CISC3024 Project

Classification of Street View House Numbers Using PyTorch

AgustinYan DC125714

SVHN

Street View House Number dataset

- Real-world image dataset
- Similar to MNIST
- But it is harder and unsolved

Data Processing and Augmentation

```python
class SVHNAugmentedDataset(torch.utils.data.Dataset):
    def __init__(self, root, split='train', download=True,
                 max_rotation=15, min_crop_size=16, max_aspect_ratio_change=0.1, train=True):
        self.dataset = SVHN(root=root, split=split, download=download) # load the dataset
        self.train = train # to control the train and test

        self.augment = A.Compose([ # when in train mode, do the augmentation
            A.Rotate(limit=max_rotation, p=0.5),
            A.RandomResizedCrop(height=32, width=32, scale=(min_crop_size/32, 1.0), ratio=(1-max_aspect_ratio_change, 1+max_aspect_ratio_ch
            A.Normalize(mean=[0.4377, 0.4438, 0.4728], std=[0.1980, 0.2010, 0.1970]),
            ToTensorV2()
        ])

        self.normalize = A.Compose([ # when in test mode, only do normalization
            A.Normalize(mean=[0.4377, 0.4438, 0.4728], std=[0.1980, 0.2010, 0.1970]),
            ToTensorV2()
        ])

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

    def __getitem__(self, idx):
        image, label = self.dataset[idx]
        image = np.array(image)

        if self.train:
            augmented = self.augment(image=image)
        else:
            augmented = self.normalize(image=image)

        image = augmented['image']
        return image, label
```

Data Processing and Augmentation

```
# load and transform the dataset
train_dataset = SVHNAugmentedDataset(root='./data', split='train', download=True, train=True)
test_dataset = SVHNAugmentedDataset(root='./data', split='test', download=True, train=False)

# create dataloader
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

Neural Network Setup

```
class SmallVGG(nn.Module):
    def __init__(self):
        super(SmallVGG, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 8, kernel_size=3, padding=1), nn.GroupNorm(2, 8), nn.ReLU(),
            nn.Conv2d(8, 16, kernel_size=3, padding=1), nn.GroupNorm(4, 16), nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), nn.Dropout(0.25),

            nn.Conv2d(16, 32, kernel_size=3, padding=1), nn.GroupNorm(8, 32), nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.GroupNorm(8, 32), nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), nn.Dropout(0.25),

            nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.GroupNorm(8, 32), nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1), nn.GroupNorm(8, 32), nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), nn.Dropout(0.25)
        )

        self.fc_layers = nn.Sequential(
            nn.Linear(32 * 4 * 4, 256), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10) #output layer 10 classes
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x
```

Training and Evaluation process

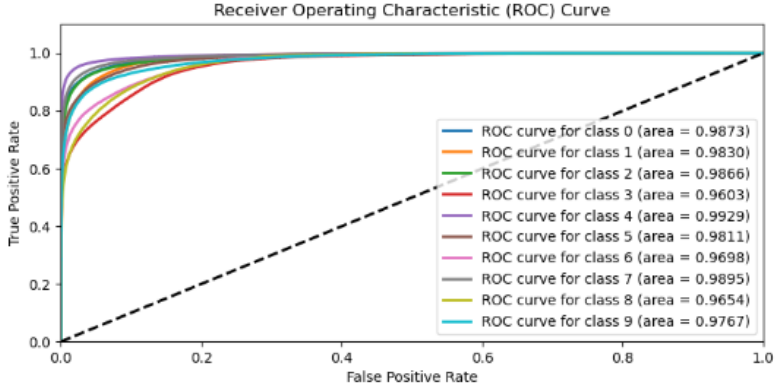
```
def train_and_evaluate(model, train_loader, test_loader, criterion, optimizer, num_epochs):
    train_losses = []
    test_losses = []
    test_accuracies = []
    all_labels = []
    all_scores = []

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

        # Training loop
        for images, labels in tqdm(train_loader):
            images, labels = images.to(device), labels.to(device)

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

            running_loss += loss.item() * images.size(0)

        epoch_train_loss = running_loss / len(train_loader.dataset)
        train_losses.append(epoch_train_loss)

        model.eval()
        test_loss = 0.0
        correct = 0
        total = 0
```

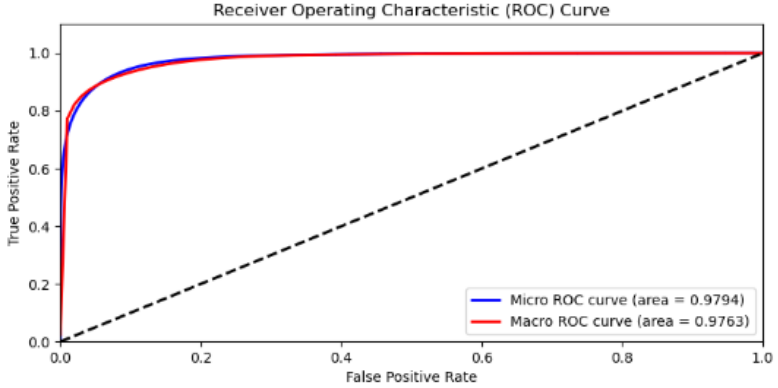
Training and Evaluation process

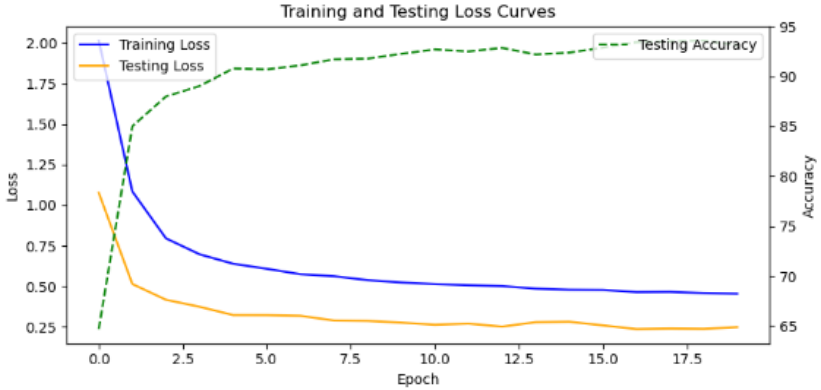
```
    # Evaluation loop
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item() * images.size(0)

            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_labels.append(labels.cpu().numpy())
            all_scores.append(outputs.cpu().numpy())

    epoch_test_loss = test_loss / len(test_loader.dataset)
    test_losses.append(epoch_test_loss)
    accuracy = 100 * correct / total
    test_accuracies.append(accuracy)

    print(f"Epoch [{epoch + 1}/{num_epochs}] - "
          f"Train Loss: {epoch_train_loss:.4f}, "
          f"Test Loss: {epoch_test_loss:.4f}, "
          f"Accuracy: {accuracy:.2f}%")

# Convert lists to numpy arrays for ROC AUC computation
all_labels = np.concatenate(all_labels)
all_scores = np.concatenate(all_scores)

return train_losses, test_losses, test_accuracies, all_labels, all_scores
```

Analysis Training and Evaluation process

Optimizer (with different Learning rate)





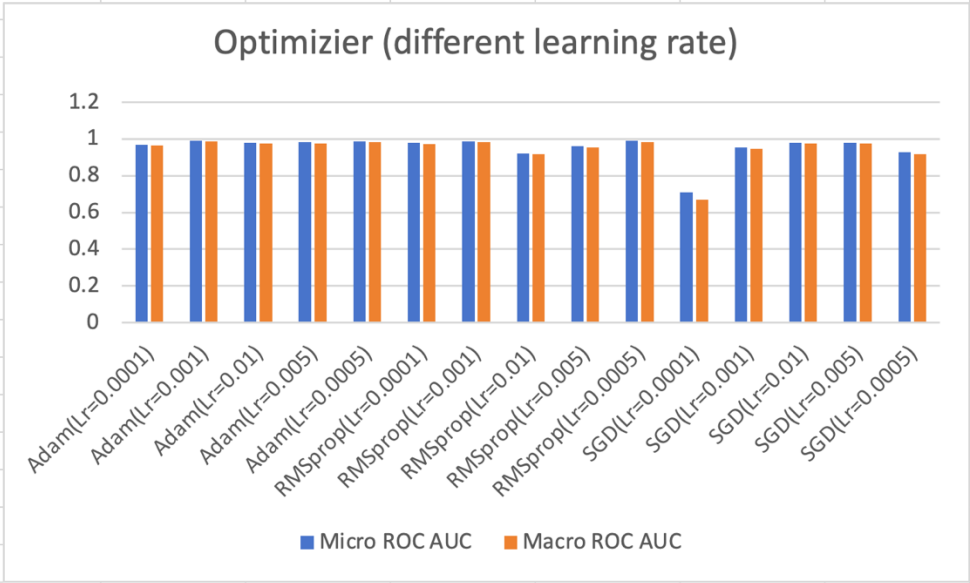


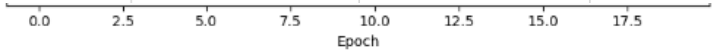
Optimizer (with different Learning rate)

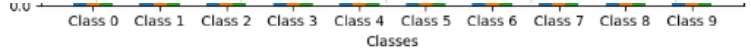
Receiver Operating Characteristic (ROC) Curve

Receiver Operating Characteristic (ROC) Curve

	Micro ROC AUC	Macro ROC AUC
Adam(Lr=0.0001)	0.9685	0.9626
Adam(Lr=0.001)	0.9901	0.9859
Adam(Lr=0.01)	0.9794	0.9763
Adam(Lr=0.005)	0.9808	0.9763
Adam(Lr=0.0005)	0.9871	0.9829
RMSprop(Lr=0.0001)	0.9774	0.9724
RMSprop(Lr=0.001)	0.9869	0.9833
RMSprop(Lr=0.01)	0.9216	0.9171
RMSprop(Lr=0.005)	0.9598	0.9537
RMSprop(Lr=0.0005)	0.9881	0.9842
SGD(Lr=0.0001)	0.7088	0.6703
SGD(Lr=0.001)	0.9521	0.9452
SGD(Lr=0.01)	0.9796	0.9748
SGD(Lr=0.005)	0.9804	0.9759
SGD(Lr=0.0005)	0.9269	0.916

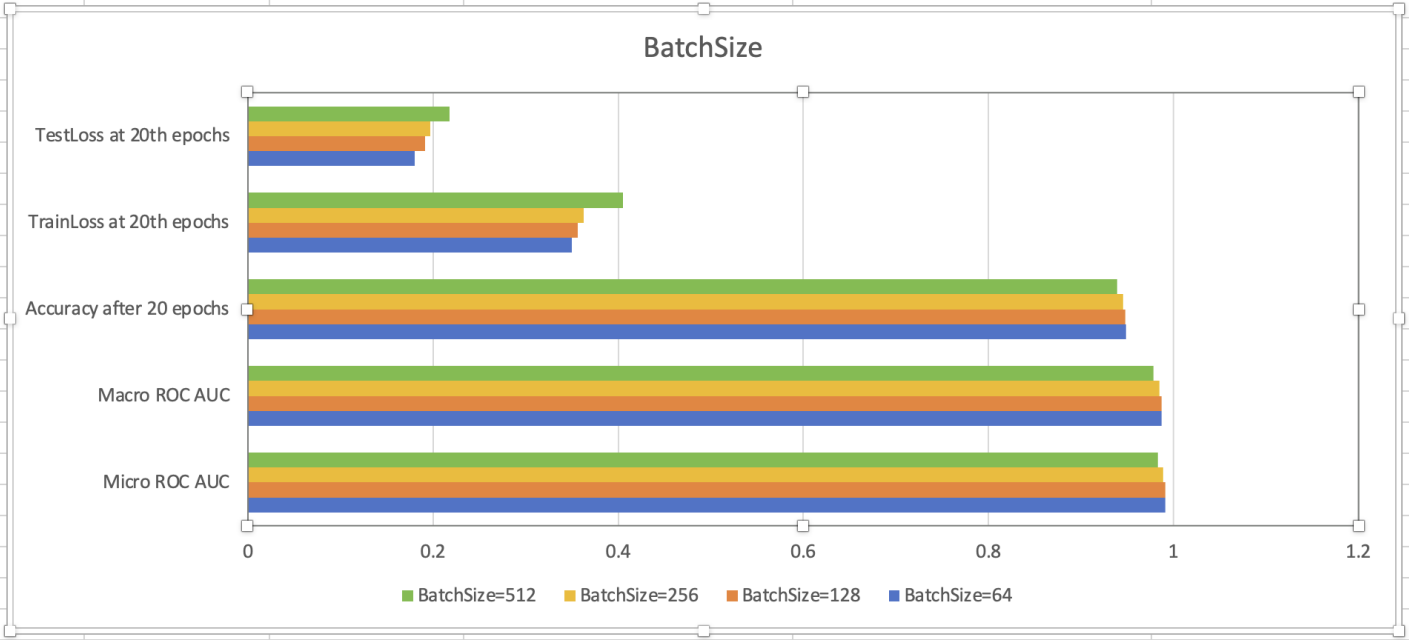






BatchSize

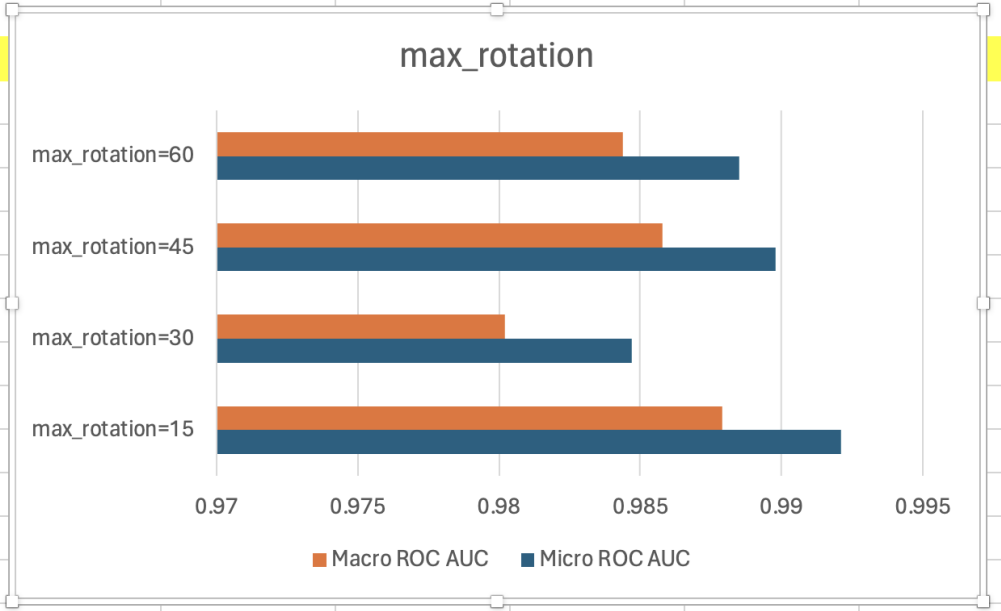
	Micro ROC AUC	Macro ROC AUC	Accuracy after 20 epochs	TrainLoss at 20th epochs	TestLoss at 20th epochs
BatchSize=64	0.9913	0.9873	94.91%	0.35	0.18
BatchSize=128	0.991	0.987	94.79%	0.3564	0.1912
BatchSize=256	0.9889	0.9847	94.54%	0.363	0.1968
BatchSize=512	0.9832	0.9786	93.91%	0.4047	0.2175



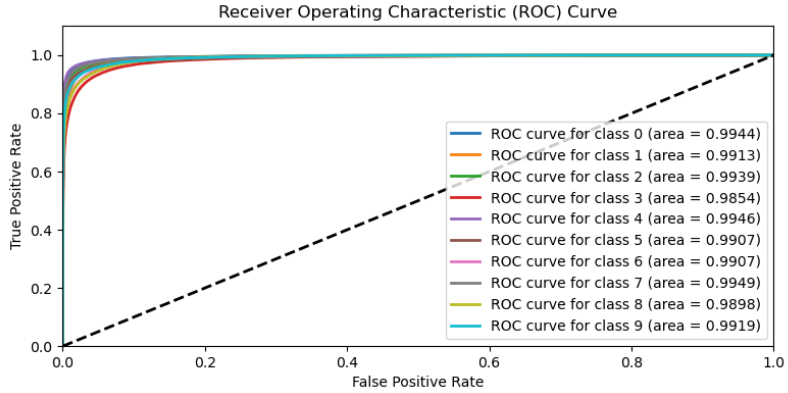
Maximum angle of rotation

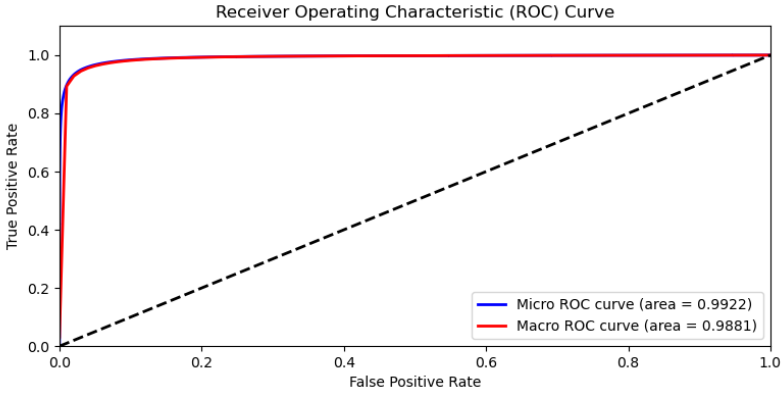
	Micro ROC AUC	Macro ROC AUC
max_rotation=15	0.9921	0.9879
max_rotation=30	0.9847	0.9802
max_rotation=45	0.9898	0.9858
max_rotation=60	0.9885	0.9844

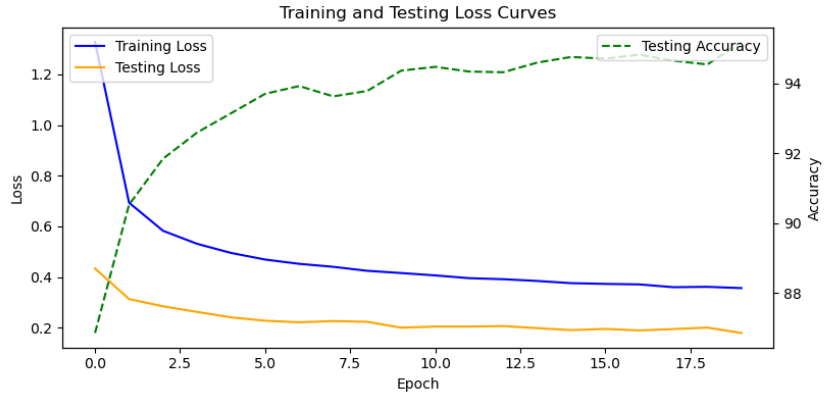
图表区



Minimum Crop Size

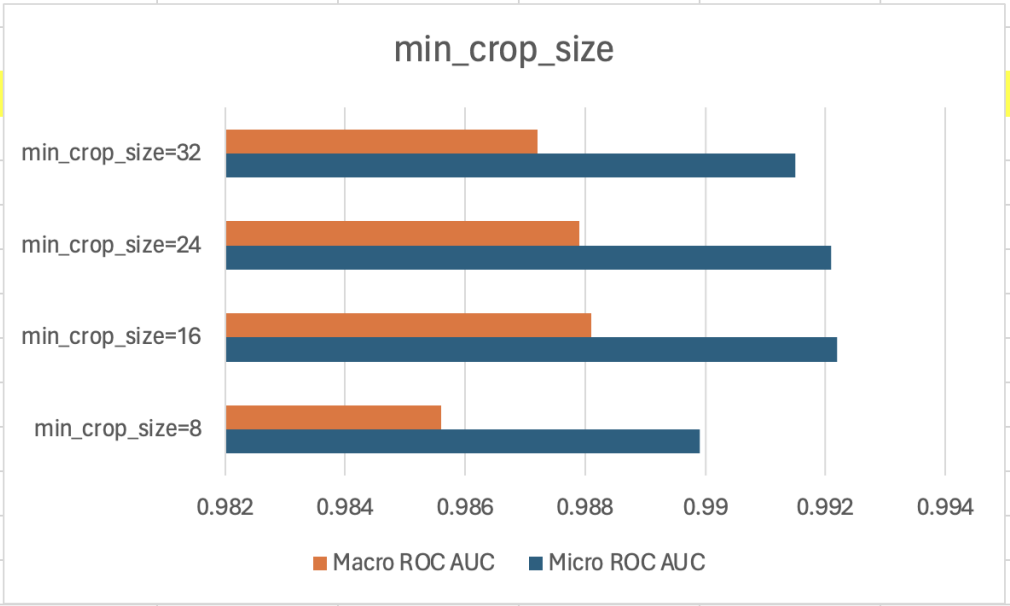




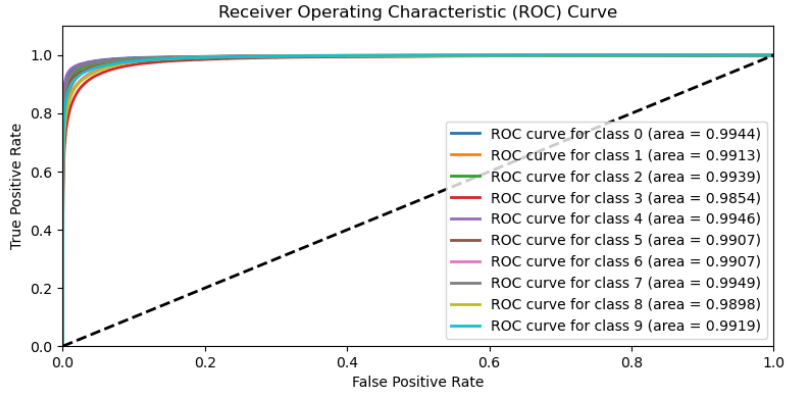


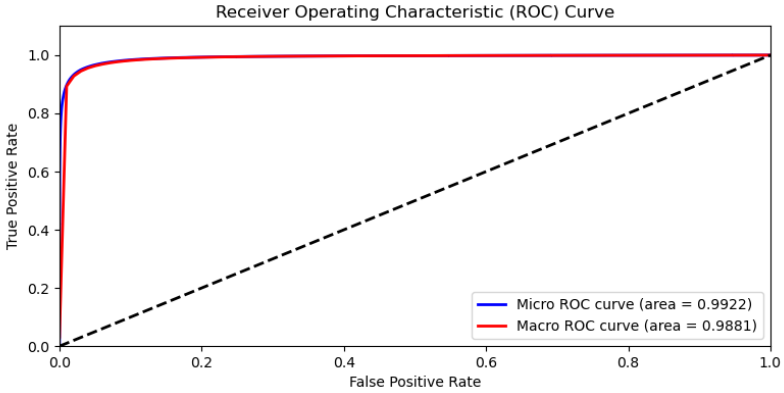
Minimum Crop Size

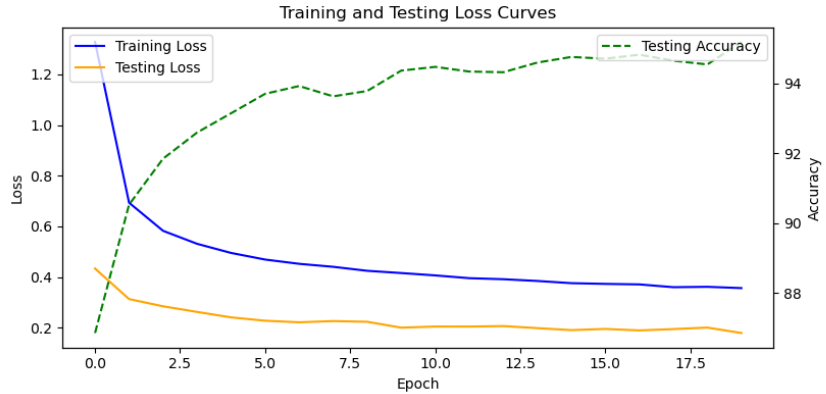
	Micro ROC AUC	Macro ROC AUC
min_crop_size=8	0.9899	0.9856
min_crop_size=16	0.9922	0.9881
min_crop_size=24	0.9921	0.9879
min_crop_size=32	0.9915	0.9872



Maximum aspect ratio change

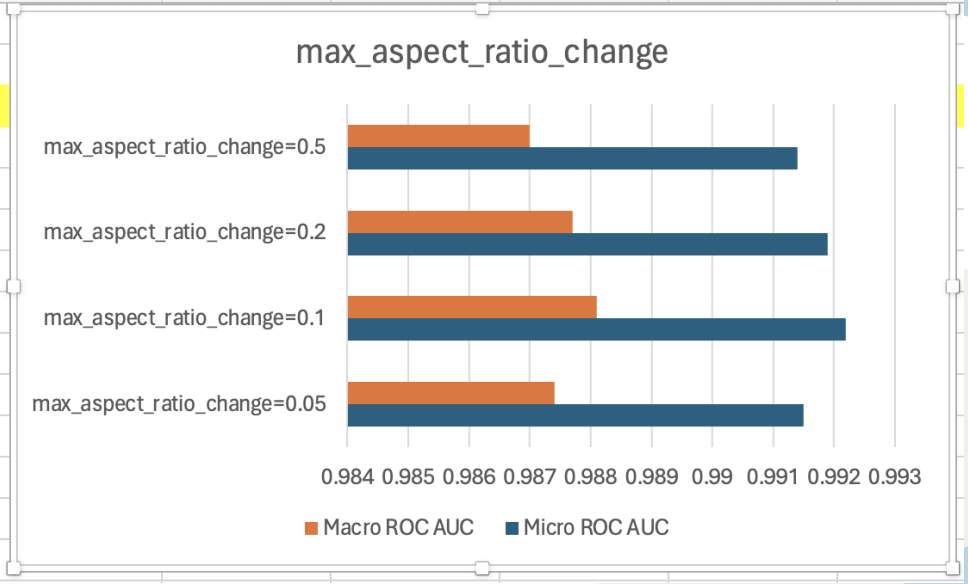






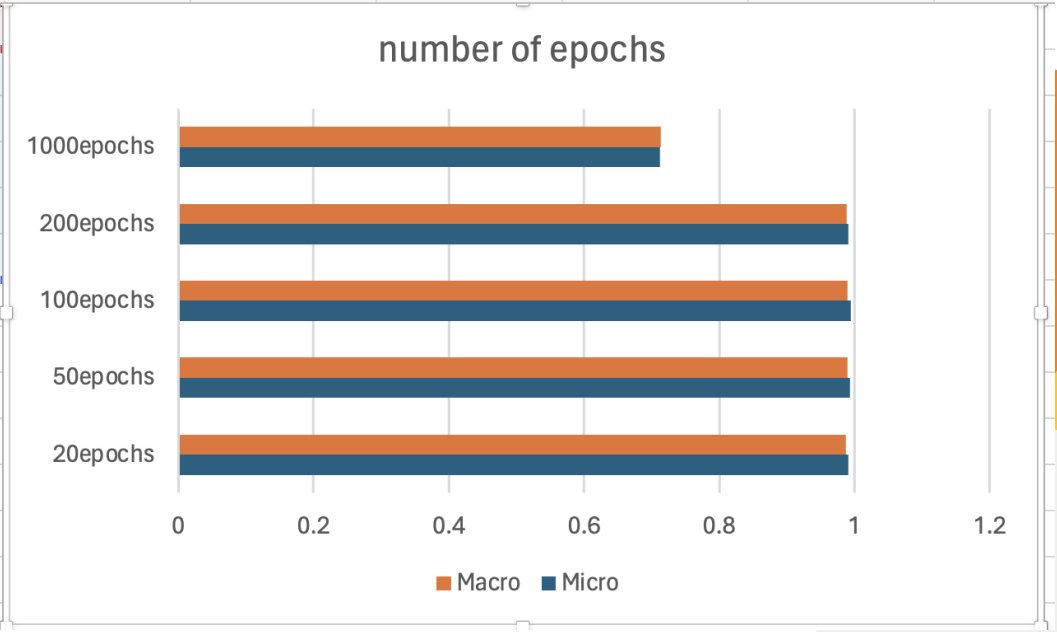
Maximum aspect ratio change

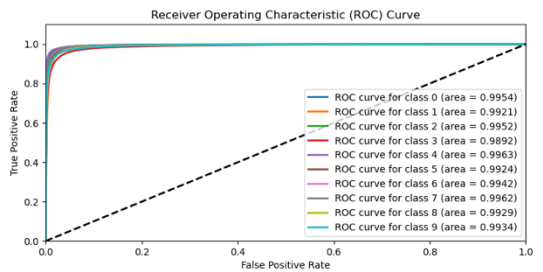
	Micro ROC AUC	Macro ROC AUC
max_aspect_ratio_change=0.05	0.9915	0.9874
max_aspect_ratio_change=0.1	0.9922	0.9881
max_aspect_ratio_change=0.2	0.9919	0.9877
max_aspect_ratio_change=0.5	0.9914	0.987

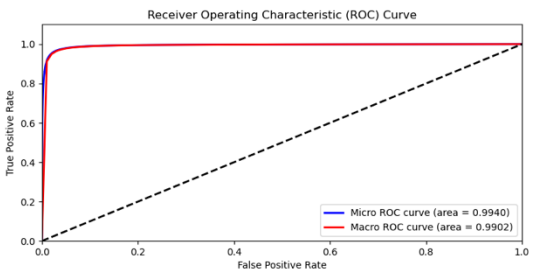


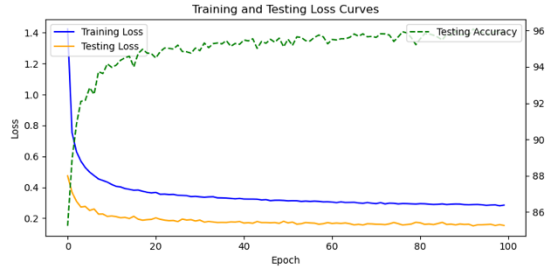
Number of Epochs

	Micro	Macro
20epochs	0.9909	0.9868
50epochs	0.9938	0.9896
100epochs	0.994	0.9902
200epochs	0.9903	0.9879
1000epochs	0.7122	0.713

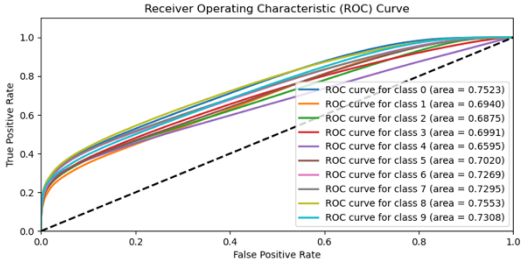


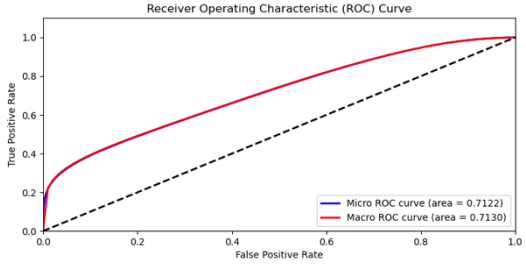


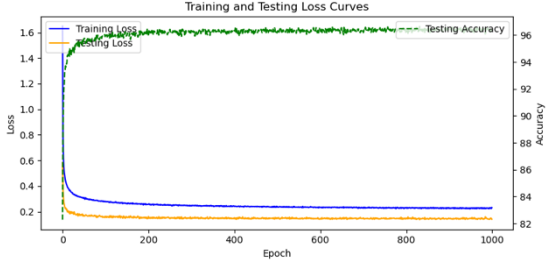




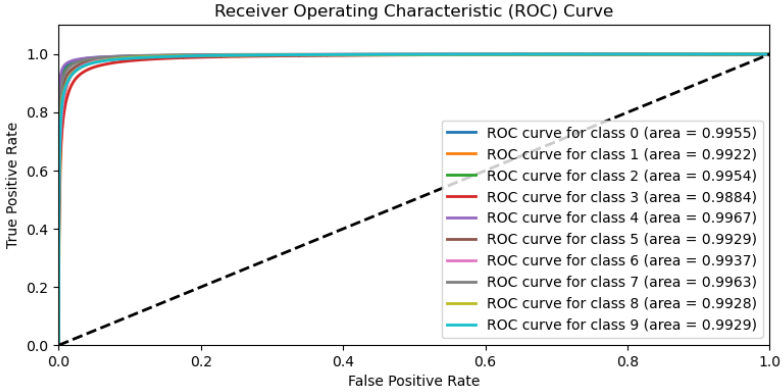
100 Epochs and 1000 Epochs

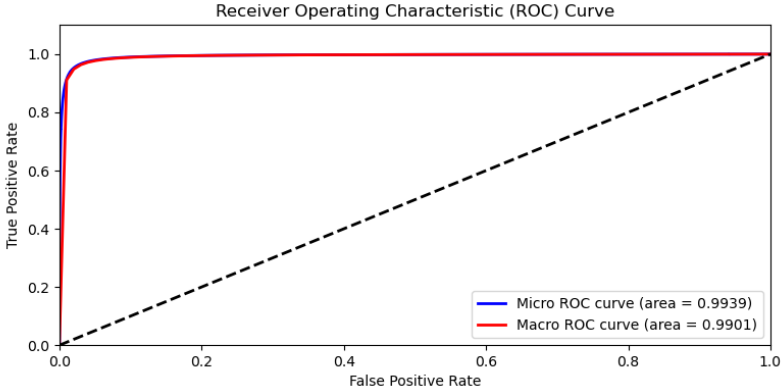


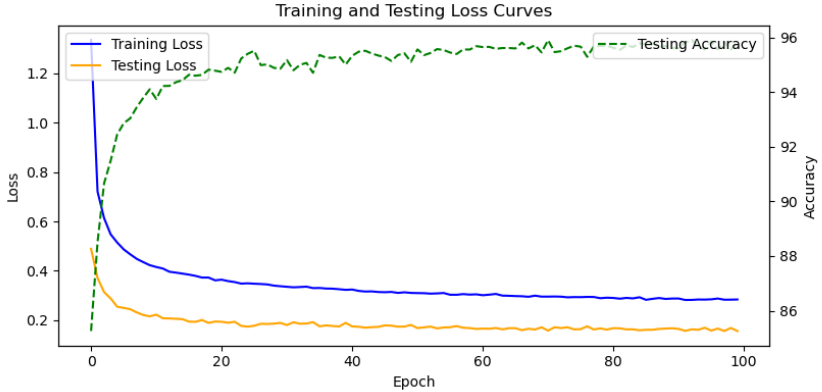




Final performance of the model







Model Demonstration

With Sketchpad in Gradio

Thanks!

For your kind listening

Any questions?