CISC3024 Pattern Recognition Final Project

Group Members:

- Huang Yanzhen, DC126732
- Mai Jiajun, DC127853

0. Project Setup

0.1 Packages & Device

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader, Subset
from tqdm import tqdm

import numpy as np
import cv2
import os
import time
```

[2]:
```
device_name = "cuda" if torch.cuda.is_available() else "cpu"
device = torch.device(device_name)
print(f"Using device: {device_name}")
```

```
Using device: cuda
```

0.2 Global Configurations

[3]:
```
path_dataset = "./data/SVHN_mat"
norm_mean = [0.4377, 0.4438, 0.4728]
norm_std = [0.1980, 0.2010, 0.1970]
```

1. Data Processing and Augmentation

1.1 Download Datasets

Define dataset class, retrieve dataset.

[4]:
```
import albumentations as A
from albumentations.pytorch import ToTensorV2
import scipy.io as sio
```

```
E:\Courses\CISC3024-Pattern-Recognition\cisc3024_pr_venv\lib\site-packages\albumentations\__init__.py:13: UserWarning: A new version of Albumentatio
ns is available: 1.4.20 (you have 1.4.18). Upgrade using: pip install -U albumentations. To disable automatic update checks, set the environment var
iable NO_ALBUMENTATIONS_UPDATE to 1.
  check_for_updates()
```

[5]:
```
class SVHNDataset(Dataset):
    def __init__(self, mat_file, transform=None):
        data = sio.loadmat(mat_file)
        self.images = np.transpose(data['X'], (3, 0, 1, 2))
        self.labels = data['y'].flatten()
        self.labels[self.labels == 10] = 0
        self.transform = transform

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

    def __getitem__(self, idx):
        image = self.images[idx]
        label = self.labels[idx]
```

```
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image=image)['image']

        return image, label
```

[6]:
```
transform = A.Compose([
    A.RandomResizedCrop(32, 32),
    A.Rotate(limit=30),
    A.Normalize(mean=norm_mean, std=norm_std),
    ToTensorV2()
])

train_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"train_32x32.mat"), transform=transform)
test_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"test_32x32.mat"), transform=transform)
extra_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"extra_32x32.mat"), transform=transform)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
extra_loader = DataLoader(Subset(extra_dataset, indices=list(range(30000))), batch_size=64, shuffle=False)

print(f"Train Size:{train_dataset.__len__()}\nTest Size:{test_dataset.__len__()}\nExtra Size:{extra_dataset.__len__()}")
```

```
Train Size:73257
Test Size:26032
Extra Size:531131
```

1.2 Peak A Data

+ 2 cells hidden

2. Neuron Network Structure

2.1 Specify Model Structure

[8]:
```
class SmallVGG(nn.Module):
    def __init__(self, frame_size=32):
        super(SmallVGG, self).__init__()
        self.frame_size = frame_size
        self. conv_layers = nn.Sequential(
            nn.Conv2d(3, 8, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(8, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), # 16x16

            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), # 8x8

            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), # 4x4
        )

        self.fc_layers = nn.Sequential(
            nn.Linear(frame_size * 4 * 4, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )

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

2.2 Initialize with Hyper Parameters

[9]:
```
num_epochs = 30
learning_rate = 0.001
model = SmallVGG().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

2.3 Train and Evaluate

+2 cells hidden

2.4 Visualize Result

Multiple functions are defined to evaluate data. Below is a list of them.

`get_predictions`

Get a model prediction on 30,000 data on `extra.mat`.

- params:
  - `model_path`: The directory at which the model is stored.
  - `extra_loader`: The `DataLoader` object for loading `extra.mat`.
- returns:
  - `pred_scores`: A list of softmax-normalized predictions, one list of probabilities per image.
  - `true_labels`: A list of ground-truth labels.
  - `pred_labels`: A list of predicted labels.

`display_cm`

Plots the Confusion Matrix.

- params:
  - `true_labels`: A list of ground-truth labels.
  - `pred_labels`: A list of predicted labels.
- returns: `None`

`get_metrics`

Get evaluation metrics based on the given prediction results.

- params:
  - `true_labels`: A list of ground-truth labels.
  - `pred_labels`: A list of predicted labels.
- returns:
  - `accuracy`: A single universal accuracy. $acc = \frac{\#.\text{ of correctly classified data}}{\#.\text{ of data}}$
  - `precision`: A class-wise list of classification precisions. $P_i = \frac{TP}{TP+FP}$
  - `recall`: A class-wise list of classification recalls. $R_i = \frac{TP}{TP+FN}$
  - `f1`: A class-wise list of harmonic mean of precision and recall. $F_1 = \frac{2}{\frac{1}{P}+\frac{1}{R}}$

`display_pr_curve`

Plot the precision-recall curve.

- params:
  - `true_labels`: A list of ground-truth labels.
  - `pred_scores`: A list of softmax-normalized predictions, one list of probabilities per image.
- returns: `None`

`get_roc_auc`

Get the `roc_auc` scores.

- params:
  - `true_labels`: A list of ground-truth labels.
  - `pred_scores`: A list of softmax-normalized predictions, one list of probabilities per image.
- returns:
  - `roc_auc`: A class-wise list of `roc_auc` scores.

`display_roc_auc`

Plot the `roc` curve.

- params:

- `true_labels` : A list of ground-truth labels.
- `pred_scores` : A list of softmax-normalized predictions, one list of probabilities per image.

- returns: `None`

+22 cells hidden

3. Experiments

[19]:
```
import itertools
```

3.0 Plot Functions

The experiments will be a list of the following structures:

```
{
    "HYPER_PARAM_1": combo[0],
    "HYPER_PARAM_2": combo[1],
    "train_losses": train_losses,
    "test_losses": test_losses,
    "model_state_dict": exp_model.state_dict()
}
```

3.0.0 Epoch-Loss Curves

[194]:
```
def plot_el(loaded_experiments, hyper_param_names, n_rows=4, n_cols=4):
    fig_size = (n_cols * 5, n_rows * 5)
    n1, n2 = hyper_param_names

    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=fig_size)
    # plt.tight_layout()

    for i, ax in enumerate(axes.flat):
        train_losses, test_losses = loaded_experiments[i]["train_losses"], loaded_experiments[i]["test_losses"]

        train_difference = train_losses[-1]-train_losses[0]
        test_difference = test_losses[-1]-test_losses[0]

        ax.plot(train_losses, label=f"Training Loss, Diff={train_difference:.3f}")
        ax.plot(test_losses, label=f"Testing Loss, Diff={test_difference:.3f}")
        ax.set_xlabel("Epochs")
        ax.set_ylabel("Loss")
        ax.set_title(f"{n1}={loaded_experiments[i][n1]}, {n2}={loaded_experiments[i][n2]}")
        ax.legend()

    plt.show()
```

3.0.1 Get Experiment Results

[87]:
```
def get_experiment_results(loaded_experiments, test_hyperparam_names, extra_loader):
    experiment_results = []
    n1, n2 = test_hyperparam_names
    for i, exp in enumerate(loaded_experiments):
        pred_scores, true_labels, pred_labels = get_predictions(exp['model_state_dict'], extra_loader)
        experiment_results.append({
            n1: exp[n1],
            n2: exp[n2],
            "true_labels": true_labels,
            "pred_labels": pred_labels,
            "pred_scores": pred_scores
        })

        print(f"First 10 true labels:")
        [print(num, end=" ") for num in true_labels[:10]]
        print(f"...\n")

        print(f"First 10 pred labels:")
        [print(num, end=" ") for num in pred_labels[:10]]
        print(f"...\n")

        print(f"First 5 pred_scores:")
        [print(num, end=" ") for num in pred_scores[:5]]
        print(f"...\n")

        # del pred_scores, true_labels, pred_lables
        torch.cuda.empty_cache()
    return experiment_results
```

3.0.2 Confusion Matrix

[105]:
```
def plot_cm(experiment_results, hyper_param_names, n_rows=4, n_cols=4):
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 5, n_rows * 5))
    axes = axes.flatten()

    hparam_1, hparam_2 = hyper_param_names

    for i, exp_rs in enumerate(experiment_results):
        true_labels, pred_labels = exp_rs['true_labels'], exp_rs['pred_labels']
        cm = confusion_matrix(true_labels, pred_labels)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(0,10))
        disp.plot(ax=axes[i], cmap = plt.cm.Blues)
        axes[i].set_title(f"Exp {i+1}: {hparam_1}={exp_rs[hparam_1]}, {hparam_2}={exp_rs[hparam_2]}")

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

3.0.3 Precision-Recall Curve

[125]:
```
def plot_pr(experiment_results, hyper_param_names, n_rows=4, n_cols=4):
    fig, axes = plt.subplots(n_rows,n_cols, figsize=(n_cols * 5, n_rows * 5))
    axes = axes.flatten()

    hparam_1, hparam_2 = hyper_param_names
    accuracies = []
    f1_scores = []

    for i, exp_rs in enumerate(experiment_results):
        true_labels, pred_labels, pred_scores = exp_rs['true_labels'], exp_rs['pred_labels'], exp_rs['pred_scores']
        true_labels_bin, pred_labels_bin = label_binarize(true_labels, classes=range(0,10)), label_binarize(pred_labels, classes=range(0,10))

        accuracy, precision, recall, f1 = get_metrics(true_labels, pred_labels)
        accuracies.append(accuracy)
        f1_scores.append(f1)

        for j in range(0,10):
            # print(f"Class {j}: Prec:{precision[j]:.2f}, Recall:{recall[j]:.2f}, F_1 Score:{f1[j]:.2f}")
            precision_i, recall_i, _ = precision_recall_curve(true_labels_bin[:, j], np.array(pred_scores)[:, j])

            average_precision = average_precision_score(true_labels_bin[:, j], np.array(pred_scores)[:, j])
            axes[i].step(recall_i, precision_i, where="post", label=f"Class {j} AP={average_precision:.2f}")
            axes[i].set_title(f"PR-Curve {hparam_1}={exp_rs[hparam_1]}, {hparam_2}={exp_rs[hparam_2]}")
        axes[i].legend()
        axes[i].set_xlabel("Recall")
        axes[i].set_ylabel("Precision")

    # for j in range(i+1, 16):
    #     fig.delaxes(axes[j])

    plt.tight_layout()
    plt.show()
    return accuracies, f1_scores
```

3.0.4 ROC-AUC Curve

[116]:
```
def plot_rocauc(experiment_results, hyper_param_names, curve_type, n_rows=4, n_cols=4):
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 5, n_rows * 5))
    axes = axes.flatten()

    hparam_1, hparam_2 = hyper_param_names

    for i, exp_rs in enumerate(experiment_results):
        true_labels, pred_scores = exp_rs['true_labels'], exp_rs['pred_scores']
        true_labels_bin = label_binarize(true_labels, classes=range(0, 10))

        # All Classes' ROC curve & ROC Area Under Curve
        fpr = dict()
        tpr = dict()
        roc_auc = dict()

        for j in range(10):
            fpr[j], tpr[j], _ = roc_curve(true_labels_bin[:, j], np.array(pred_scores)[:, j])
            roc_auc[j] = auc(fpr[j], tpr[j])

        # Macro-Average ROC & ROC-AUC
        all_fpr = np.unique(np.concatenate([fpr[j] for j in range(10)]))
        mean_tpr = np.zeros_like(all_fpr)
        for j in range(10):
            mean_tpr += np.interp(all_fpr, fpr[j], tpr[j])
        mean_tpr /= 10
```

```
        fpr["macro"] = all_fpr
        tpr["macro"] = mean_tpr
        roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

        # Compute micro-average ROC curve and ROC area
        fpr["micro"], tpr["micro"], _ = roc_curve(true_labels_bin.ravel(), np.array(pred_scores).ravel())
        roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

        # Plot only Macro or Micro ROC curves
        if curve_type == "macro_micro":
            axes[i].plot(fpr["macro"], tpr["macro"], label=f"Macro (AUC={roc_auc['macro']:.2f})")
            axes[i].plot(fpr["micro"], tpr["micro"], label=f"Micro (AUC={roc_auc['micro']:.2f})")
        elif curve_type == "all":
            # Plot all ROC curves
            for j in range(10):
                axes[i].plot(fpr[j], tpr[j], label=f"Class {j} (AUC={roc_auc[j]:.2f})")

        axes[i].plot([0, 1], [0, 1], "k--")
        axes[i].set_xlabel("False Positive Rate")
        axes[i].set_ylabel("True Positive Rate")
        axes[i].set_title(f"ROC Curve {i+1}, {hparam_1}={exp_rs[hparam_1]}, {hparam_2}={exp_rs[hparam_2]}")
        axes[i].legend(loc='lower right')

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

3.1 Experiment 1: Optimizer

3.1.1 Initialization

Initialize variables and non-variables.

[72]:
```
exp1_hyperparams = {
    "num_epochs": 15,
    "lr": 0.001,
    "criterion": nn.CrossEntropyLoss(),
    "transform": A.Compose([
        A.Normalize(mean=norm_mean, std=norm_std),
        ToTensorV2()
    ])
}

exp1_models = [SmallVGG().to(device) for _ in range(0,6)]

candidate_optimizers = [
    optim.Adam(exp1_models[0].parameters(), lr=exp1_lr),
    optim.SGD(exp1_models[1].parameters(), lr=exp1_lr, momentum=0.9),
    optim.RMSprop(exp1_models[2].parameters(), lr=exp1_lr),
    optim.AdamW(exp1_models[3].parameters(), lr=exp1_lr, weight_decay=0.01),
    optim.Adagrad(exp1_models[4].parameters(), lr=exp1_lr),
    optim.SGD(exp1_models[5].parameters(), lr=exp1_lr, momentum=0.9, nesterov=True)]
```

3.1.2 Retrieve Data

Retrieve data from dataset.

[52]:
```
exp1_train_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"train_32x32.mat"), transform=exp1_hyperparams['transform'])
exp1_test_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"test_32x32.mat"), transform=exp1_hyperparams['transform'])
exp1_extra_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"extra_32x32.mat"), transform=exp1_hyperparams['transform'])

exp1_train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
exp1_test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
exp1_extra_loader = DataLoader(Subset(extra_dataset, indices=list(range(30000))), batch_size=128, shuffle=False)

print(f"Train Size:{train_dataset.__len__()}\nTest Size:{test_dataset.__len__()}\nExtra Size:{extra_dataset.__len__()}")
```

```
Train Size:73257
Test Size:26032
Extra Size:531131
```

3.1.3 Run Experiments

[75]:
```
def run_exp1(optimizers, models, hyper_params, train_loader, test_loader):
    experiments = []
    for i, [optimizer, exp1_model] in enumerate(zip(optimizers, models)):
        print(f"Experiment {i+1}. Running experiment on optimizer: {optimizer.__class__.__name__}")
        # exp1_model = copy.deepcopy(hyper_params['model'])
        criterion = hyper_params['criterion']
        num_epochs = hyper_params['num_epochs']
        train_losses, test_losses = train_and_evaluate(exp1_model, train_loader, test_loader, criterion, optimizer, num_epochs)
```

```
        experiments.append({
            "optimizer": optimizer.__class__.__name__,
            "others":"same",
            "train_losses": train_losses,
            "test_losses": test_losses,
            "model_state_dict": exp1_model.state_dict()
        })

        del exp1_model, criterion, optimizer
        torch.cuda.empty_cache()

    return experiments
```

[76]:
```
exp1 = run_exp1(candidate_optimizers, exp1_models, exp1_hyperparams, exp1_train_loader, exp1_test_loader)
time_str = str(time.time()).replace(".","")
torch.save(exp1, f"./models/exp1_{time_str}.pth")
```

```
100%|██████████| 573/573 [00:18<00:00, 30.31it/s]
Epoch[6/15], Train Loss:285.9288, Test Loss:283.8042
100%|██████████| 573/573 [00:18<00:00, 31.48it/s]
Epoch[7/15], Train Loss:285.9302, Test Loss:283.7845
100%|██████████| 573/573 [00:17<00:00, 31.97it/s]
Epoch[8/15], Train Loss:285.9285, Test Loss:283.7938
100%|██████████| 573/573 [00:17<00:00, 32.20it/s]
Epoch[9/15], Train Loss:285.9289, Test Loss:283.7943
```

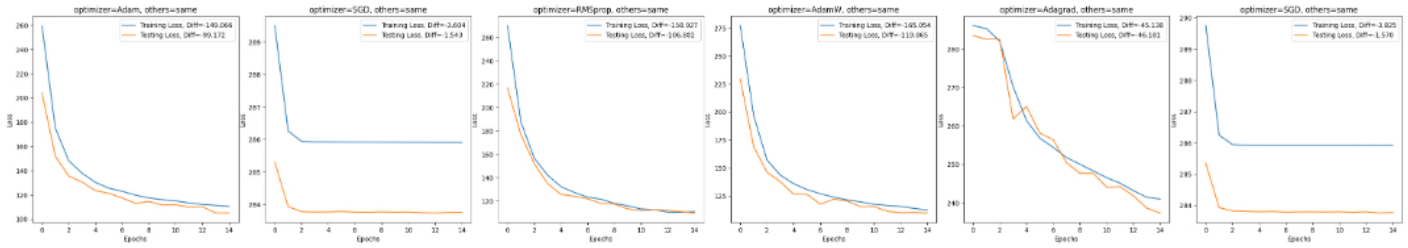
3.1.4 Load Experiments

Load Experiments and plot results.

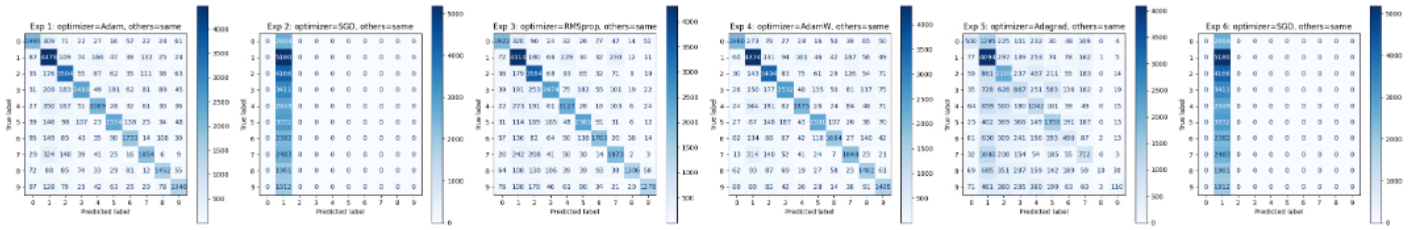
[88]:
```
exp1_loaded = torch.load("./models/exp1_1730104561164903.pth")
exp1_results = get_experiment_results(exp1_loaded, test_hyperparam_names=["optimizer", "others"], extra_loader=exp1_extra_loader)
```

```
0.4548557996749878, 0.11541266739368439, 0.10191038995981216, 0.09351465106010437, 0.07812810689210892, 0.07577980309724808, 0.06849437206983566, 0.
0627790242433548] ...
```

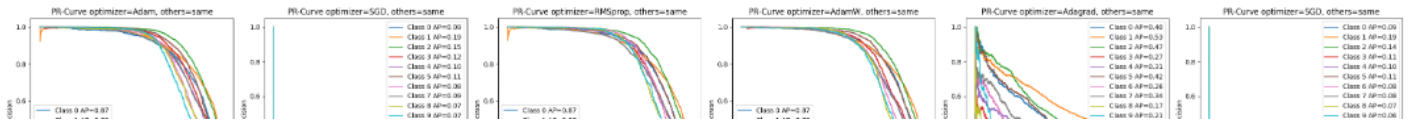
[98]:
```
plot_el(exp1_loaded, ["optimizer", "others"], n_rows=1, n_cols=6)
```

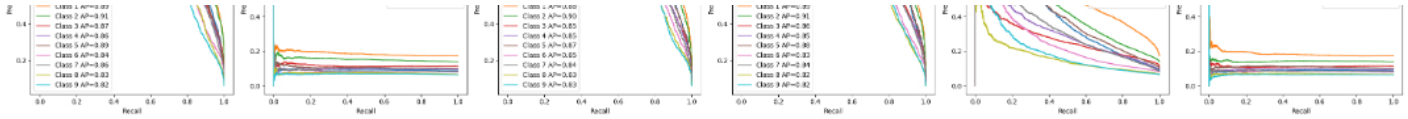


[106]:
```
plot_cm(exp1_results, ["optimizer", "others"], n_rows=1, n_cols=6)
```



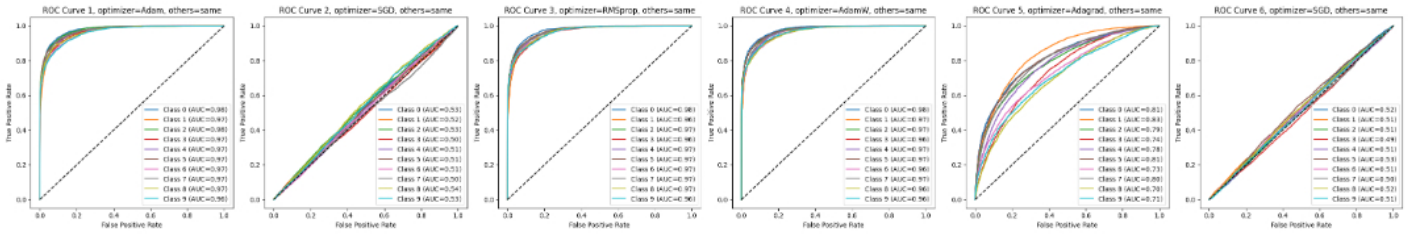
[133]:
```
exp1_accuracies, exp1_f1s = plot_pr(exp1_results, ["optimizer", "others"], n_rows=1, n_cols=6)
print(f"Accuracies:")
for acc in exp1_accuracies:
    print(f"{acc:.3f}", end=", ")
print("\n")

print(f"F1 Score Lists:")
for f1 in exp1_f1s:
    for val in f1:
        print(f"{val:.3f}", end=", ")
    print(f"Avg F1={np.mean(f1):.3f}")
```



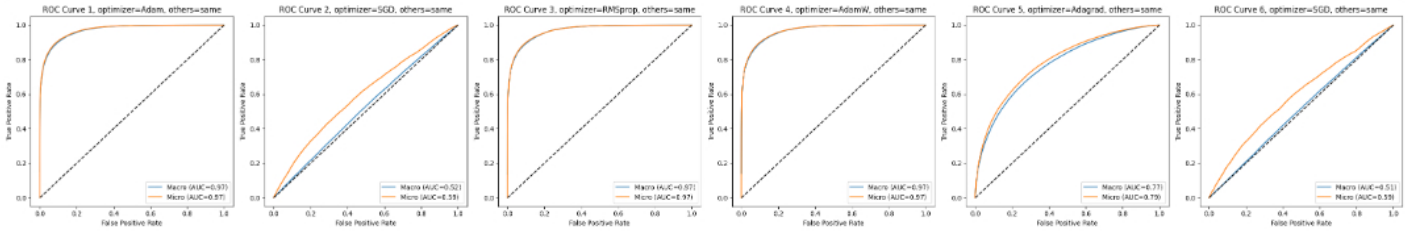


```
Accuracies:
0.776, 0.173, 0.768, 0.768, 0.376, 0.173,

F1 Score Lists:
0.781, 0.777, 0.807, 0.777, 0.763, 0.797, 0.754, 0.769, 0.751, 0.742, Avg F1=0.772
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
0.778, 0.773, 0.778, 0.758, 0.753, 0.786, 0.774, 0.747, 0.763, 0.751, Avg F1=0.766
0.792, 0.782, 0.798, 0.764, 0.743, 0.792, 0.752, 0.748, 0.721, 0.734, Avg F1=0.763
0.278, 0.503, 0.442, 0.280, 0.351, 0.429, 0.263, 0.332, 0.010, 0.103, Avg F1=0.299
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
```

[118]:
```
plot_rocauc(exp1_results, ["optimizer", "others"], curve_type="all", n_rows=1, n_cols=6)
```



[120]:
```
plot_rocauc(exp1_results, ["optimizer", "others"], curve_type="macro_micro", n_rows=1, n_cols=6)
```



3.2 Experiment 2: Epoch Number and Learning Rate

Initialize variables and non-variables.

[136]:
```
exp2_hyperparams = {
    "criterion": nn.CrossEntropyLoss(),
    "transform": A.Compose([
        A.Normalize(mean=norm_mean, std=norm_std),
        ToTensorV2()
    ]),
    "optimizer":optim.Adam,
}

candidate_epochs = [10, 15, 20, 25]
candidate_lr = [1e-2, 1e-3, 1e-4, 1e-5]
```

[138]:
```
torch.cuda.empty_cache()
# del exp1_train_dataset, exp1_test_dataset, exp1_extra_dataset, exp1_train_loader, exp1_test_loader, exp1_extra_loader

exp2_train_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"train_32x32.mat"), transform=exp2_hyperparams['transform'])
exp2_test_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"test_32x32.mat"), transform=exp2_hyperparams['transform'])
exp2_extra_dataset = SVHNDataset(mat_file=os.path.join(path_dataset,"extra_32x32.mat"), transform=exp2_hyperparams['transform'])

exp2_train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
exp2_test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
exp2_extra_loader = DataLoader(Subset(extra_dataset, indices=list(range(30000))), batch_size=128, shuffle=False)

print(f"Train Size:{train_dataset.__len__()}\nTest Size:{test_dataset.__len__()}\nExtra Size:{extra_dataset.__len__()}")
```

```
Train Size:73257
Test Size:26032
Extra Size:531131
```

[142]:
```
def run_exp2(epochs, lr_list, hyper_params, train_loader, test_loader):
    combinations = list(itertools.product(epochs, lr_list))
    experiments = []
    for i, combo in enumerate(combinations):
        num_epochs, lr = combo

        print(f"Running Exp {i+1}: num_epoch={num_epochs}, lr={lr}")
        this_model = SmallVGG().to(device)
        criterion = hyper_params['criterion']
        optimizer = hyper_params['optimizer'](this_model.parameters(), lr=lr)
        train_losses, test_losses = train_and_evaluate(this_model, train_loader, test_loader, criterion, optimizer, num_epochs)
```

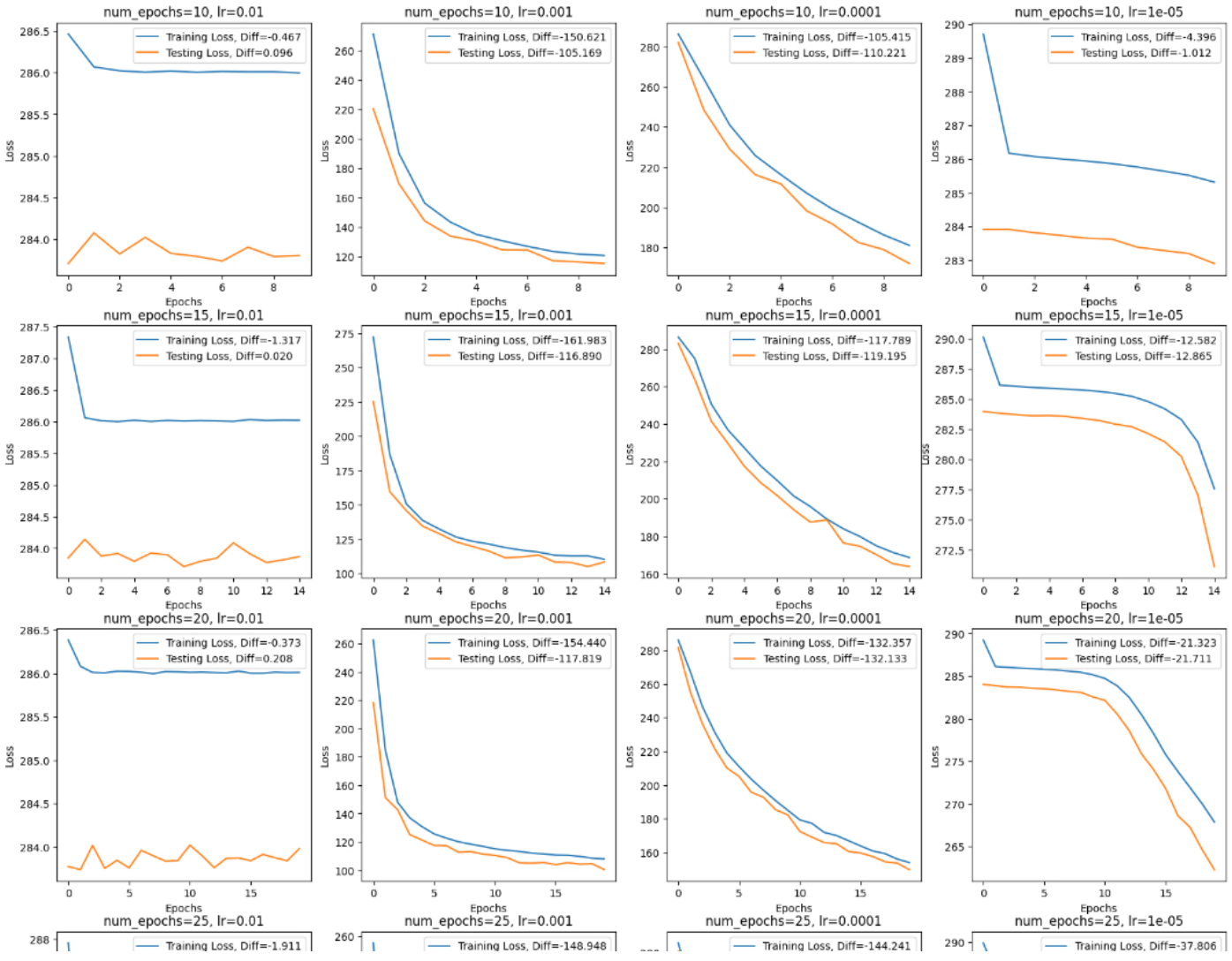
```
        experiments.append({
            "num_epochs": num_epochs,
            "lr": lr,
            "train_losses": train_losses,
            "test_losses": test_losses,
            "model_state_dict": this_model.state_dict()
        })

        del this_model, criterion, optimizer
    return experiments
```

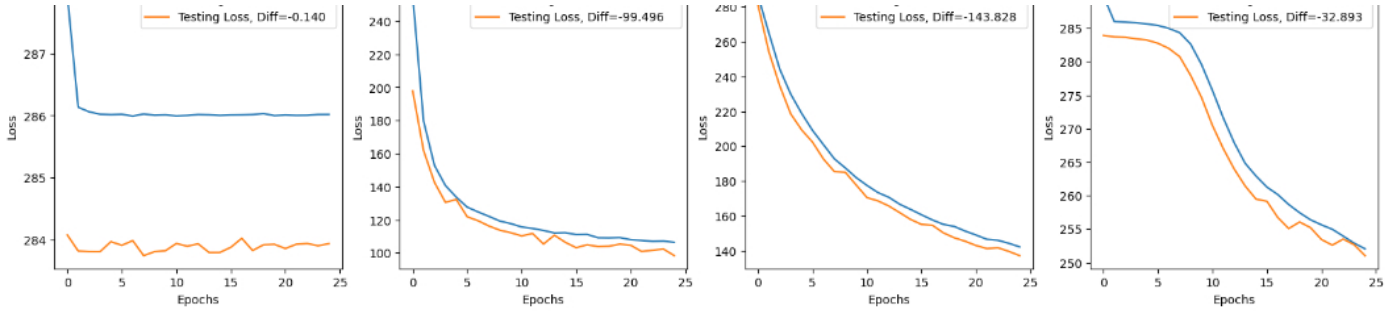
[146]:
```
exp2 = run_exp2(candidate_epochs, candidate_lr, exp2_hyperparams, exp2_train_loader, exp2_test_loader)
time_str = str(time.time()).replace(".","")
torch.save(exp2, f"./models/exp2_{time_str}.pth")
```

[152]:
```
exp2_loaded = torch.load("./models/exp2_17301145036294217.pth")
exp2_results = get_experiment_results(exp2_loaded, test_hyperparam_names=["num_epochs", "lr"], extra_loader=exp2_extra_loader)
```

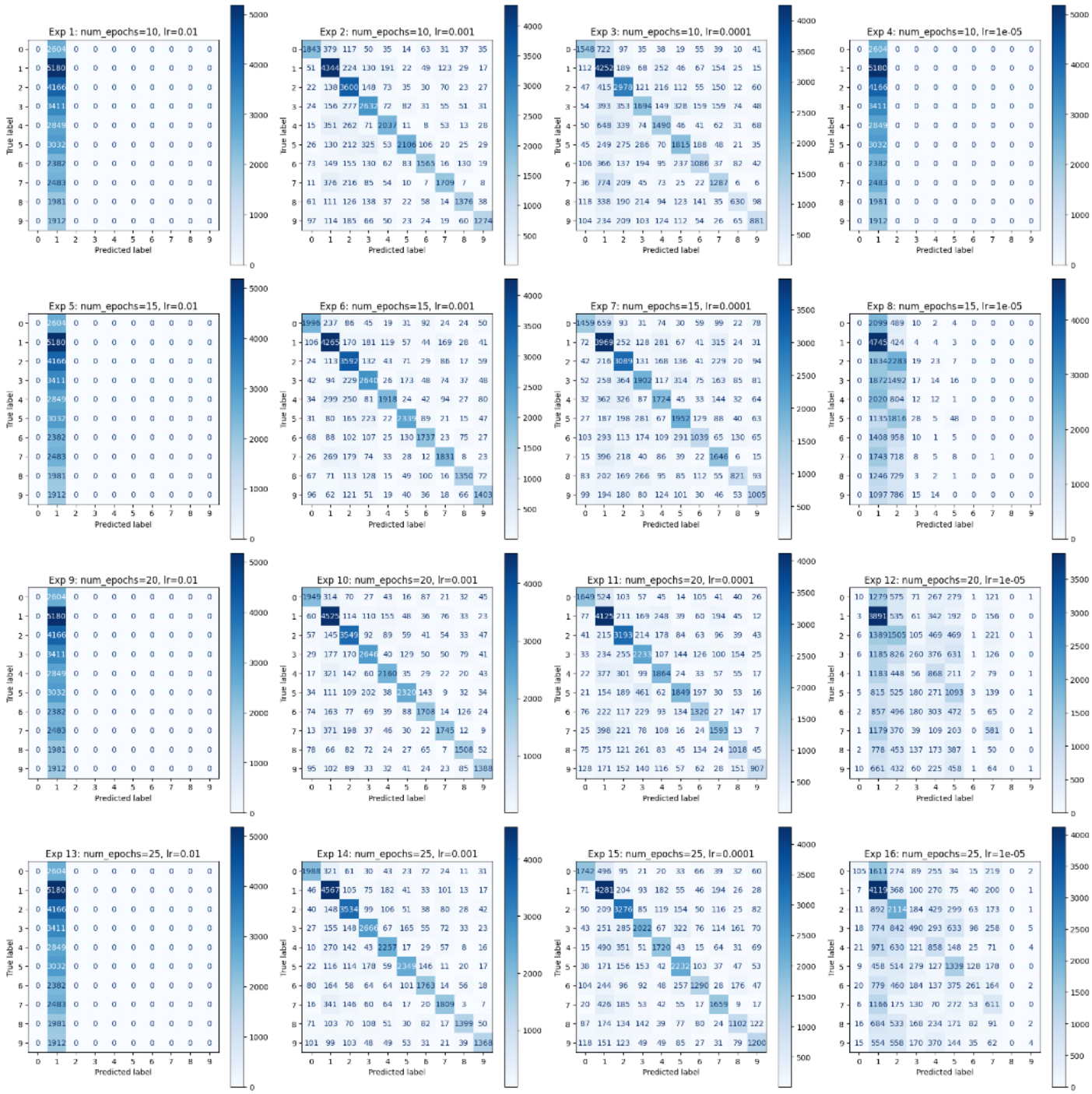
```
First 5 pred_scores:
[0.063282810151577, 0.19283294677734375, 0.14088118076324463, 0.11341533064842224, 0.10681350529193878, 0.0898299440741539, 0.08774298429489136,
0.07479149103164673, 0.06797333061695099, 0.062436480075120926] [0.063282810151577, 0.19283294677734375, 0.14088118076324463, 0.11341533064842224,
0.10681350529193878, 0.0898299440741539, 0.08774298429489136, 0.07479149103164673, 0.06797333061695099, 0.062436480075120926] [0.063282810151577,
0.19283294677734375, 0.14088118076324463, 0.11341533064842224, 0.10681350529193878, 0.0898299440741539, 0.08774298429489136, 0.07479149103164673,
0.06797333061695099, 0.062436480075120926] [0.063282810151577, 0.19283294677734375, 0.14088118076324463, 0.11341533064842224, 0.10681350529193878,
0.0898299440741539, 0.08774298429489136, 0.07479149103164673, 0.06797333061695099, 0.062436480075120926] [0.063282810151577, 0.19283294677734375,
0.14088118076324463, 0.11341533064842224, 0.10681350529193878, 0.0898299440741539, 0.08774298429489136, 0.07479149103164673, 0.06797333061695099,
0.062436480075120926] ...

100%|██████████| 235/235 [00:06<00:00, 34.31it/s]
First 10 true labels:
4 7 8 7 1 1 7 4 3 0 ...

First 10 pred labels:
4 7 8 7 7 7 2 7 3 0 ...
```

[196]:
```
plot_el(exp2_loaded, ["num_epochs", "lr"], n_rows=4, n_cols=4)
```



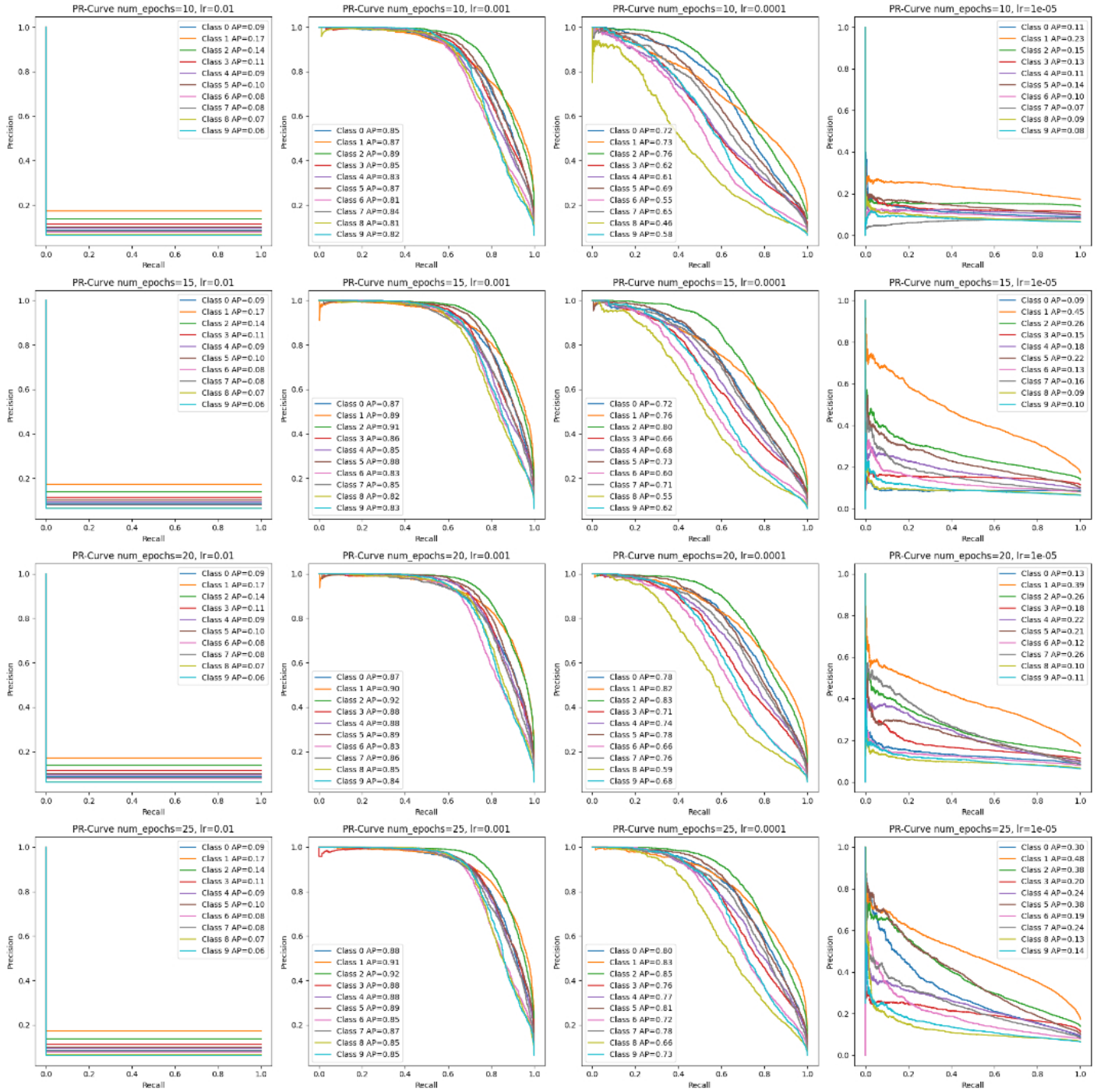


[162]:
```
plot_cm(exp2_results, ["num_epochs", "lr"], n_rows=4, n_cols=4)
```



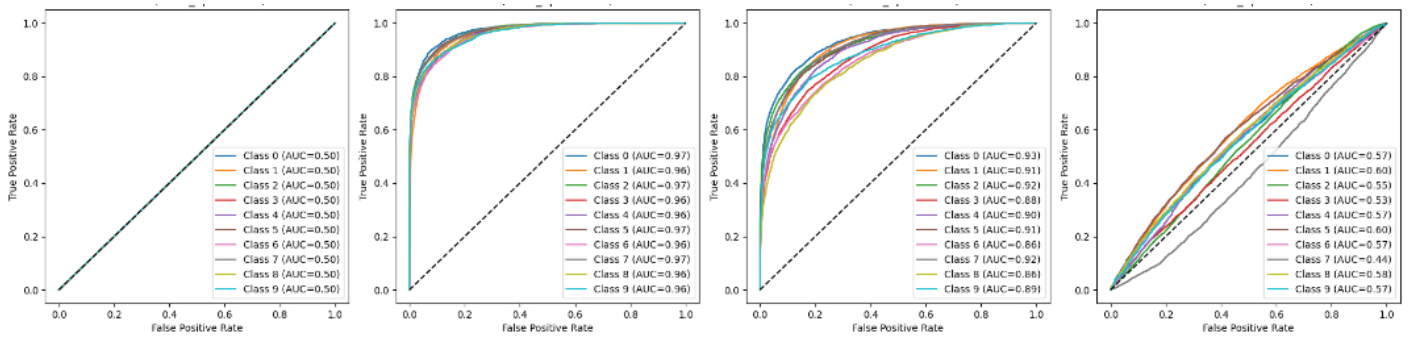
[165]:
```
exp1_accuracies, exp1_f1s = plot_pr(exp2_results, ["num_epochs", "lr"], n_rows=4, n_cols=4)
print(f"Accuracies:")
for acc in exp1_accuracies:
    print(f"{acc:.3f}", end=", ")
print("\n")

print(f"F1 Score Lists:")
for f1 in exp1_f1s:
```

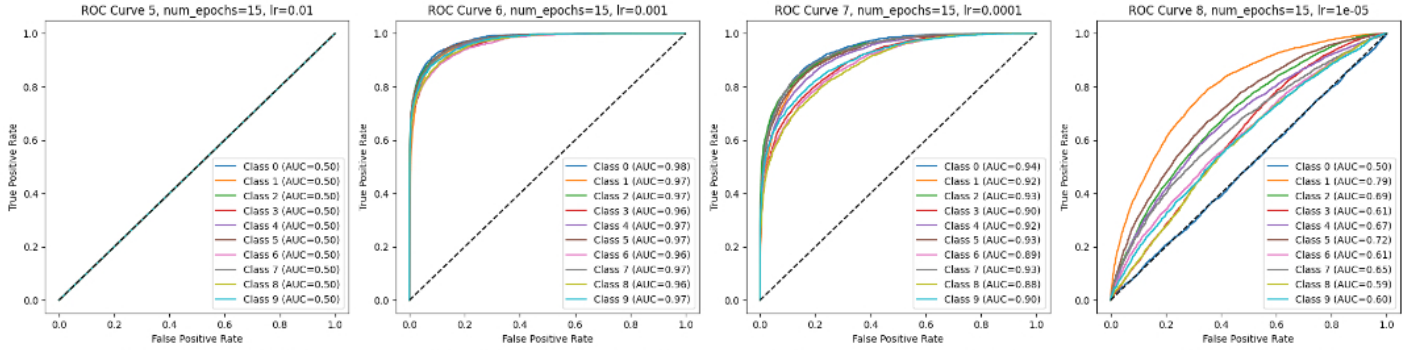
```
    for val in f1:
        print(f"{val:.3f}", end=", ")
    print(f"Avg F1={np.mean(f1):.3f}")
```

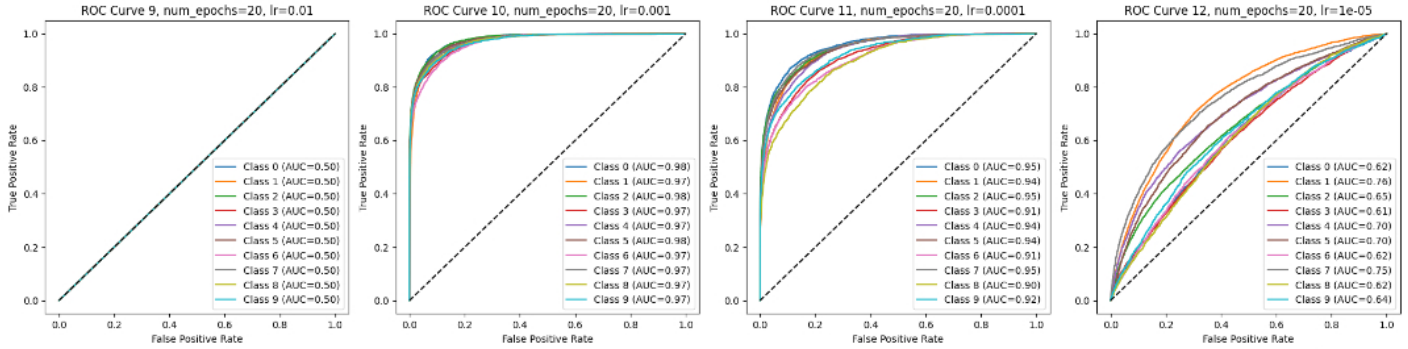


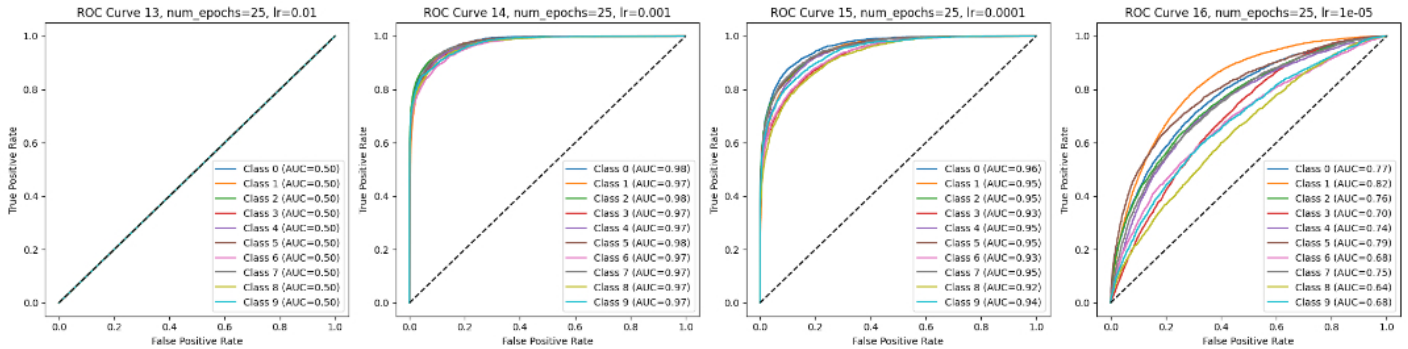

```
Accuracies:
0.173, 0.750, 0.589, 0.173, 0.173, 0.769, 0.620, 0.237, 0.173, 0.783, 0.658, 0.274, 0.173, 0.790, 0.684, 0.330,

F1 Score Lists:
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
0.764, 0.760, 0.755, 0.733, 0.739, 0.774, 0.724, 0.744, 0.737, 0.745, Avg F1=0.748
0.642, 0.627, 0.651, 0.543, 0.547, 0.616, 0.511, 0.575, 0.429, 0.550, Avg F1=0.569
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
0.784, 0.793, 0.783, 0.747, 0.754, 0.783, 0.753, 0.757, 0.744, 0.746, Avg F1=0.764
0.636, 0.666, 0.674, 0.582, 0.606, 0.641, 0.524, 0.617, 0.511, 0.574, Avg F1=0.603
0.000, 0.389, 0.311, 0.010, 0.008, 0.031, 0.000, 0.001, 0.000, 0.000, Avg F1=0.075
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
0.778, 0.789, 0.810, 0.783, 0.783, 0.797, 0.745, 0.775, 0.765, 0.767, Avg F1=0.779
0.694, 0.701, 0.707, 0.607, 0.648, 0.680, 0.586, 0.682, 0.551, 0.599, Avg F1=0.646
0.008, 0.423, 0.291, 0.114, 0.278, 0.294, 0.004, 0.284, 0.000, 0.001, Avg F1=0.170
0.000, 0.294, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, Avg F1=0.029
0.794, 0.797, 0.817, 0.786, 0.779, 0.799, 0.758, 0.772, 0.779, 0.781, Avg F1=0.786
0.712, 0.709, 0.722, 0.655, 0.664, 0.704, 0.621, 0.693, 0.601, 0.656, Avg F1=0.674
0.074, 0.479, 0.398, 0.184, 0.291, 0.411, 0.164, 0.271, 0.000, 0.004, Avg F1=0.228
```

[164]:
```
plot_rocauc(exp2_results, ["num_epochs", "lr"], curve_type="all", n_rows=4, n_cols=4)
```

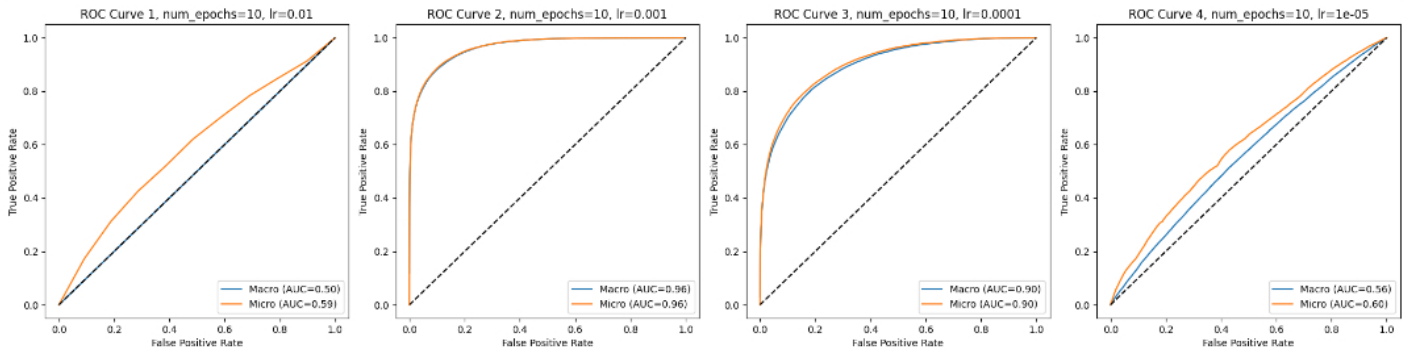


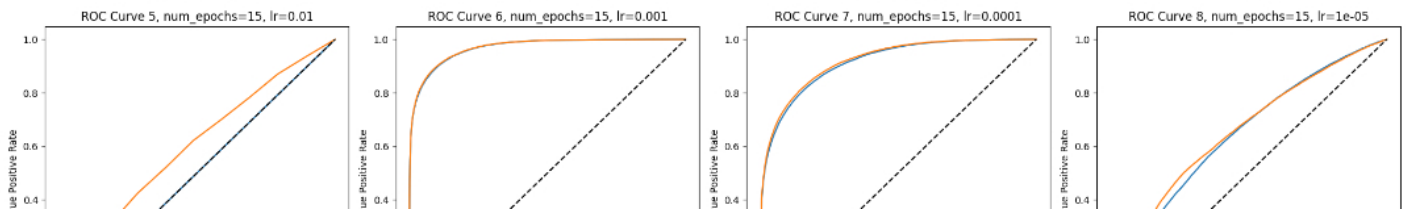


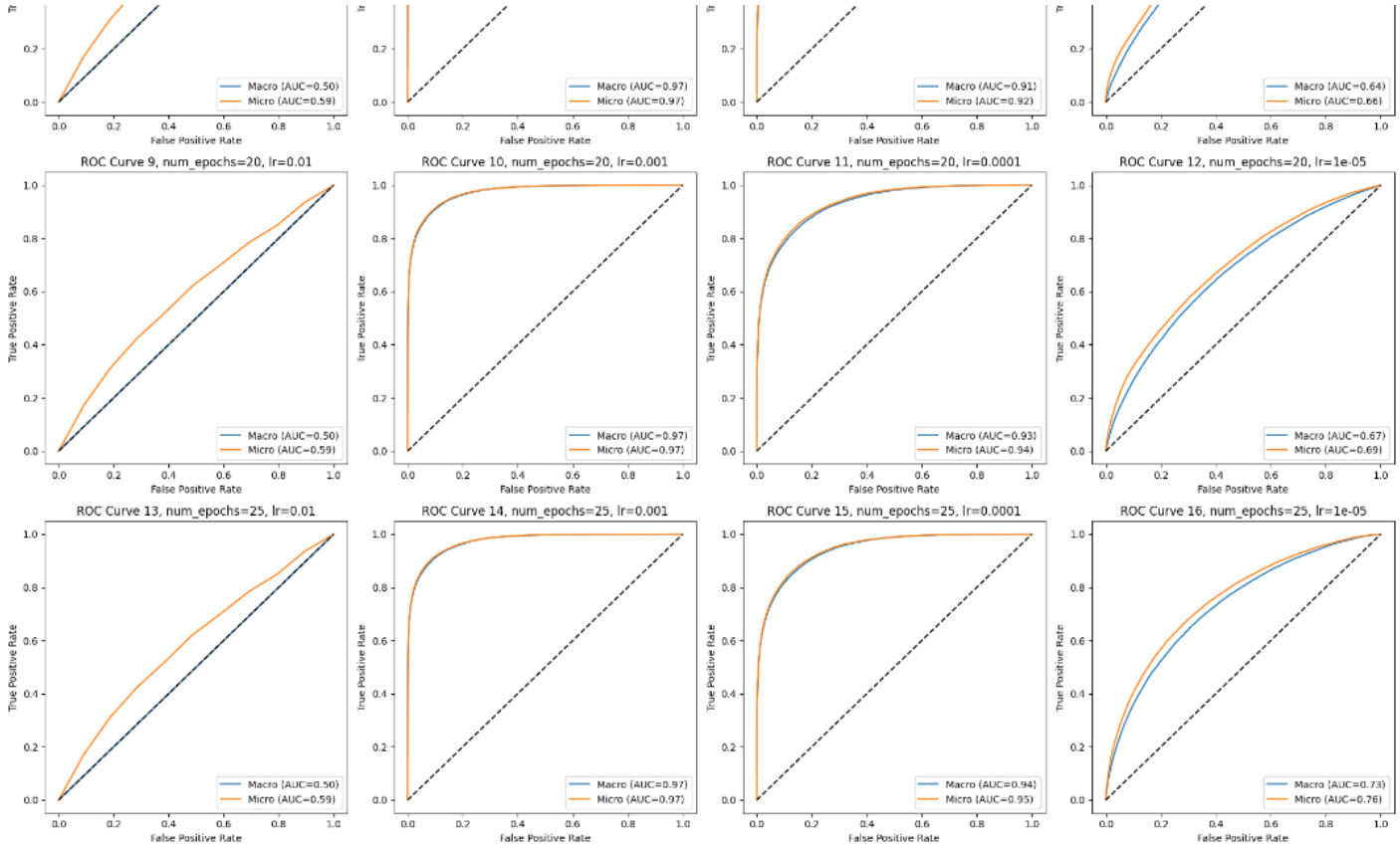




[166]:
```
plot_rocauc(exp2_results, ["num_epochs", "lr"], curve_type="macro_micro", n_rows=4, n_cols=4)
```







3.3 Experiment 3: Transform

[167]:
```python
exp3_1_hyperparams = {
    "num_epochs": 15,
    "lr": 1e-3,
    "criterion": nn.CrossEntropyLoss(),
    "optimizer": optim.Adam
}

# Group 1
candidate_angles = [15, 30, 45, 60]
candidate_crops = [0.08, 0.24, 0.40, 0.60] # Left Boundary

# Group 2
candidate_ratios = [0.25, 0.42, 0.58, 0.75]
candidate_channel_biases = [0, 32, 64, 128]
```

[177]:
```python
exp3_train_dataset = SVHNDataset(mat_file=os.path.join(path_dataset, "train_32x32.mat"))
exp3_test_dataset = SVHNDataset(mat_file=os.path.join(path_dataset, "test_32x32.mat"))
exp3_extra_dataset = SVHNDataset(mat_file=os.path.join(path_dataset, "extra_32x32.mat"))

print(f"Train Size:{train_dataset.__len__()}\nTest Size:{test_dataset.__len__()}\nExtra Size:{extra_dataset.__len__()}")
```

```
Train Size:73257
Test Size:26032
Extra Size:531131
```

[189]:
```python
def run_exp3_1(angles, crops, hyper_params, train_dataset, test_dataset):
    combinations = list(itertools.product(angles, crops))
    experiments = []
    for i, combo in enumerate(combinations):
        angle, crop = combo

        print(f"Running Exp {i+1}: angles={angle}, crop={crop}")
        this_model = SmallVGG().to(device)
        num_epochs = hyper_params['num_epochs']
        lr = hyper_params['lr']
        criterion = hyper_params['criterion']
        optimizer = hyper_params['optimizer'](this_model.parameters(), lr=lr)

        this_transform = A.Compose([
            A.RandomResizedCrop(32, 32, scale=(crop, 1.0)),
            A.Rotate(limit=angle),
            A.Normalize(mean=norm_mean, std=norm_std),
            ToTensorV2()
        ])
```

```
        # Generate Dataset
        print(f"Exp {i+1}: Generating dataset from transform")
        train_dataset.transform = this_transform
        test_dataset.transform = this_transform

        train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
        test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

        # Train Model
        train_losses, test_losses = train_and_evaluate(this_model,
                                                       train_loader,
                                                       test_loader,
                                                       criterion,
                                                       optimizer,
                                                       num_epochs)

        experiments.append({
            "angle": angle,
            "crop": crop,
            "train_losses": train_losses,
            "test_losses": test_losses,
            "model_state_dict": this_model.state_dict()
        })

        del this_model, criterion, optimizer
        del train_loader, test_loader
        torch.cuda.empty_cache()

    return experiments
```

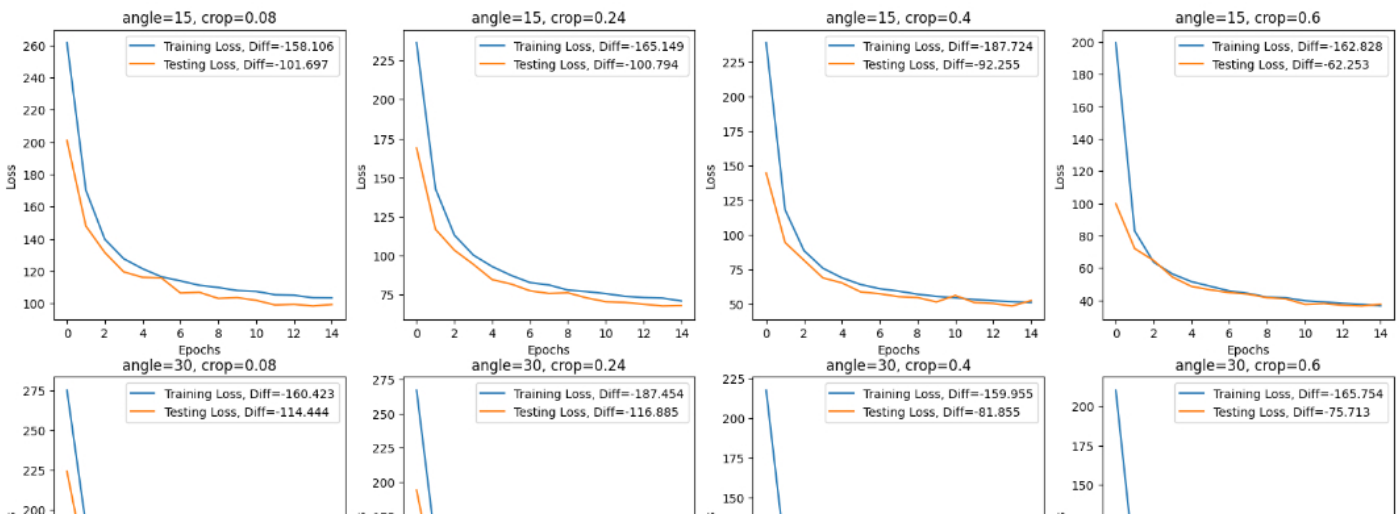
[190]:
```
exp3_1 = run_exp3_1(candidate_angles, candidate_crops, exp3_1_hyperparams, exp3_train_dataset, exp3_test_dataset)
time_str = str(time.time()).replace(".","")
torch.save(exp3_1, f"./models/exp3_1_{time_str}.pth")
```

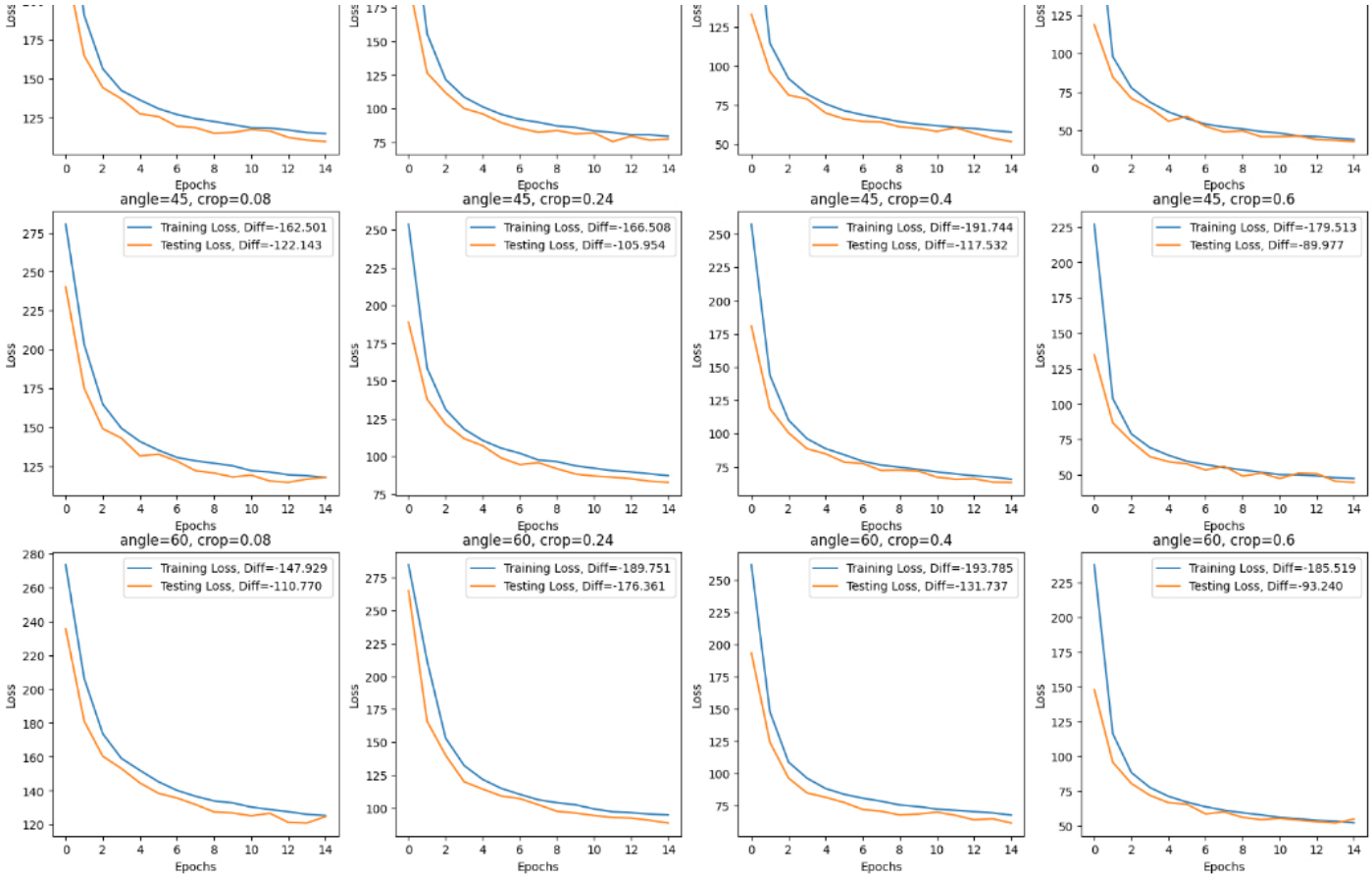
```
Epoch[7/15], Train Loss:63.8747, Test Loss:58.7209
100%|| 573/573 [00:16<00:00, 33.87it/s]
Epoch[8/15], Train Loss:61.3178, Test Loss:60.1422
100%|| 573/573 [00:15<00:00, 36.52it/s]
Epoch[9/15], Train Loss:59.5554, Test Loss:56.2422
100%|| 573/573 [00:16<00:00, 33.78it/s]
Epoch[10/15], Train Loss:58.0028, Test Loss:54.6090
100%|| 573/573 [00:16<00:00, 34.51it/s]
Epoch[11/15], Train Loss:56.2025, Test Loss:55.5614
100%|| 573/573 [00:17<00:00, 33.42it/s]
Epoch[12/15], Train Loss:55.2669, Test Loss:54.2610
100%|| 573/573 [00:16<00:00, 33.77it/s]
Epoch[13/15], Train Loss:54.0181, Test Loss:53.1277
```

[206]:
```
exp3_1_loaded = torch.load("./models/exp3_1_1730131987195526.pth")
exp3_1_results = get_experiment_results(exp3_1_loaded, test_hyperparam_names=["angle", "crop"], extra_loader=exp2_extra_loader)
```

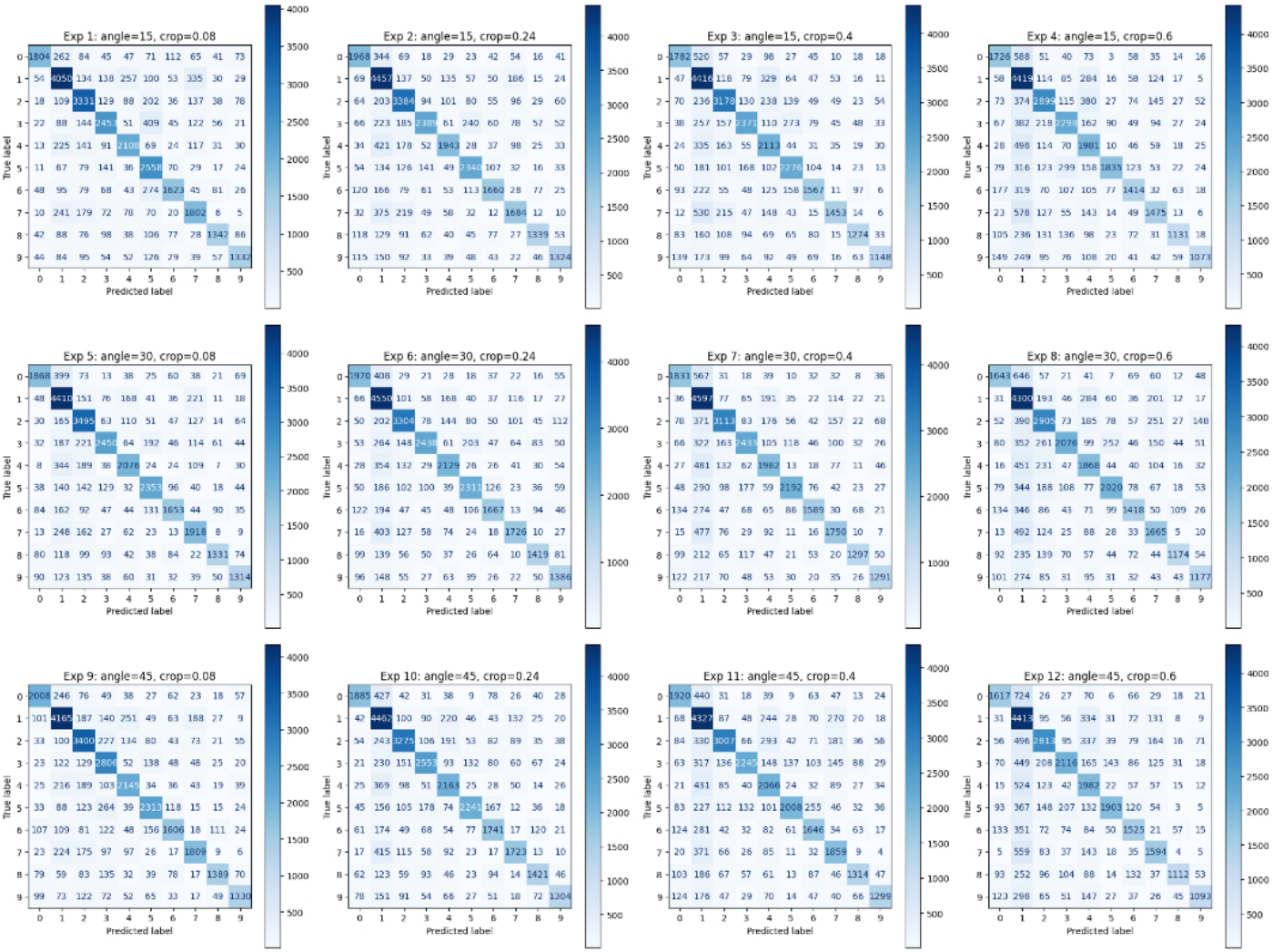
```
First 5 pred_scores:
[0.0001420140906702727, 0.00713851023465395, 0.003791375085711479, 0.005049004219472408, 0.9780939221382141, 0.002496036933735013, 0.0004309658834
245056, 0.0009765718132257462, 0.0005876566283404827, 0.0012939583975821733] [0.025841131806373596, 0.6686941385269165, 0.004654822405427694, 0.00
09682995732873678, 0.04681430384516716, 0.00045565434265881777, 0.006434406153857708, 0.23315496742725372, 0.0014618063578382134, 0.01152052730321
```

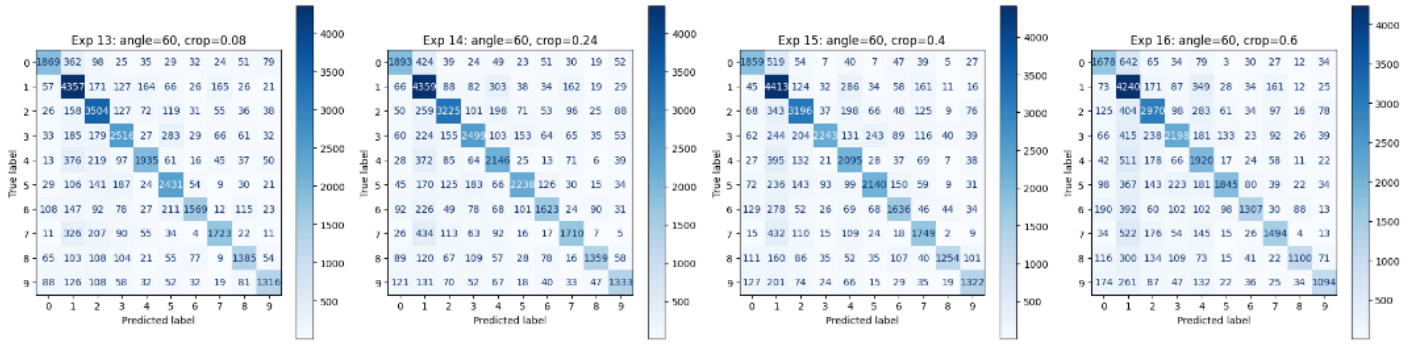
[207]:
```
plot_el(exp3_1_loaded, ["angle", "crop"], n_rows=4, n_cols=4)
```



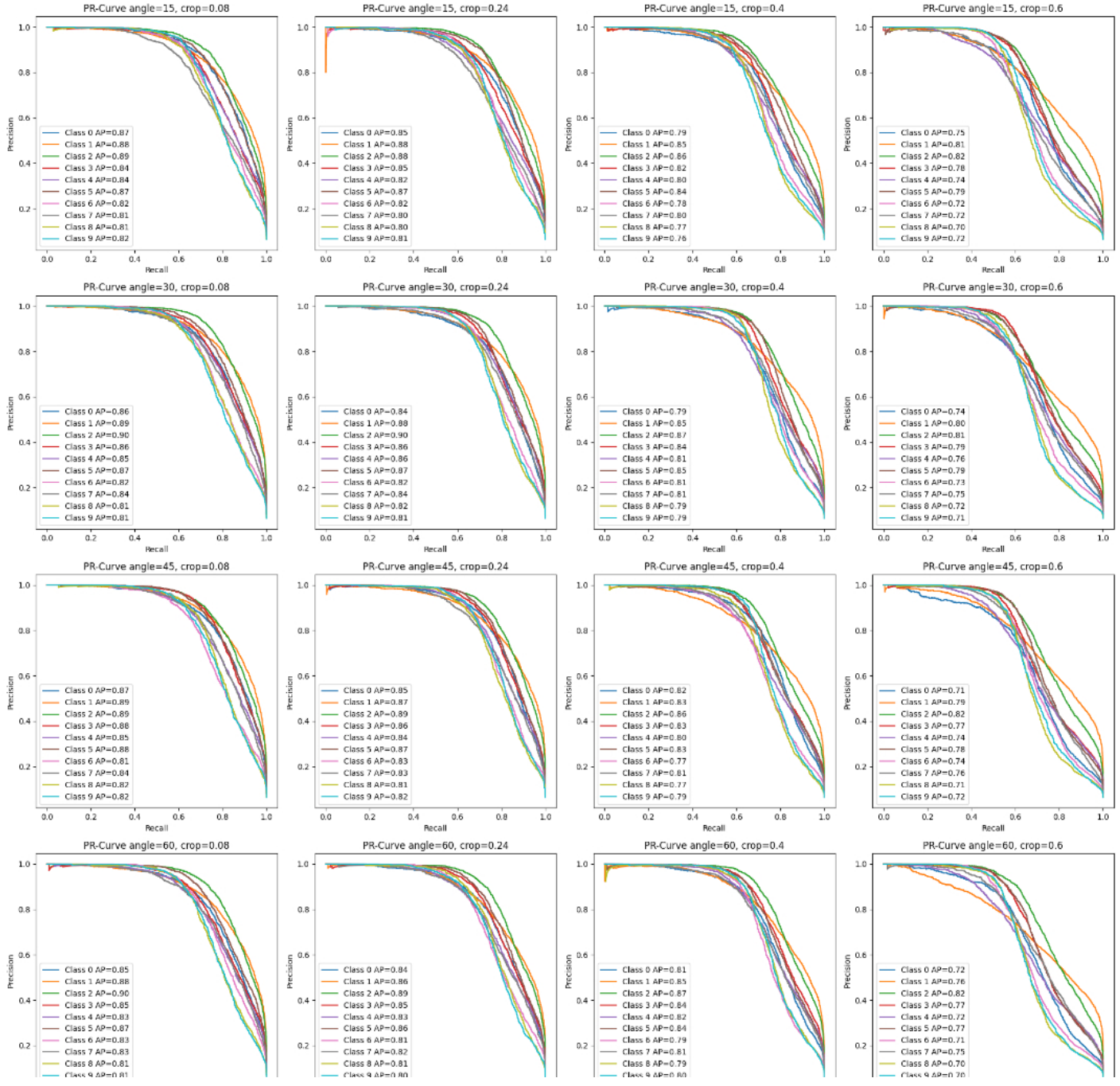


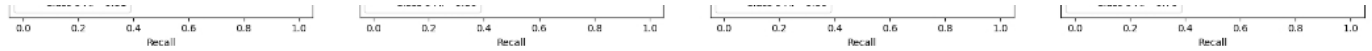
[208]:
```
plot_cm(exp3_1_results, ["angle", "crop"], n_rows=4, n_cols=4)
```





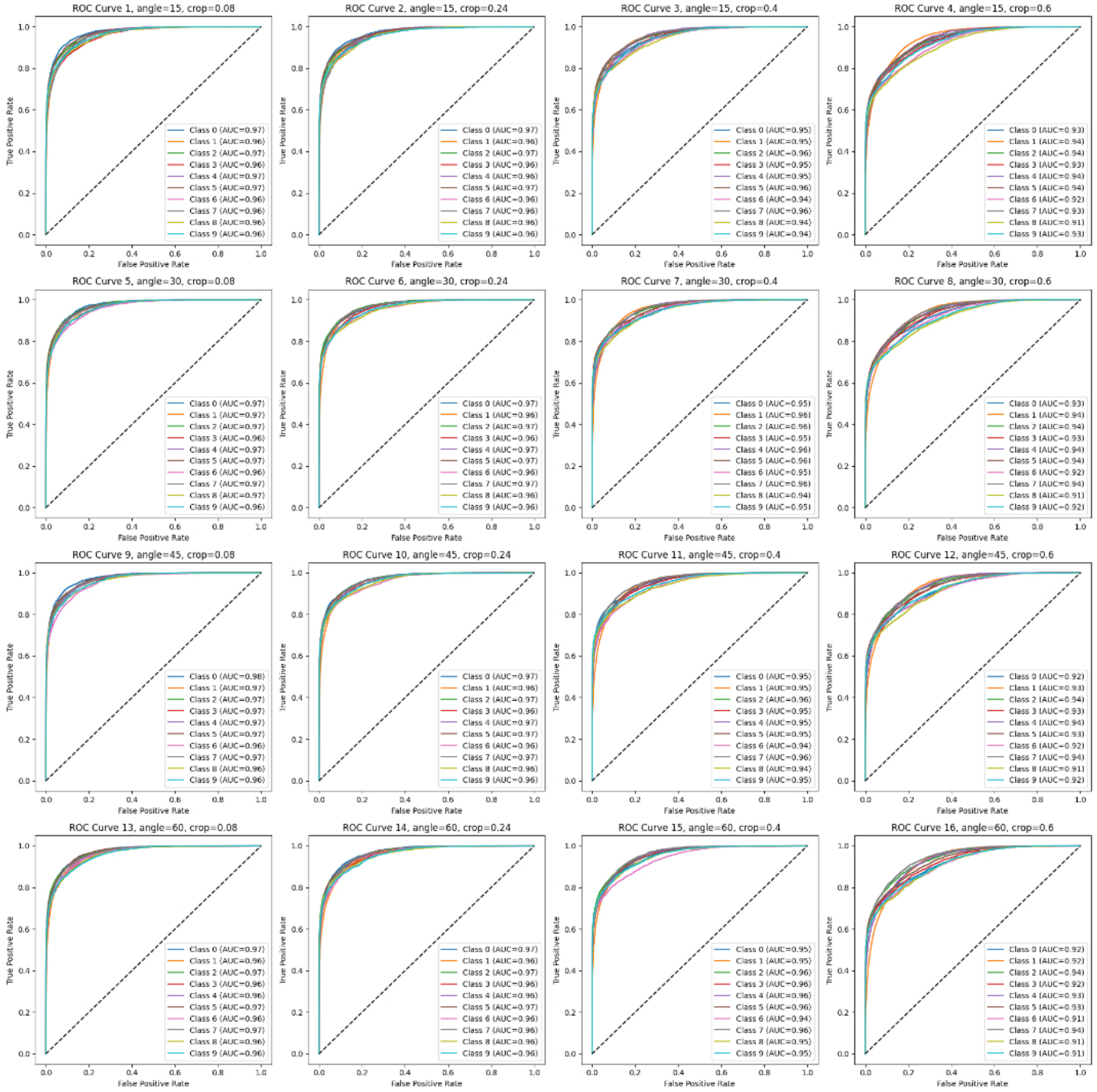
[209]:
```
exp1_accuracies, exp1_f1s = plot_pr(exp3_1_results, ["angle", "crop"], n_rows=4, n_cols=4)
print(f"Accuracies:")
for acc in exp1_accuracies:
    print(f"{acc:.3f}", end=", ")
print("\n")

print(f"F1 Score Lists:")
for f1 in exp1_f1s:
    for val in f1:
        print(f"{val:.3f}", end=", ")
    print(f"Avg F1={np.mean(f1):.3f}")
```



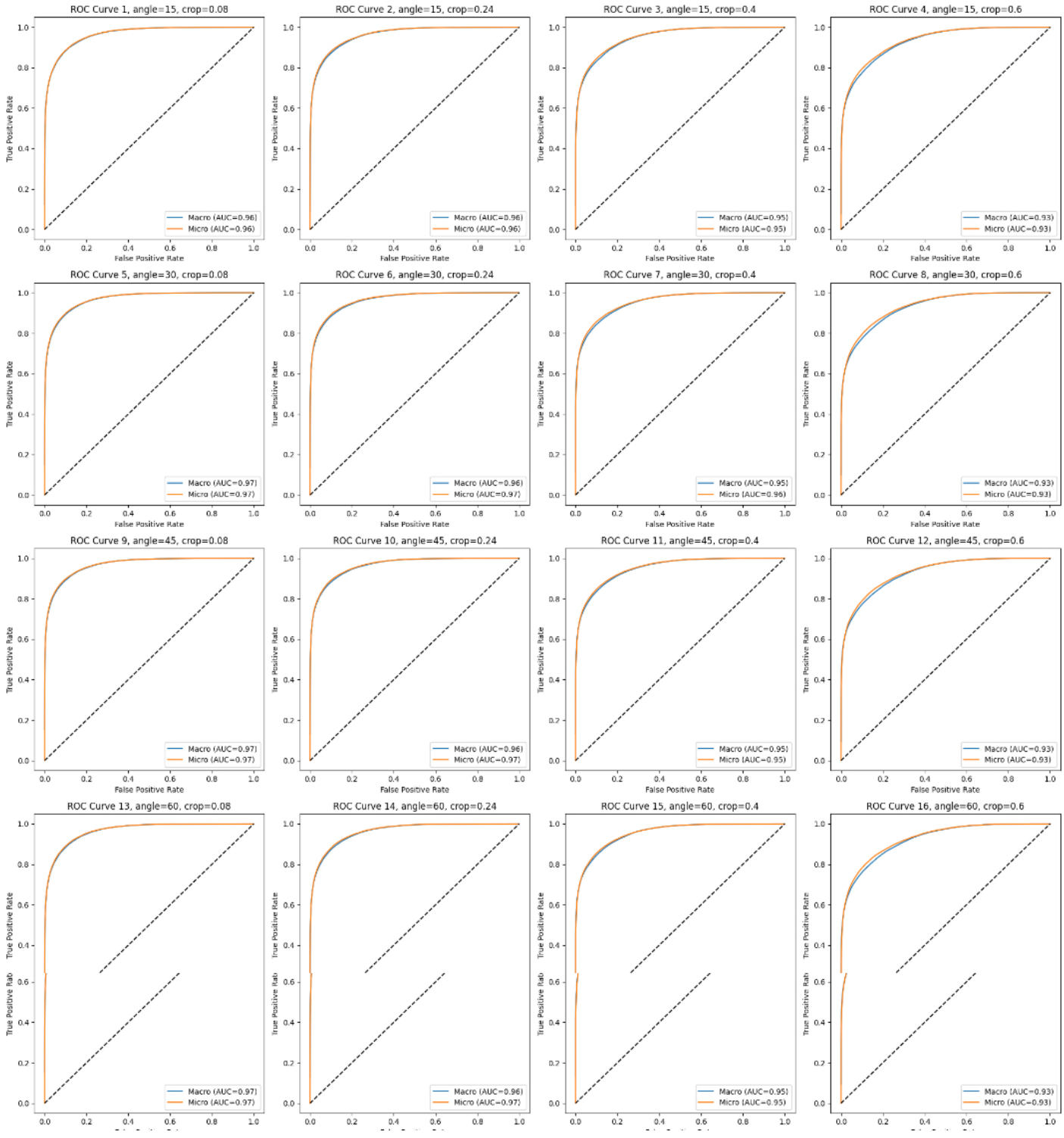



```
Accuracies:
0.747, 0.750, 0.719, 0.675, 0.762, 0.763, 0.736, 0.675, 0.766, 0.759, 0.723, 0.672, 0.753, 0.746, 0.730, 0.662,

F1 Score Lists:
0.773, 0.772, 0.783, 0.732, 0.747, 0.729, 0.726, 0.693, 0.729, 0.737, Avg F1=0.742
0.751, 0.757, 0.776, 0.751, 0.725, 0.775, 0.734, 0.703, 0.741, 0.742, Avg F1=0.746
0.721, 0.723, 0.755, 0.730, 0.674, 0.738, 0.701, 0.695, 0.713, 0.703, Avg F1=0.715
0.678, 0.673, 0.715, 0.687, 0.625, 0.713, 0.648, 0.645, 0.671, 0.676, Avg F1=0.673
0.763, 0.769, 0.783, 0.767, 0.749, 0.792, 0.739, 0.744, 0.741, 0.727, Avg F1=0.758
0.764, 0.757, 0.799, 0.772, 0.755, 0.783, 0.744, 0.747, 0.751, 0.728, Avg F1=0.760
0.724, 0.708, 0.775, 0.747, 0.701, 0.782, 0.740, 0.723, 0.741, 0.737, Avg F1=0.738
0.678, 0.661, 0.689, 0.698, 0.654, 0.709, 0.665, 0.651, 0.682, 0.667, Avg F1=0.675
0.782, 0.787, 0.779, 0.756, 0.748, 0.776, 0.716, 0.764, 0.758, 0.750, Avg F1=0.762
0.770, 0.748, 0.794, 0.763, 0.735, 0.788, 0.731, 0.745, 0.743, 0.757, Avg F1=0.757
0.736, 0.706, 0.767, 0.736, 0.684, 0.747, 0.688, 0.710, 0.720, 0.747, Avg F1=0.724
0.668, 0.648, 0.713, 0.680, 0.626, 0.720, 0.664, 0.675, 0.676, 0.680, Avg F1=0.675
0.762, 0.763, 0.779, 0.738, 0.738, 0.763, 0.738, 0.748, 0.724, 0.740, Avg F1=0.749
0.746, 0.733, 0.788, 0.750, 0.716, 0.779, 0.724, 0.725, 0.754, 0.734, Avg F1=0.745
0.726, 0.712, 0.766, 0.755, 0.699, 0.752, 0.711, 0.711, 0.742, 0.733, Avg F1=0.731
0.645, 0.641, 0.708, 0.684, 0.610, 0.700, 0.651, 0.660, 0.665, 0.656, Avg F1=0.662
```

[210]:
```
plot_rocauc(exp3_1_results, ["angle", "crop"], curve_type="all", n_rows=4, n_cols=4)
```



[213]:
```
plot_rocauc(exp3_1_results, ["angle", "crop"], curve_type="macro micro", n_rows=4, n_cols=4)
```


ROC Curve 1, angle=15, crop=0.08
Macro (AUC=0.96)
Micro (AUC=0.96)
ROC Curve 2, angle=15, crop=0.24
Macro (AUC=0.96)
Micro (AUC=0.96)
ROC Curve 3, angle=15, crop=0.4
Macro (AUC=0.95)
Micro (AUC=0.95)
ROC Curve 4, angle=15, crop=0.6
Macro (AUC=0.93)
Micro (AUC=0.93)
ROC Curve 5, angle=30, crop=0.08
Macro (AUC=0.97)
Micro (AUC=0.97)
ROC Curve 6, angle=30, crop=0.24
Macro (AUC=0.96)
Micro (AUC=0.97)
ROC Curve 7, angle=30, crop=0.4
Macro (AUC=0.95)
Micro (AUC=0.96)
ROC Curve 8, angle=30, crop=0.6
Macro (AUC=0.93)
Micro (AUC=0.93)
ROC Curve 9, angle=45, crop=0.08
Macro (AUC=0.97)
Micro (AUC=0.97)
ROC Curve 10, angle=45, crop=0.24
Macro (AUC=0.96)
Micro (AUC=0.97)
ROC Curve 11, angle=45, crop=0.4
Macro (AUC=0.95)
Micro (AUC=0.95)
ROC Curve 12, angle=45, crop=0.6
Macro (AUC=0.93)
Micro (AUC=0.93)
ROC Curve 13, angle=60, crop=0.08
Macro (AUC=0.97)
Micro (AUC=0.97)
ROC Curve 14, angle=60, crop=0.24
Macro (AUC=0.96)
Micro (AUC=0.97)
ROC Curve 15, angle=60, crop=0.4
Macro (AUC=0.95)
Micro (AUC=0.95)
ROC Curve 16, angle=60, crop=0.6
Macro (AUC=0.93)
Micro (AUC=0.93)
True Positive Rate
False Positive Rate