels = []
    all_predictions = []
```

```
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0

        for inputs, labels in train_loader:
            if train_on_gpu:
                inputs, labels = inputs.cuda(), labels.cuda()

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_train_loss = running_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        model.eval()
        correct = 0
        total = 0

        with torch.no_grad():
            for inputs, labels in valid_loader:
                if train_on_gpu:
                    inputs, labels = inputs.cuda(), labels.cu

                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
                all_labels.extend(labels.cpu().numpy())
                all_predictions.extend(predicted.cpu().numpy(

            accuracy = correct / total
            valid_accuracies.append(accuracy)
```

```
            print(f'Epoch {epoch + 1}/{epochs}, Training Loss

        model.eval()
        test_loss = 0.0

        with torch.no_grad():
            for inputs, labels in test_loader:
                if train_on_gpu:
                    inputs, labels = inputs.cuda(), labels.cu

                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

        avg_test_loss = test_loss / len(test_loader)
        test_losses.append(avg_test_loss)

        print(f'Epoch {epoch + 1}/{epochs}, Training Loss: {a

    return train_losses, test_losses, valid_accuracies,all_pr
```

First Model :

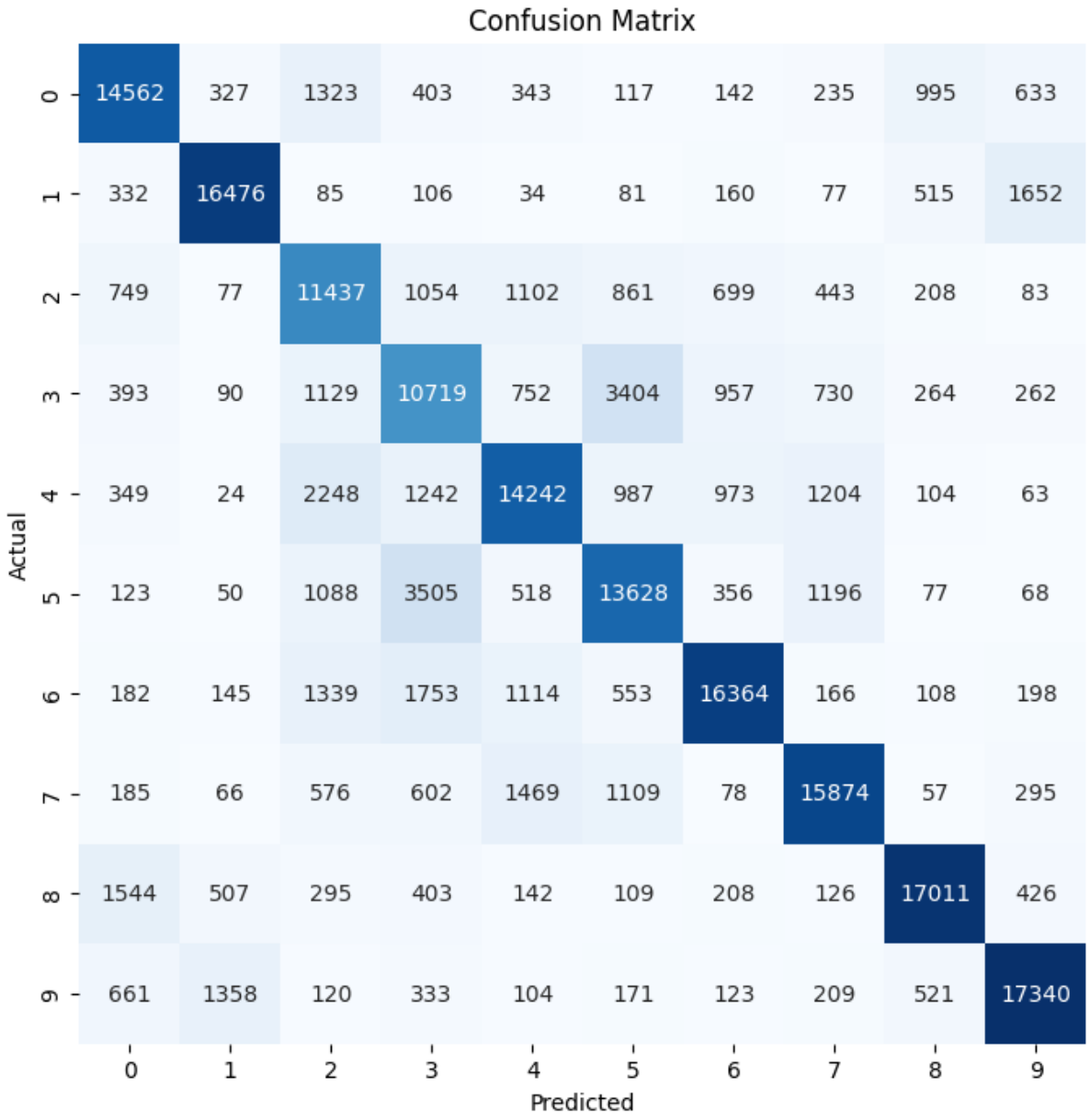
```
class ImprovedNet(nn.Module):
    def __init__(self):
        super(ImprovedNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(256 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)
        self.dropout = nn.Dropout(0.5)

    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 = x.view(-1, 256 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x

improved_model = ImprovedNet()
if train_on_gpu:
    improved_model.cuda()
```

```
[ ]  criterion = nn.CrossEntropyLoss()
     optimizer = optim.Adam(improved_model.parameters(), lr=0.001)
```

Confusion Matrix and Classification Report :

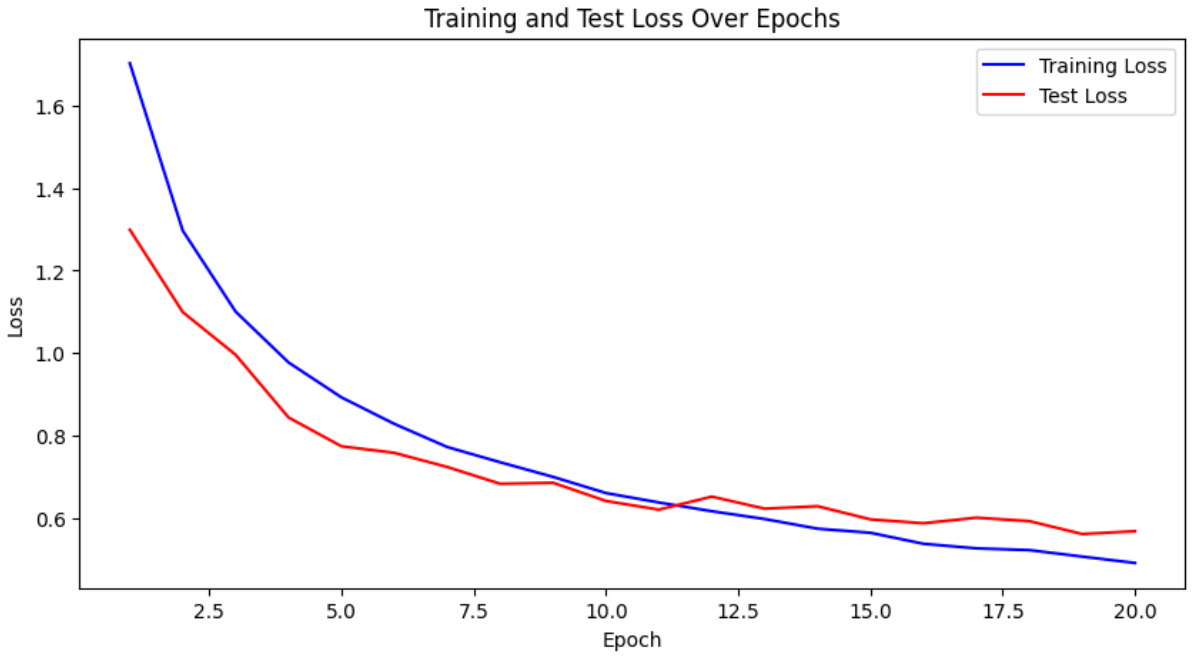


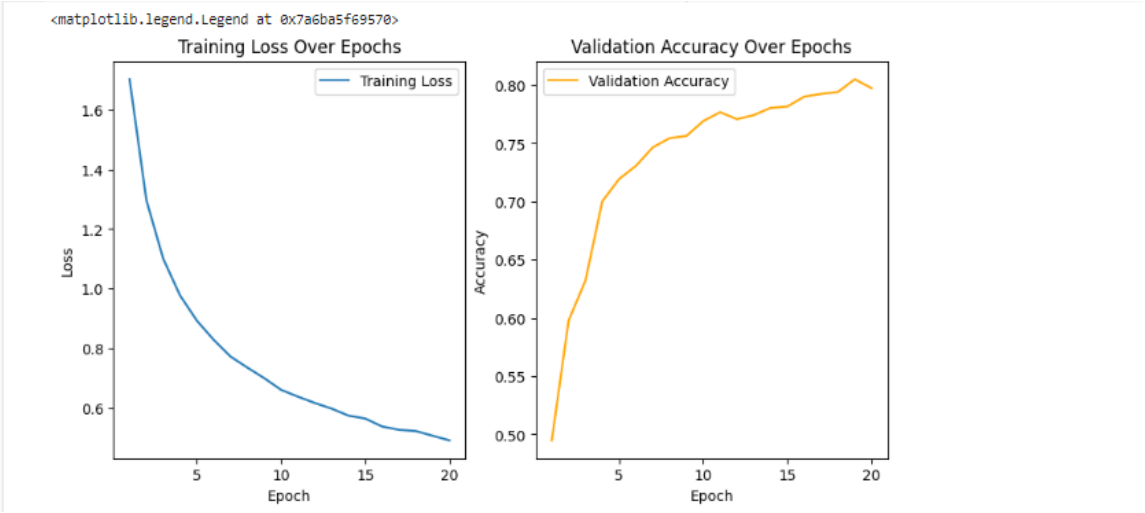
```
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.76      0.76     19080
           1       0.86      0.84      0.85     19518
           2       0.58      0.68      0.63     16713
           3       0.53      0.57      0.55     18700
           4       0.72      0.66      0.69     21436
           5       0.65      0.66      0.65     20609
           6       0.82      0.75      0.78     21922
           7       0.78      0.78      0.78     20311
           8       0.86      0.82      0.84     20771
           9       0.82      0.83      0.83     20940

    accuracy                           0.74    200000
   macro avg       0.74      0.74      0.74    200000
weighted avg       0.74      0.74      0.74    200000
```

Evaluation Graphs





The changes I made to get 2nd model :

- **I added learning rate scheduler For convergence speed**
- **I added new `Conv layer and Batch Normalization`**
- **I added `weight_decay` =1e-4 as a form of L2 regularization applied to the optimizer to help prevent overfitting.**
- **I added `lr_scheduler` to speed convergence**
- **I changed the number of epochs to be 50 instead of 20 (I actually tried 30 and I applied some of the changes above but i got (80,82,83) as Accuracy so I kept the best of them which was the below model**

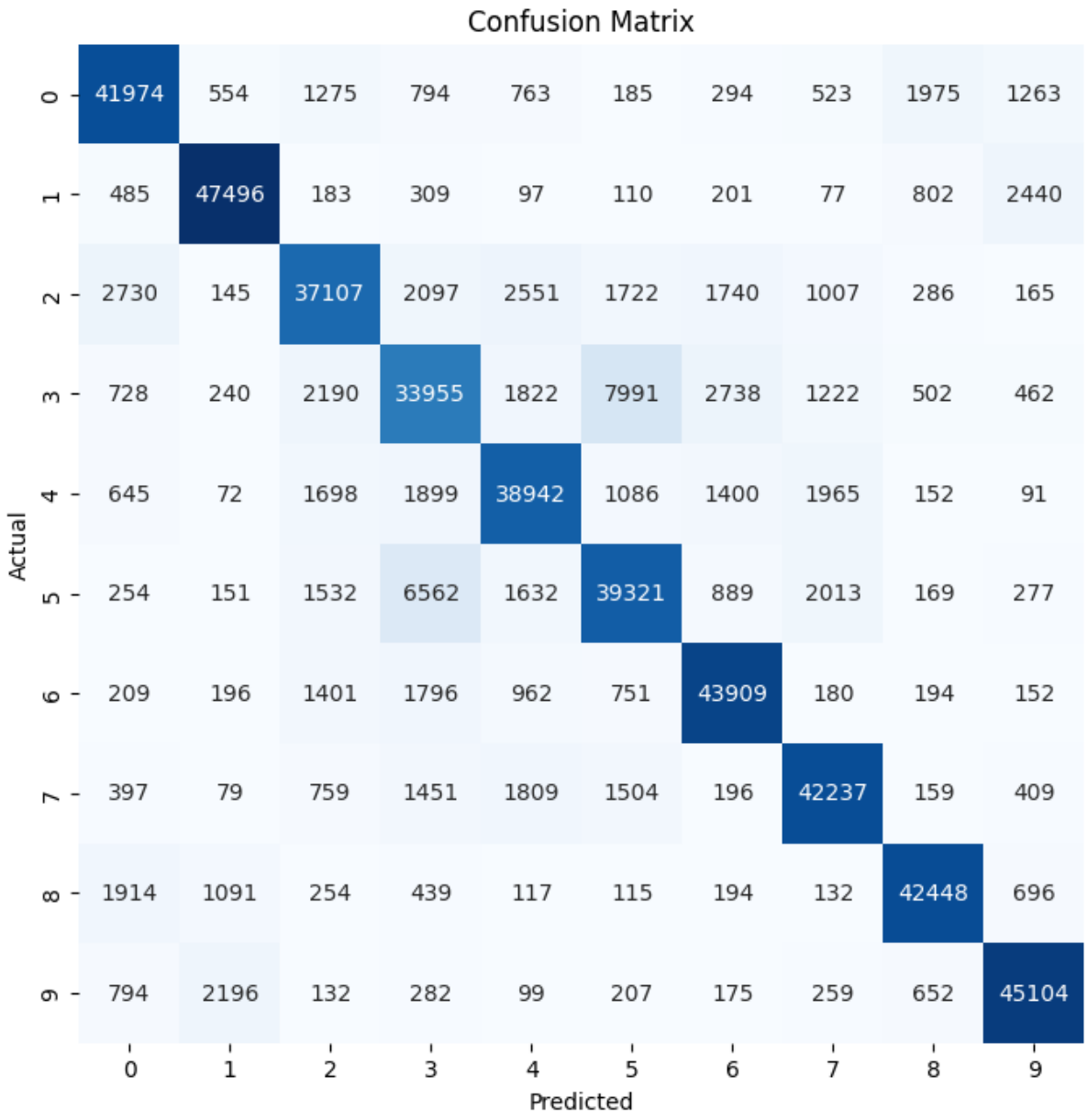
Second Model Arch :

```
class ImprovedNet(nn.Module):
    def __init__(self, input_channels=3, input_size=32):
        super(ImprovedNet, self).__init__()
        self.conv1 = nn.Conv2d(input_channels, 64, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.conv4 = nn.Conv2d(256, 512, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        self.pool = nn.MaxPool2d(2, 2)
        fc_input_size = 512 * (input_size // 16) * (input_size // 16)

        self.fc1 = nn.Linear(fc_input_size, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 10)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = self.pool(F.relu(self.bn4(self.conv4(x))))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

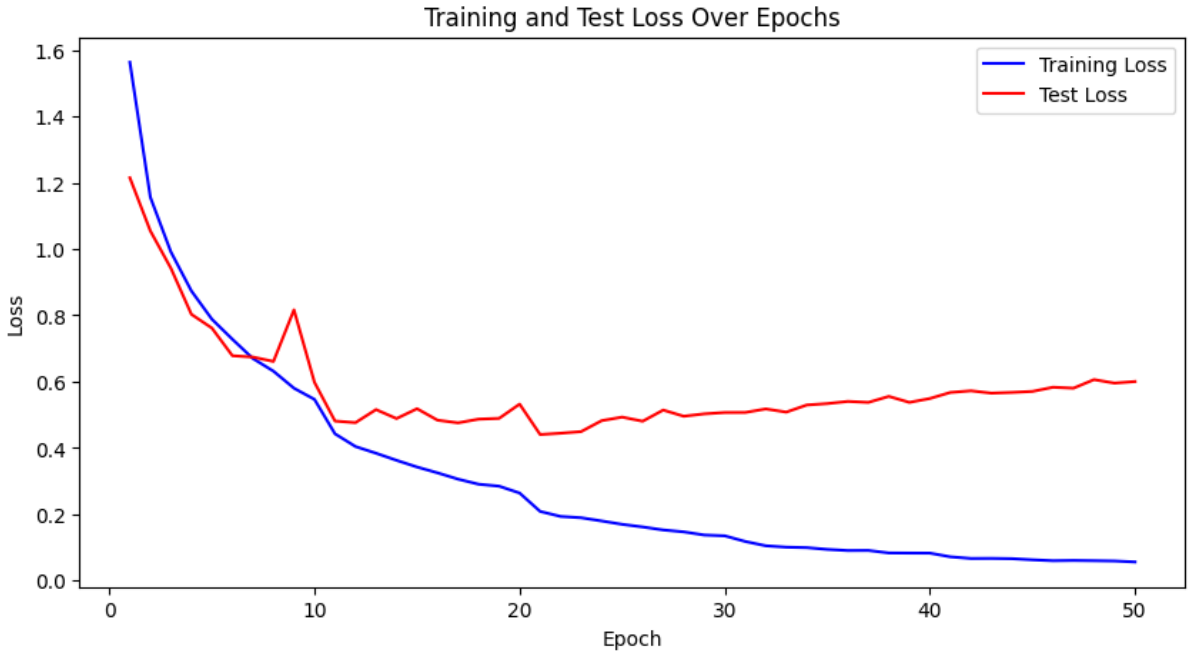
Confusion Matrix and Classification Report :

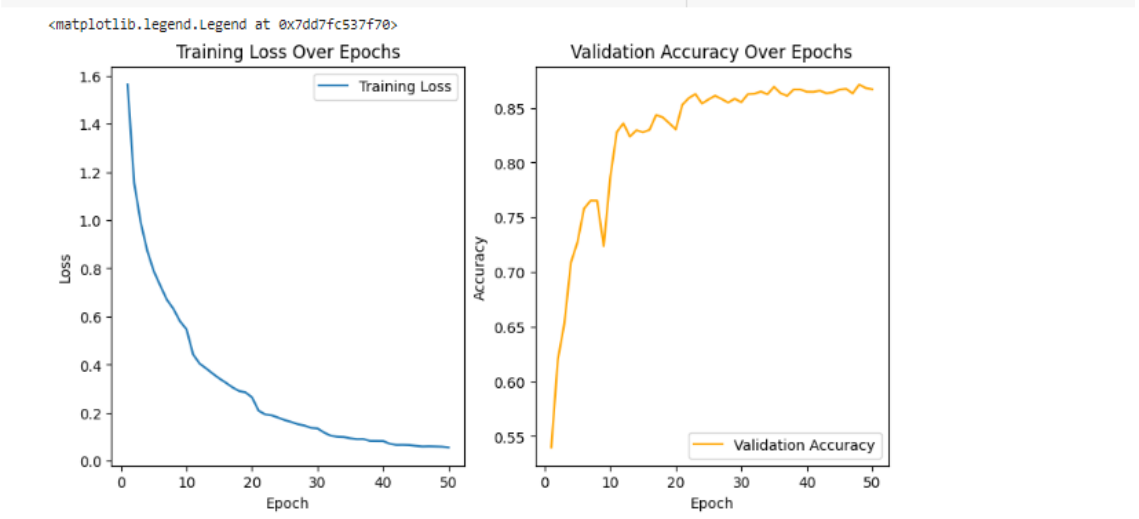


Classification Report:

	precision	recall	f1-score	support
0	0.84	0.85	0.84	49600
1	0.91	0.91	0.91	52200
2	0.80	0.75	0.77	49550
3	0.68	0.65	0.67	51850
4	0.80	0.81	0.81	47950
5	0.74	0.74	0.74	52800
6	0.85	0.88	0.87	49750
7	0.85	0.86	0.86	49000
8	0.90	0.90	0.90	47400
9	0.88	0.90	0.89	49900
accuracy			0.82	500000
macro avg	0.82	0.83	0.83	500000
weighted avg	0.82	0.82	0.82	500000

Evaluation Graphs:

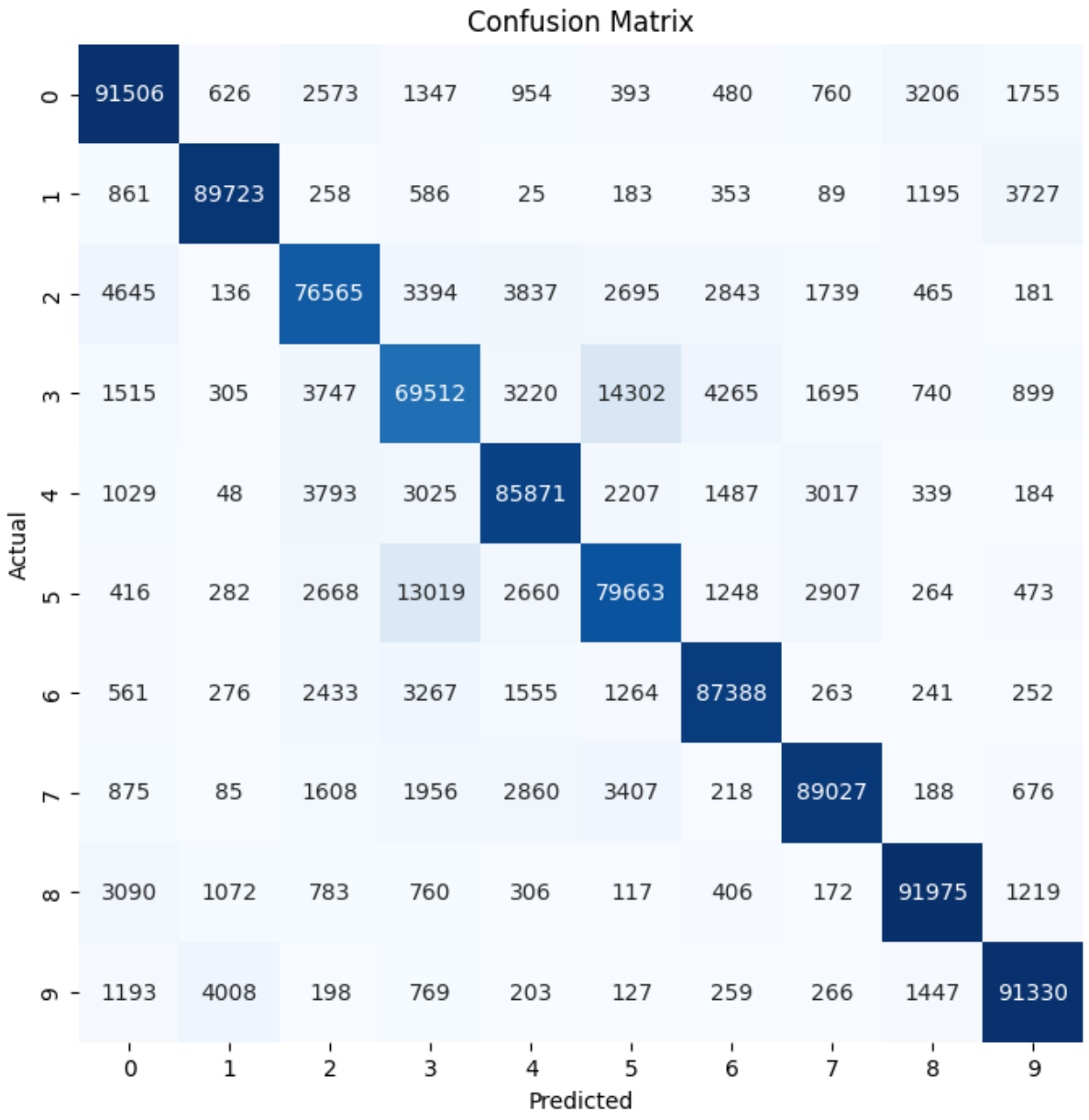




The Third Model :

Same architecture but I increased the number of epochs from 50 to 100

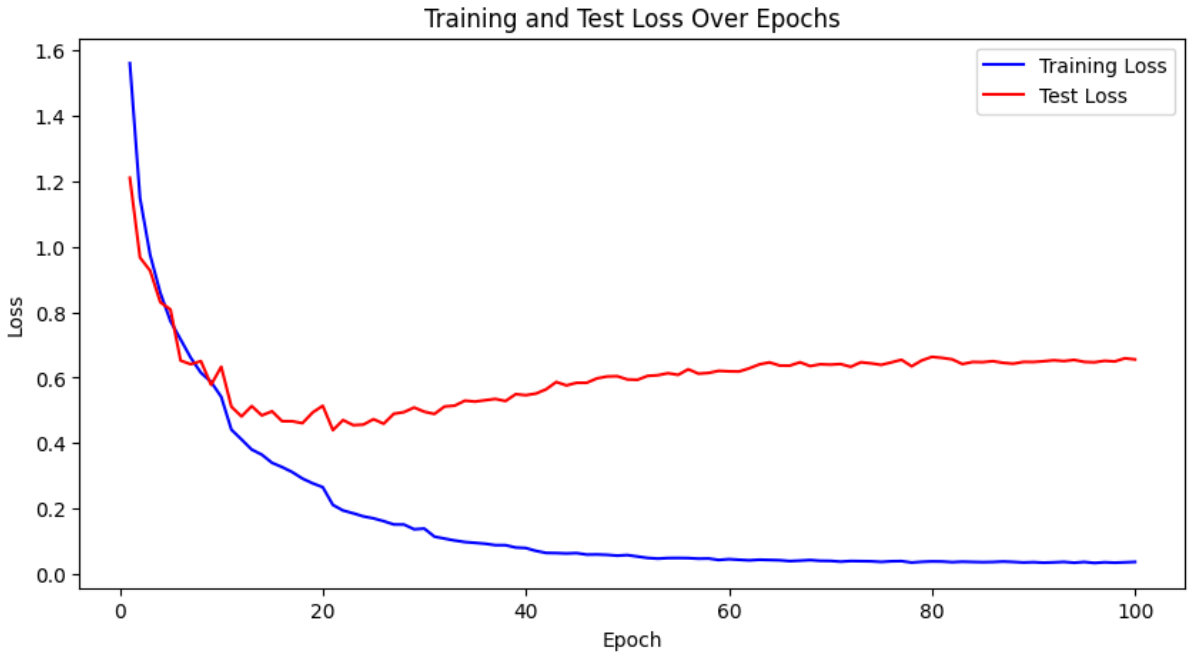
Confusion Matrix and Classification Report :

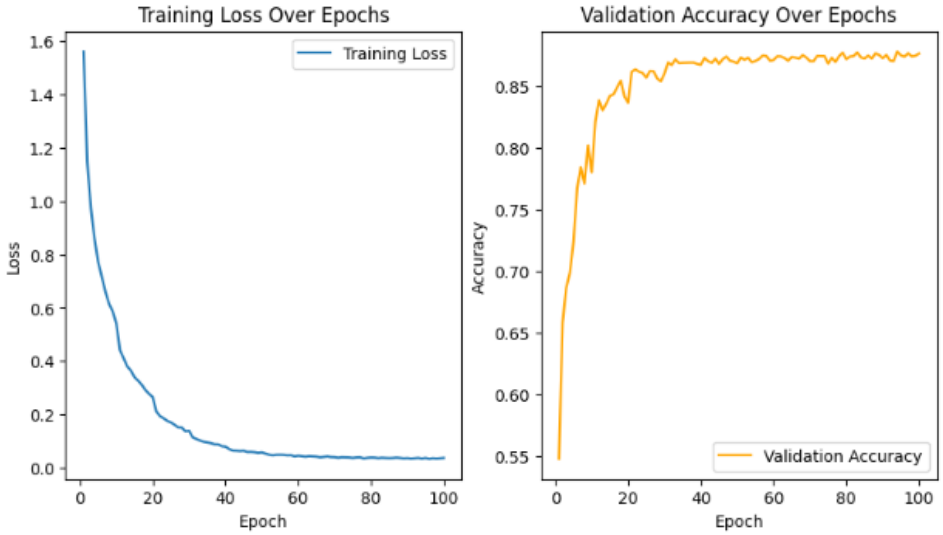


Classification Report:

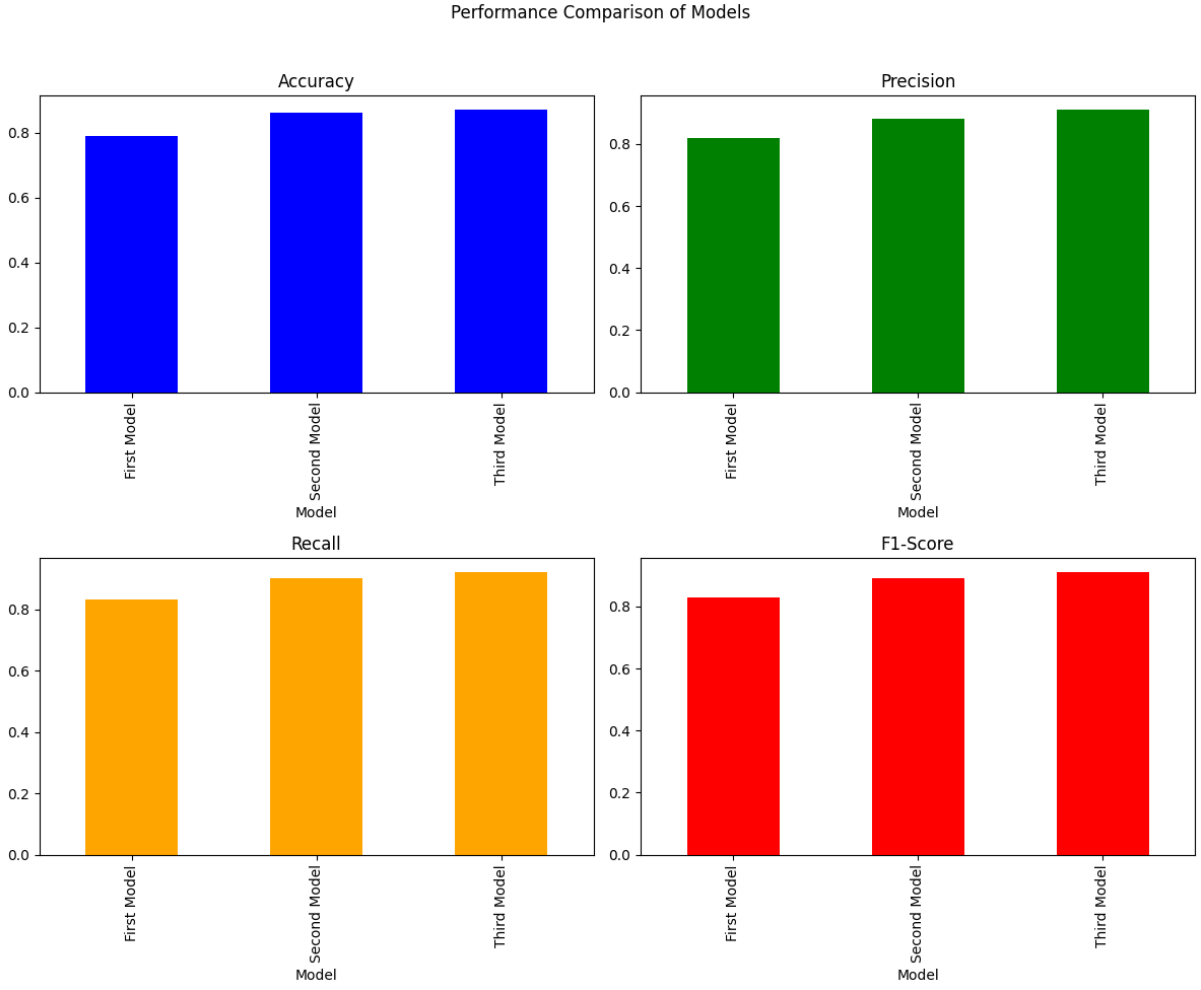
	precision	recall	f1-score	support
0	0.87	0.88	0.87	103600
1	0.93	0.92	0.93	97000
2	0.81	0.79	0.80	96500
3	0.71	0.69	0.70	100200
4	0.85	0.85	0.85	101000
5	0.76	0.77	0.77	103600
6	0.88	0.90	0.89	97500
7	0.89	0.88	0.89	100900
8	0.92	0.92	0.92	99900
9	0.91	0.92	0.91	99800
accuracy			0.85	1000000
macro avg	0.85	0.85	0.85	1000000
weighted avg	0.85	0.85	0.85	1000000

Evaluation Graphs:

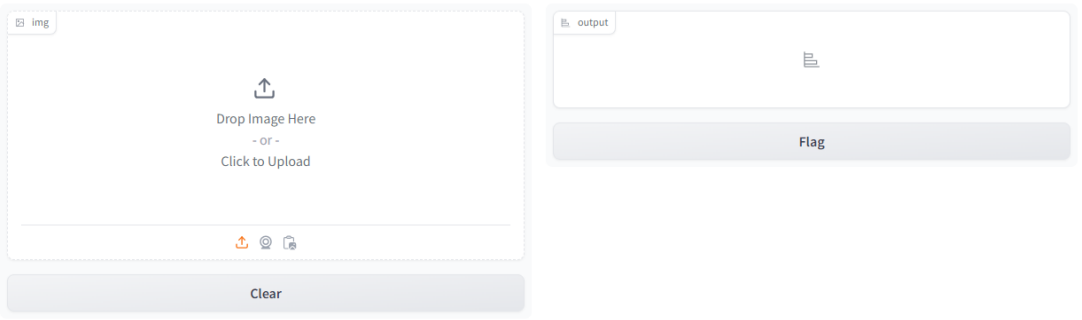


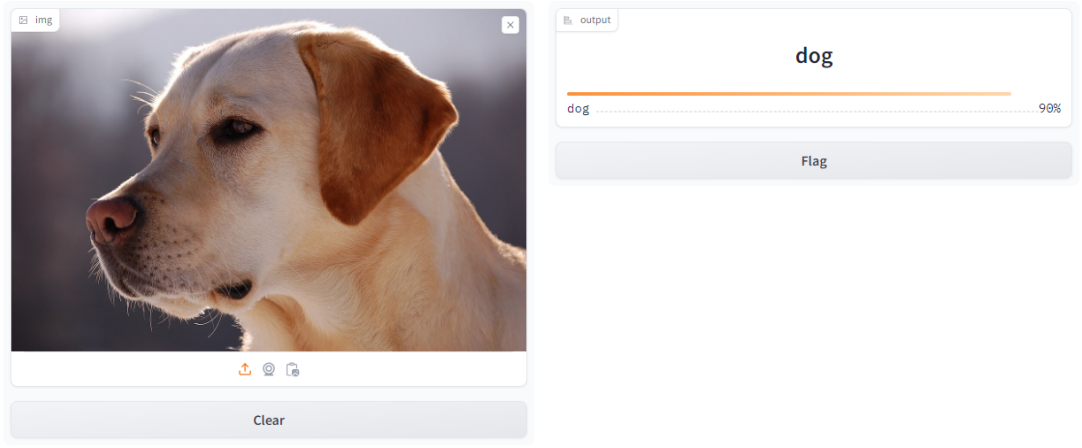


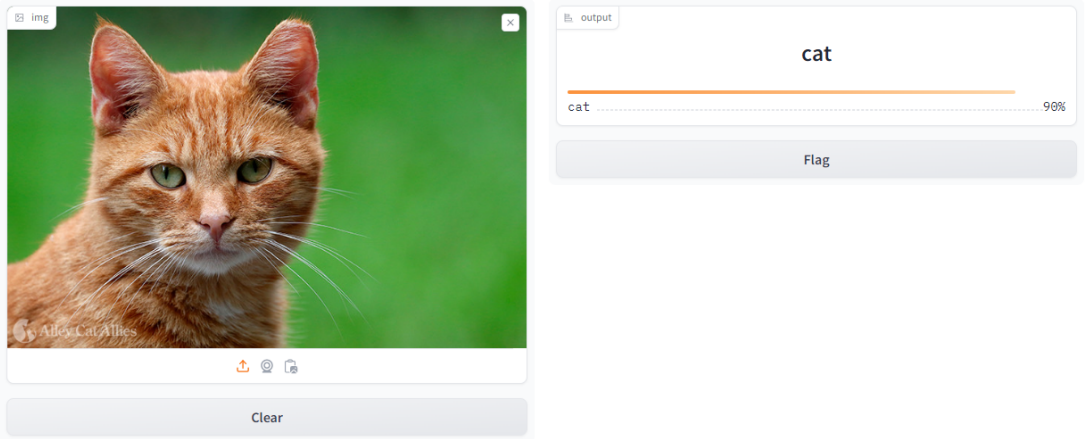
Model comparison :



Demo :


img
Drop Image Here
- or -
Click to Upload
Clear
output
Flag


Use via API
Built with Gradio
img
Clear
output
dog
dog
90%
Flag


img
Alley Cat Allies
Clear
output
cat
cat
90%
Flag