In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn.functional as F
from torchsummary import summary
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn import decomposition
from sklearn import manifold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import numpy as np
from tqdm import tqdm
import random
```

In [ ]:
```
SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

In [ ]:
```
class BasicBlock(nn.Module):

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(planes)
            )
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

In [ ]:
```
class ResNetSmall(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNetSmall, self).__init__()
        self.in_planes = 16

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.layer1 = self._make_layer(block, 16, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 32, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 64, num_blocks[2], stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(64, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        downsample = None
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x


device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
layers=[1, 1, 1, 1]
modelsmall = ResNetSmall(BasicBlock, layers).to(device)
summary(modelsmall, input_size=(3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 32, 32]             432
       BatchNorm2d-2           [-1, 16, 32, 32]              32
            Conv2d-3           [-1, 16, 32, 32]           2,304
       BatchNorm2d-4           [-1, 16, 32, 32]              32
            Conv2d-5           [-1, 16, 32, 32]           2,304
       BatchNorm2d-6           [-1, 16, 32, 32]              32
        BasicBlock-7           [-1, 16, 32, 32]               0
            Conv2d-8           [-1, 32, 16, 16]           4,608
       BatchNorm2d-9           [-1, 32, 16, 16]              64
           Conv2d-10           [-1, 32, 16, 16]           9,216
      BatchNorm2d-11           [-1, 32, 16, 16]              64
           Conv2d-12           [-1, 32, 16, 16]             512
      BatchNorm2d-13           [-1, 32, 16, 16]              64
       BasicBlock-14           [-1, 32, 16, 16]               0
           Conv2d-15             [-1, 64, 8, 8]          18,432
      BatchNorm2d-16             [-1, 64, 8, 8]             128
           Conv2d-17             [-1, 64, 8, 8]          36,864
      BatchNorm2d-18             [-1, 64, 8, 8]             128
           Conv2d-19             [-1, 64, 8, 8]           2,048
      BatchNorm2d-20             [-1, 64, 8, 8]             128
       BasicBlock-21             [-1, 64, 8, 8]               0
AdaptiveAvgPool2d-22             [-1, 64, 1, 1]               0
           Linear-23                   [-1, 10]             650
================================================================
Total params: 78,042
Trainable params: 78,042
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.53
Params size (MB): 0.30
Estimated Total Size (MB): 1.84
----------------------------------------------------------------
```

In [ ]:
```
class ResNetMedium(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
```

```
        super(ResNetMedium, self).__init__()
        self.in_planes = 32

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(32)
        self.layer1 = self._make_layer(block, 32, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 64, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 128, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(256, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        downsample = None
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)


    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x

modelmedium = ResNetMedium(BasicBlock, layers).to(device)
summary(modelmedium, input_size=(3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             864
       BatchNorm2d-2           [-1, 32, 32, 32]              64
            Conv2d-3           [-1, 32, 32, 32]           9,216
       BatchNorm2d-4           [-1, 32, 32, 32]              64
            Conv2d-5           [-1, 32, 32, 32]           9,216
       BatchNorm2d-6           [-1, 32, 32, 32]              64
        BasicBlock-7           [-1, 32, 32, 32]               0
            Conv2d-8           [-1, 64, 16, 16]          18,432
       BatchNorm2d-9           [-1, 64, 16, 16]             128
           Conv2d-10           [-1, 64, 16, 16]          36,864
      BatchNorm2d-11           [-1, 64, 16, 16]             128
           Conv2d-12           [-1, 64, 16, 16]           2,048
      BatchNorm2d-13           [-1, 64, 16, 16]             128
       BasicBlock-14           [-1, 64, 16, 16]               0
           Conv2d-15            [-1, 128, 8, 8]          73,728
      BatchNorm2d-16            [-1, 128, 8, 8]             256
           Conv2d-17            [-1, 128, 8, 8]         147,456
      BatchNorm2d-18            [-1, 128, 8, 8]             256
           Conv2d-19            [-1, 128, 8, 8]           8,192
      BatchNorm2d-20            [-1, 128, 8, 8]             256
       BasicBlock-21            [-1, 128, 8, 8]               0
           Conv2d-22            [-1, 256, 4, 4]         294,912
      BatchNorm2d-23            [-1, 256, 4, 4]             512
           Conv2d-24            [-1, 256, 4, 4]         589,824
      BatchNorm2d-25            [-1, 256, 4, 4]             512
           Conv2d-26            [-1, 256, 4, 4]          32,768
      BatchNorm2d-27            [-1, 256, 4, 4]             512
       BasicBlock-28            [-1, 256, 4, 4]               0
AdaptiveAvgPool2d-29            [-1, 256, 1, 1]               0
           Linear-30                  [-1, 10]           2,570
================================================================
Total params: 1,228,970
Trainable params: 1,228,970
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.28
Params size (MB): 4.69
Estimated Total Size (MB): 7.98
----------------------------------------------------------------
```

In [ ]:

```
class ResNetLarge(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNetLarge, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[2], stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(512, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        downsample = None
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)


    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)
        return x

modellarge = ResNetLarge(BasicBlock, layers).to(device)
summary(modellarge, input_size=(3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,728
       BatchNorm2d-2           [-1, 64, 32, 32]             128
            Conv2d-3           [-1, 64, 32, 32]          36,864
       BatchNorm2d-4           [-1, 64, 32, 32]             128
            Conv2d-5           [-1, 64, 32, 32]          36,864
       BatchNorm2d-6           [-1, 64, 32, 32]             128
        BasicBlock-7           [-1, 64, 32, 32]               0
            Conv2d-8          [-1, 128, 16, 16]          73,728
       BatchNorm2d-9          [-1, 128, 16, 16]             256
           Conv2d-10          [-1, 128, 16, 16]         147,456
      BatchNorm2d-11          [-1, 128, 16, 16]             256
           Conv2d-12          [-1, 128, 16, 16]           8,192
      BatchNorm2d-13          [-1, 128, 16, 16]             256
       BasicBlock-14          [-1, 128, 16, 16]               0
           Conv2d-15            [-1, 256, 8, 8]         294,912
      BatchNorm2d-16            [-1, 256, 8, 8]             512
           Conv2d-17            [-1, 256, 8, 8]         589,824
      BatchNorm2d-18            [-1, 256, 8, 8]             512
           Conv2d-19            [-1, 256, 8, 8]          32,768
      BatchNorm2d-20            [-1, 256, 8, 8]             512
       BasicBlock-21            [-1, 256, 8, 8]               0
           Conv2d-22            [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-23            [-1, 512, 4, 4]           1,024
           Conv2d-24            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-25            [-1, 512, 4, 4]           1,024
           Conv2d-26            [-1, 512, 4, 4]         131,072
      BatchNorm2d-27            [-1, 512, 4, 4]           1,024
       BasicBlock-28            [-1, 512, 4, 4]               0
AdaptiveAvgPool2d-29            [-1, 512, 1, 1]               0
           Linear-30                  [-1, 10]           5,130
================================================================
Total params: 4,903,242
Trainable params: 4,903,242
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 6.57
Params size (MB): 18.70
Estimated Total Size (MB): 25.28
----------------------------------------------------------------
```

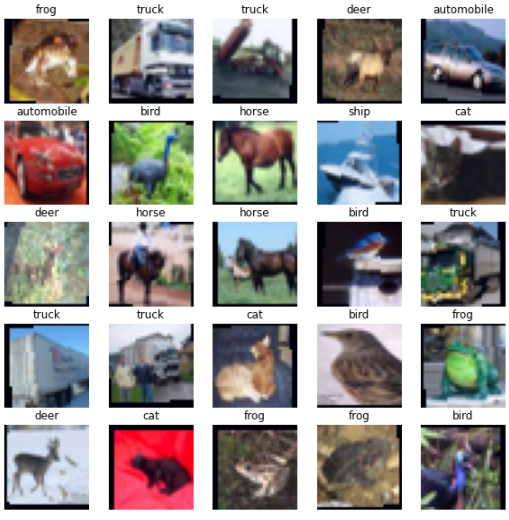
In [ ]:

```
# Load the dataset
# Defining Transformers for train and test set differently
train_transform = transforms.Compose([
    transforms.RandomRotation(5),
    transforms.RandomHorizontalFlip(0.5),
    transforms.RandomCrop(32, padding=2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023, 0.1994, 0.2010])
])
test_transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize(mean = [0.4914, 0.4822, 0.4465],
                                             std = [0.2023, 0.1994, 0.2010])
                    ])
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=train_transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=test_transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=4)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=4)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [ ]:

```
def plot_images(images, labels, classes, normalize = False):
    n_images = len(images)
    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))
    fig = plt.figure(figsize = (10, 10))
    for i in range(rows*cols):
        ax = fig.add_subplot(rows, cols, i+1)
        image = images[i]
        if normalize:
            image_min = image.min()
            image_max = image.max()
            image.clamp_(min = image_min, max = image_max)
            image.add_(-image_min).div_(image_max - image_min + 1e-5)
        ax.imshow(image.permute(1, 2, 0).cpu().numpy())
        ax.set_title(classes[labels[i]])
        ax.axis('off')

N_IMAGES = 25
images, labels = zip(*[(image, label) for image, label in [train_dataset[i] for i in range(N_IMAGES)]])
classes = test_dataset.classes
plot_images(images, labels, classes, normalize = True)
```



In [ ]:

```
def train(data_loader, model, criterion, optimizer, scheduler = None, early_stop=None):
    learning_rate_tracker = {}
    epoch_correct = 0
    running_loss = 0.0
    model.train()
```

```
    for i, (images, labels) in tqdm(enumerate(data_loader)):
        learning_rate_tracker[i] = optimizer.param_groups[0]['lr']

        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        running_loss += loss.item()
        predicted = torch.max(outputs.data, 1)[1]
        epoch_correct += (predicted == labels).sum().item()

        if early_stop and i==early_stop:
            break

        loss.backward()
        optimizer.step()
        if scheduler:
            scheduler.step()


    return epoch_correct , running_loss, learning_rate_tracker

def evaluate(data_loader, model, criterion):
    epoch_correct = 0
    running_loss = 0.0
    y_true = []
    y_pred = []
    model.eval()
    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            epoch_correct += (predicted == labels).sum().item()
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(predicted.cpu().numpy())

    return epoch_correct, running_loss, y_true, y_pred
```

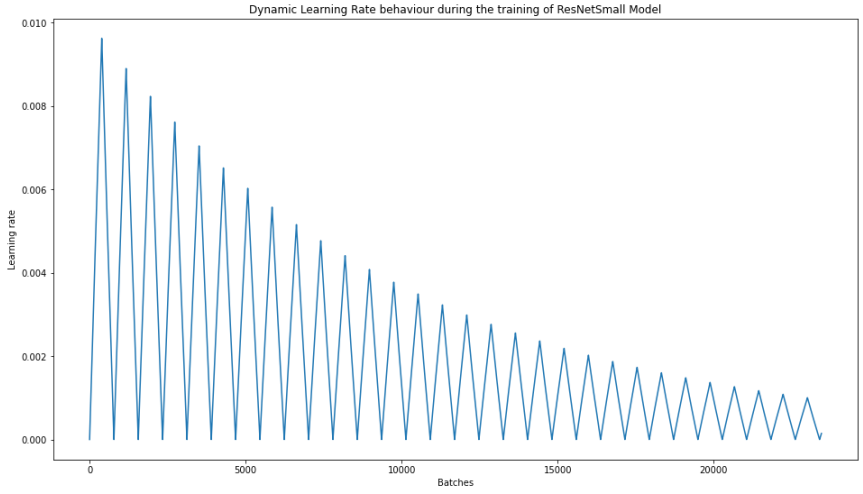
In [ ]:

```
lr_min = 1e-6
lr_max = 1e-2
epochs = 30
step_size = (len(train_dataset)/64) // 2

model = modelsmall
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=lr_min, momentum=0.9, nesterov=True)
scheduler = optim.lr_scheduler.CyclicLR(optimizer, base_lr=lr_min, max_lr=lr_max, step_size_up=step_size, step_size_down=step_size, gamma=0.9999, mode="exp_range", cycle_momentum=False)
lr_tracker = {}

train_loss_history_small = []
train_acc_history_small = []
val_loss_history_small = []
val_acc_history_small = []
y_pred_small = []
y_true_small = []
best_valid_loss = float('inf')

for epoch in range(epochs):
    print(f"Epoch: {epoch+1}/{epochs}")
    correct, loss, rate_tracker = train(data_loader=train_loader, model=model, criterion=criterion, optimizer=optimizer, scheduler=scheduler)
    accuracy = correct / len(train_loader.dataset)
    loss = loss / len(train_loader)
    train_loss = loss
    train_acc_history_small.append(accuracy)
    train_loss_history_small.append(loss)
    for key in rate_tracker.keys():
        lr_tracker[(epoch,key)] = rate_tracker[key]
    correct, loss, y_true_small, y_pred_small = evaluate(data_loader = test_loader, model=model, criterion=criterion)
    validation_accuracy = correct / len(test_loader.dataset)
    validation_loss = loss / len(test_loader)
    print(f"Train Accuracy: {accuracy*100:.2f}%, Train Loss: {train_loss}")
    print(f"Test Accuracy: {validation_accuracy*100:.2f}%, Test Loss: {validation_loss}")
    if validation_loss < best_valid_loss:
        best_valid_loss = validation_loss
        torch.save(model.state_dict(), 'ResNetSmall.pt')
    val_acc_history_small.append(validation_accuracy)
    val_loss_history_small.append(validation_loss)
```

```
Epoch: 1/30
782it [00:07, 111.24it/s]
Train Accuracy: 37.43%, Train Loss: 1.6959722177756718
Test Accuracy: 51.58%, Test Loss: 1.3311625202749944
Epoch: 2/30
782it [00:06, 112.60it/s]
Train Accuracy: 53.40%, Train Loss: 1.2890575701165992
Test Accuracy: 60.55%, Test Loss: 1.09142566448564
Epoch: 3/30
782it [00:07, 110.35it/s]
Train Accuracy: 59.81%, Train Loss: 1.118633839251745
Test Accuracy: 64.86%, Test Loss: 0.9786295127716793
Epoch: 4/30
782it [00:07, 110.35it/s]
Train Accuracy: 63.56%, Train Loss: 1.0208450611754022
Test Accuracy: 67.33%, Test Loss: 0.9052985675016026
Epoch: 5/30
782it [00:07, 109.85it/s]
Train Accuracy: 65.61%, Train Loss: 0.9602375282046131
Test Accuracy: 68.86%, Test Loss: 0.8616930933514978
Epoch: 6/30
782it [00:07, 109.27it/s]
Train Accuracy: 67.60%, Train Loss: 0.9112818957975758
Test Accuracy: 70.44%, Test Loss: 0.8231870995205679
Epoch: 7/30
782it [00:07, 110.69it/s]
Train Accuracy: 68.71%, Train Loss: 0.8794156540842617
Test Accuracy: 71.47%, Test Loss: 0.7964165343600473
Epoch: 8/30
782it [00:07, 111.09it/s]
Train Accuracy: 70.22%, Train Loss: 0.8425300571772144
Test Accuracy: 72.51%, Test Loss: 0.7651751240727248
Epoch: 9/30
782it [00:07, 111.42it/s]
Train Accuracy: 71.13%, Train Loss: 0.8148341079807038
Test Accuracy: 73.77%, Test Loss: 0.745846070681408
Epoch: 10/30
782it [00:06, 112.79it/s]
Train Accuracy: 71.87%, Train Loss: 0.7913056919184487
Test Accuracy: 74.29%, Test Loss: 0.7224326752553321
Epoch: 11/30
```

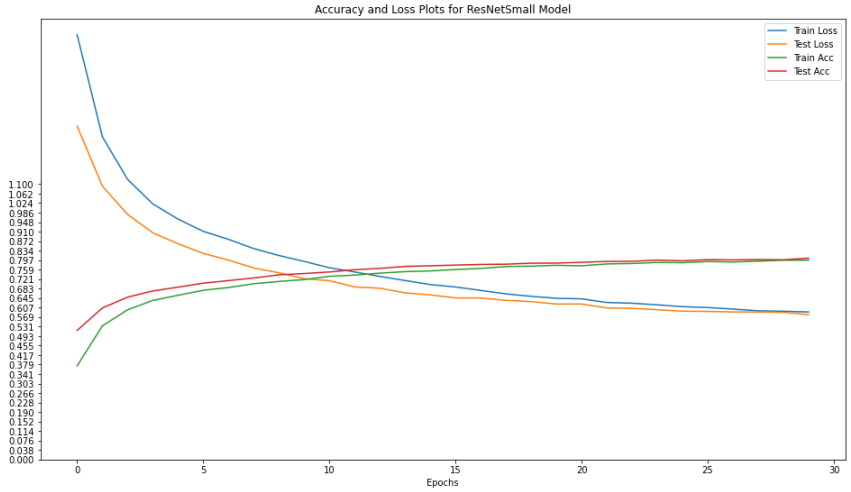
```
782it [00:07, 111.35it/s]
Train Accuracy: 73.18%, Train Loss: 0.7659507257401791
Test Accuracy: 74.92%, Test Loss: 0.7140307705493489
Epoch: 12/30
782it [00:07, 110.72it/s]
Train Accuracy: 73.66%, Train Loss: 0.7492306252650898
Test Accuracy: 75.81%, Test Loss: 0.6892960978921052
Epoch: 13/30
782it [00:06, 111.93it/s]
Train Accuracy: 74.45%, Train Loss: 0.731330990219665
Test Accuracy: 76.36%, Test Loss: 0.6842481922951473
Epoch: 14/30
782it [00:06, 113.17it/s]
Train Accuracy: 75.05%, Train Loss: 0.715022104742277
Test Accuracy: 77.13%, Test Loss: 0.6654635948739993
Epoch: 15/30
782it [00:06, 113.00it/s]
Train Accuracy: 75.32%, Train Loss: 0.6988651131653725
Test Accuracy: 77.40%, Test Loss: 0.6576877752686762
Epoch: 16/30
782it [00:07, 109.40it/s]
Train Accuracy: 75.88%, Train Loss: 0.6891592695661213
Test Accuracy: 77.66%, Test Loss: 0.6454789684076977
Epoch: 17/30
782it [00:07, 110.57it/s]
Train Accuracy: 76.33%, Train Loss: 0.6753186381152828
Test Accuracy: 77.92%, Test Loss: 0.6449884510344002
Epoch: 18/30
782it [00:07, 111.69it/s]
Train Accuracy: 77.09%, Train Loss: 0.6616063108072257
Test Accuracy: 78.02%, Test Loss: 0.6358467556868389
Epoch: 19/30
782it [00:06, 113.52it/s]
Train Accuracy: 77.25%, Train Loss: 0.6518765431840706
Test Accuracy: 78.43%, Test Loss: 0.6307068647472722
Epoch: 20/30
782it [00:07, 111.69it/s]
Train Accuracy: 77.57%, Train Loss: 0.6439396278632571
Test Accuracy: 78.46%, Test Loss: 0.6210683910710038
Epoch: 21/30
782it [00:06, 114.93it/s]
Train Accuracy: 77.38%, Train Loss: 0.6418194395425679
Test Accuracy: 78.78%, Test Loss: 0.6212930916600926
Epoch: 22/30
782it [00:07, 111.04it/s]
Train Accuracy: 78.13%, Train Loss: 0.62728391763042
Test Accuracy: 79.12%, Test Loss: 0.6053212379953664
Epoch: 23/30
782it [00:07, 111.57it/s]
Train Accuracy: 78.29%, Train Loss: 0.6244190858148247
Test Accuracy: 79.18%, Test Loss: 0.6040606601223065
Epoch: 24/30
782it [00:06, 112.70it/s]
Train Accuracy: 78.73%, Train Loss: 0.6179643522976609
Test Accuracy: 79.66%, Test Loss: 0.5983337256938789
Epoch: 25/30
782it [00:07, 110.78it/s]
Train Accuracy: 78.62%, Train Loss: 0.610577220692659
Test Accuracy: 79.37%, Test Loss: 0.5925299094361105
Epoch: 26/30
782it [00:06, 114.91it/s]
Train Accuracy: 79.08%, Train Loss: 0.6071711327794873
Test Accuracy: 79.83%, Test Loss: 0.591618735888961
Epoch: 27/30
782it [00:07, 111.57it/s]
Train Accuracy: 78.88%, Train Loss: 0.6008793675457426
Test Accuracy: 79.73%, Test Loss: 0.5894581791321942
Epoch: 28/30
782it [00:06, 112.79it/s]
Train Accuracy: 79.23%, Train Loss: 0.5939635429769525
Test Accuracy: 79.88%, Test Loss: 0.5892057268862512
Epoch: 29/30
782it [00:07, 111.01it/s]
Train Accuracy: 79.61%, Train Loss: 0.5923109132310619
Test Accuracy: 79.77%, Test Loss: 0.5877943929213627
Epoch: 30/30
782it [00:07, 110.57it/s]
Train Accuracy: 79.62%, Train Loss: 0.5894994664832455
Test Accuracy: 80.44%, Test Loss: 0.5786285602552875
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Dynamic Learning Rate behaviour during the training of ResNetSmall Model')
plt.plot(range(len(lr_tracker)), lr_tracker.values())
plt.xlabel('Batches')
plt.ylabel('Learning rate')
plt.show()
```



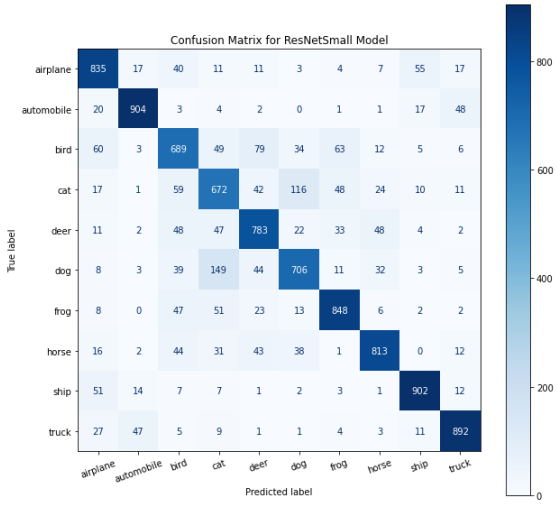
In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
```

```
plt.title('Accuracy and Loss Plots for ResNetSmall Model')
plt.plot(train_loss_history_small, label='Train Loss')
plt.plot(val_loss_history_small, label='Test Loss')
plt.plot(train_acc_history_small, label='Train Acc')
plt.plot(val_acc_history_small, label='Test Acc')
plt.legend()
plt.xlabel("Epochs")
yticks = np.linspace(0, 1.1, num=30)
ax.set_yticks(yticks)
plt.show()
```



In [ ]:
```
# Calculate the confusion matrix
cm = confusion_matrix(y_true_small, y_pred_small)
fig = plt.figure(figsize = (10, 10));
ax = fig.add_subplot(1, 1, 1);
ax.set_title('Confusion Matrix for ResNetSmall Model')
cm = ConfusionMatrixDisplay(cm, display_labels = classes);
cm.plot(values_format = 'd', cmap = 'Blues', ax = ax)
plt.xticks(rotation = 20)
```

Out[ ]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'airplane'),
  Text(1, 0, 'automobile'),
  Text(2, 0, 'bird'),
  Text(3, 0, 'cat'),
  Text(4, 0, 'deer'),
  Text(5, 0, 'dog'),
  Text(6, 0, 'frog'),
  Text(7, 0, 'horse'),
  Text(8, 0, 'ship'),
  Text(9, 0, 'truck')])
```



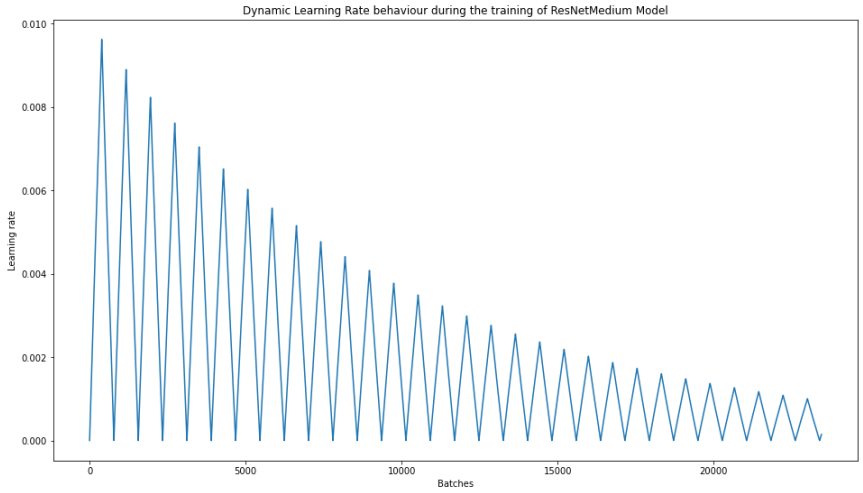
In [ ]:
```
lr_min = 1e-6
lr_max = 1e-2
epochs = 30
step_size = (len(train_dataset)/64) // 2

model = modelmedium
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=lr_min, momentum=0.9, nesterov=True)
scheduler = optim.lr_scheduler.CyclicLR(optimizer, base_lr=lr_min, max_lr=lr_max, step_size_up=step_size, step_size_down=step_size, gamma=0.9999, mode="exp_range", cycle_momentum=False)
lr_tracker = {}

train_loss_history_medium = []
train_acc_history_medium = []
val_loss_history_medium = []
val_acc_history_medium = []
y_pred_medium = []
y_true_medium = []
best_valid_loss = float('inf')

for epoch in range(epochs):
    print(f"Epoch: {epoch+1}/{epochs}")
    correct, loss, rate_tracker = train(data_loader=train_loader, model=model, criterion=criterion, optimizer=optimizer, scheduler=scheduler)
    accuracy = correct / len(train_loader.dataset)
    loss = loss / len(train_loader)
    train_loss = loss
    train_acc_history_medium.append(accuracy)
    train_loss_history_medium.append(loss)
    for key in rate_tracker.keys():
        lr_tracker[(epoch,key)] = rate_tracker[key]
```

```
    correct, loss, y_true_medium, y_pred_medium = evaluate(data_loader = test_loader, model=model, criterion=criterion)
    validation_accuracy = correct / len(test_loader.dataset)
    validation_loss = loss / len(test_loader)
    print(f"Train Accuracy: {accuracy*100:.2f}%, Train Loss: {train_loss}")
    print(f"Test Accuracy: {validation_accuracy*100:.2f}%, Test Loss: {validation_loss}")
    if validation_loss < best_valid_loss:
        best_valid_loss = validation_loss
        torch.save(model.state_dict(), 'ResNetMedium.pt')
    val_acc_history_medium.append(validation_accuracy)
    val_loss_history_medium.append(validation_loss)
```

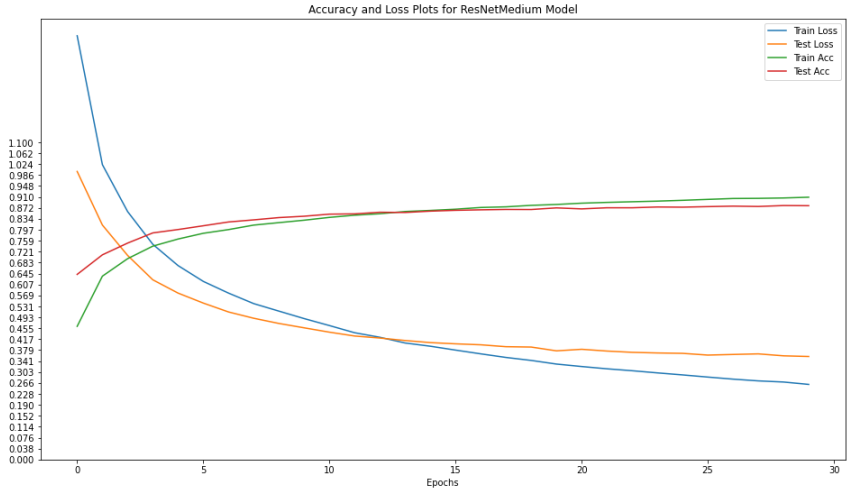
```
Epoch: 1/30
782it [00:07, 105.48it/s]
Train Accuracy: 46.18%, Train Loss: 1.4688814467633777
Test Accuracy: 64.18%, Test Loss: 0.9985658879492693
Epoch: 2/30
782it [00:07, 105.16it/s]
Train Accuracy: 63.56%, Train Loss: 1.0222012444072976
Test Accuracy: 70.96%, Test Loss: 0.812827889327031
Epoch: 3/30
782it [00:07, 106.27it/s]
Train Accuracy: 69.63%, Train Loss: 0.8594880220682725
Test Accuracy: 75.03%, Test Loss: 0.7075042455059708
Epoch: 4/30
782it [00:07, 106.92it/s]
Train Accuracy: 73.98%, Train Loss: 0.7459845128647812
Test Accuracy: 78.56%, Test Loss: 0.6229665603986971
Epoch: 5/30
782it [00:07, 107.37it/s]
Train Accuracy: 76.44%, Train Loss: 0.6723040058027447
Test Accuracy: 79.73%, Test Loss: 0.5769824285036439
Epoch: 6/30
782it [00:07, 106.98it/s]
Train Accuracy: 78.42%, Train Loss: 0.6175326255657484
Test Accuracy: 81.03%, Test Loss: 0.5423921880069053
Epoch: 7/30
782it [00:07, 104.66it/s]
Train Accuracy: 79.71%, Train Loss: 0.5769456775901872
Test Accuracy: 82.35%, Test Loss: 0.511577048783849
Epoch: 8/30
782it [00:07, 106.10it/s]
Train Accuracy: 81.26%, Train Loss: 0.5404994655638704
Test Accuracy: 83.08%, Test Loss: 0.4897377863051785
Epoch: 9/30
782it [00:07, 108.86it/s]
Train Accuracy: 82.12%, Train Loss: 0.5144487246870995
Test Accuracy: 83.87%, Test Loss: 0.4717256900421373
Epoch: 10/30
782it [00:07, 106.89it/s]
Train Accuracy: 82.97%, Train Loss: 0.4885648795787026
Test Accuracy: 84.34%, Test Loss: 0.456546327775451
Epoch: 11/30
782it [00:07, 106.55it/s]
Train Accuracy: 83.94%, Train Loss: 0.4644170618804215
Test Accuracy: 85.06%, Test Loss: 0.4415438612745066
Epoch: 12/30
782it [00:07, 103.43it/s]
Train Accuracy: 84.69%, Train Loss: 0.43958293130178283
Test Accuracy: 85.16%, Test Loss: 0.4281866009922544
Epoch: 13/30
782it [00:07, 105.33it/s]
Train Accuracy: 85.20%, Train Loss: 0.423816079202362 97
Test Accuracy: 85.72%, Test Loss: 0.42119093191851475
Epoch: 14/30
782it [00:07, 108.28it/s]
Train Accuracy: 85.98%, Train Loss: 0.4039933187577426
Test Accuracy: 85.64%, Test Loss: 0.41263495081928886
Epoch: 15/30
782it [00:07, 103.39it/s]
Train Accuracy: 86.34%, Train Loss: 0.3928191062739438
Test Accuracy: 86.10%, Test Loss: 0.40537300136438603
Epoch: 16/30
782it [00:07, 107.83it/s]
Train Accuracy: 86.77%, Train Loss: 0.37938593587149744
Test Accuracy: 86.39%, Test Loss: 0.4011301859548897
Epoch: 17/30
782it [00:07, 104.63it/s]
Train Accuracy: 87.37%, Train Loss: 0.36636039492724193
Test Accuracy: 86.56%, Test Loss: 0.3977719046128024
Epoch: 18/30
782it [00:07, 108.38it/s]
Train Accuracy: 87.60%, Train Loss: 0.3539097967660031
Test Accuracy: 86.69%, Test Loss: 0.3911374845797089
Epoch: 19/30
782it [00:07, 106.17it/s]
Train Accuracy: 88.11%, Train Loss: 0.3435981313285925
Test Accuracy: 86.67%, Test Loss: 0.3899396312464574
Epoch: 20/30
782it [00:07, 103.32it/s]
Train Accuracy: 88.42%, Train Loss: 0.330970448558516 54
Test Accuracy: 87.23%, Test Loss: 0.37678804993629456
Epoch: 21/30
782it [00:07, 106.50it/s]
Train Accuracy: 88.86%, Train Loss: 0.3224521252848303
Test Accuracy: 86.88%, Test Loss: 0.3822112738326856
Epoch: 22/30
782it [00:07, 106.45it/s]
Train Accuracy: 89.14%, Train Loss: 0.3145800076729959
Test Accuracy: 87.27%, Test Loss: 0.3760678332512546
Epoch: 23/30
782it [00:07, 105.88it/s]
Train Accuracy: 89.36%, Train Loss: 0.3078972375129952
Test Accuracy: 87.27%, Test Loss: 0.37165280327105976
Epoch: 24/30
782it [00:07, 107.70it/s]
Train Accuracy: 89.57%, Train Loss: 0.3004054766138801
Test Accuracy: 87.54%, Test Loss: 0.3694511747853771
Epoch: 25/30
782it [00:07, 107.85it/s]
Train Accuracy: 89.86%, Train Loss: 0.2933846808817533
Test Accuracy: 87.48%, Test Loss: 0.368121220142978
Epoch: 26/30
782it [00:07, 104.21it/s]
Train Accuracy: 90.18%, Train Loss: 0.2858991510213336
Test Accuracy: 87.67%, Test Loss: 0.36210738222120675
Epoch: 27/30
782it [00:07, 103.29it/s]
Train Accuracy: 90.49%, Train Loss: 0.27874315253761417
Test Accuracy: 87.84%, Test Loss: 0.36425760867679197
Epoch: 28/30
```

```
782it [00:07, 106.63it/s]
Train Accuracy: 90.53%, Train Loss: 0.27268813739118675
Test Accuracy: 87.72%, Test Loss: 0.36632380762677286
Epoch: 29/30
782it [00:07, 107.13it/s]
Train Accuracy: 90.67%, Train Loss: 0.2687381294262988
Test Accuracy: 88.04%, Test Loss: 0.3595244288444519
Epoch: 30/30
782it [00:07, 103.98it/s]
Train Accuracy: 90.93%, Train Loss: 0.2604838322724223
Test Accuracy: 87.99%, Test Loss: 0.3572485315002454
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Dynamic Learning Rate behaviour during the training of ResNetMedium Model')
plt.plot(range(len(lr_tracker)), lr_tracker.values())
plt.xlabel('Batches')
plt.ylabel('Learning rate')
plt.show()
```

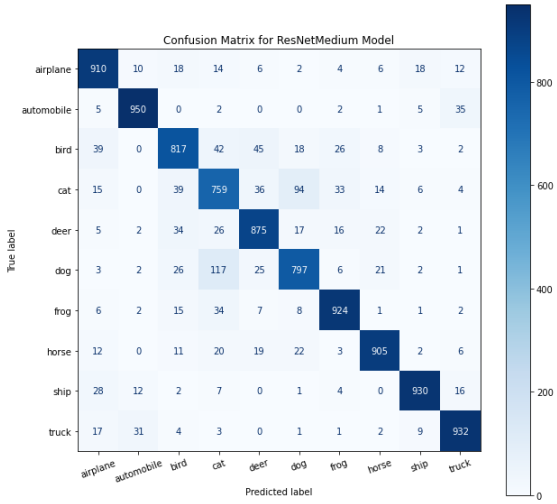


In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))

plt.title('Accuracy and Loss Plots for ResNetMedium Model')
plt.plot(train_loss_history_medium, label='Train Loss')
plt.plot(val_loss_history_medium, label='Test Loss')
plt.plot(train_acc_history_medium, label='Train Acc')
plt.plot(val_acc_history_medium, label='Test Acc')
plt.legend()
plt.xlabel("Epochs")
yticks = np.linspace(0, 1.1, num=30)
ax.set_yticks(yticks)
plt.show()
```



In [ ]:
```
# Calculate the confusion matrix
cm = confusion_matrix(y_true_medium, y_pred_medium)
fig = plt.figure(figsize = (10, 10));
ax = fig.add_subplot(1, 1, 1);
ax.set_title('Confusion Matrix for ResNetMedium Model')
cm = ConfusionMatrixDisplay(cm, display_labels = classes);
cm.plot(values_format = 'd', cmap = 'Blues', ax = ax)
plt.xticks(rotation = 20)
```

Out[ ]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'airplane'),
  Text(1, 0, 'automobile'),
  Text(2, 0, 'bird'),
  Text(3, 0, 'cat'),
  Text(4, 0, 'deer'),
  Text(5, 0, 'dog'),
  Text(6, 0, 'frog'),
  Text(7, 0, 'horse'),
  Text(8, 0, 'ship'),
  Text(9, 0, 'truck')])
```

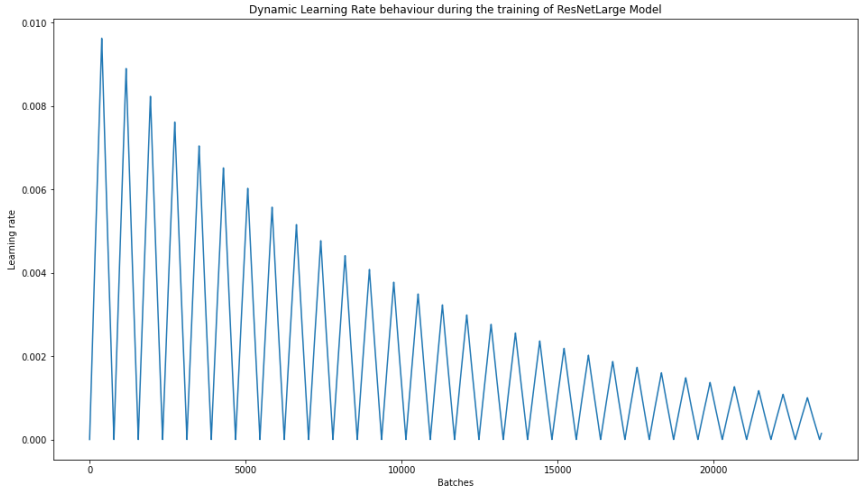


In [ ]:
```
lr_min = 1e-6
lr_max = 1e-2
epochs = 30
step_size = (len(train_dataset)/64) // 2

model = modellarge
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=lr_min, momentum=0.9, nesterov=True)
scheduler = optim.lr_scheduler.CyclicLR(optimizer, base_lr=lr_min, max_lr=lr_max, step_size_up=step_size, step_size_down=step_size, gamma=0.9999, mode="exp_range", cycle_momentum=False)
lr_tracker = {}

train_loss_history_large = []
train_acc_history_large = []
val_loss_history_large = []
val_acc_history_large = []
y_pred_large = []
y_true_large = []
best_valid_loss = float('inf')

for epoch in range(epochs):
    print(f"Epoch: {epoch+1}/{epochs}")
    correct, loss, rate_tracker = train(data_loader=train_loader, model=model, criterion=criterion, optimizer=optimizer, scheduler=scheduler)
    accuracy = correct / len(train_loader.dataset)
    loss = loss / len(train_loader)
    train_loss = loss
    train_acc_history_large.append(accuracy)
    train_loss_history_large.append(loss)
    for key in rate_tracker.keys():
        lr_tracker[(epoch,key)] = rate_tracker[key]
    correct, loss, y_true_large, y_pred_large = evaluate(data_loader = test_loader, model=model, criterion=criterion)
    validation_accuracy = correct / len(test_loader.dataset)
    validation_loss = loss / len(test_loader)
    print(f"Train Accuracy: {accuracy*100:.2f}%, Train Loss: {train_loss}")
    print(f"Test Accuracy: {validation_accuracy*100:.2f}%, Test Loss: {validation_loss}")
    if validation_loss < best_valid_loss:
        best_valid_loss = validation_loss
        torch.save(model.state_dict(), 'ResNetLarge.pt')
    val_acc_history_large.append(validation_accuracy)
    val_loss_history_large.append(validation_loss)
```

```
Epoch: 1/30
782it [00:10, 77.87it/s]
Train Accuracy: 48.92%, Train Loss: 1.4058114572254288
Test Accuracy: 67.16%, Test Loss: 0.9061055942705483
Epoch: 2/30
782it [00:09, 78.98it/s]
Train Accuracy: 66.51%, Train Loss: 0.9404833051935791
Test Accuracy: 73.88%, Test Loss: 0.7336463125268365
Epoch: 3/30
782it [00:10, 78.02it/s]
Train Accuracy: 72.53%, Train Loss: 0.77577556170466
Test Accuracy: 77.88%, Test Loss: 0.6237734108214166
Epoch: 4/30
782it [00:10, 78.19it/s]
Train Accuracy: 76.42%, Train Loss: 0.6752028043389016
Test Accuracy: 81.01%, Test Loss: 0.5592745715265821
Epoch: 5/30
782it [00:10, 77.55it/s]
Train Accuracy: 78.86%, Train Loss: 0.6052702906567727
Test Accuracy: 82.03%, Test Loss: 0.5175294848574195
Epoch: 6/30
782it [00:09, 78.68it/s]
Train Accuracy: 81.01%, Train Loss: 0.5501928740297742
Test Accuracy: 83.42%, Test Loss: 0.4905243561526013
Epoch: 7/30
782it [00:09, 78.51it/s]
Train Accuracy: 82.43%, Train Loss: 0.5081557086323534
Test Accuracy: 84.51%, Test Loss: 0.4533748639996644
Epoch: 8/30
782it [00:09, 79.31it/s]
Train Accuracy: 83.75%, Train Loss: 0.4682275569995346
Test Accuracy: 85.21%, Test Loss: 0.44090694616175
Epoch: 9/30
782it [00:09, 78.73it/s]
Train Accuracy: 84.74%, Train Loss: 0.44072029074592056
Test Accuracy: 85.90%, Test Loss: 0.4206506500768054
Epoch: 10/30
782it [00:09, 78.85it/s]
Train Accuracy: 85.72%, Train Loss: 0.4126177037425358
Test Accuracy: 86.30%, Test Loss: 0.40764474109479576
Epoch: 11/30
782it [00:09, 78.94it/s]
Train Accuracy: 86.45%, Train Loss: 0.3917580034841052
Test Accuracy: 86.57%, Test Loss: 0.39937475295203506
Epoch: 12/30
782it [00:09, 78.92it/s]
Train Accuracy: 87.17%, Train Loss: 0.37006695851531174
Test Accuracy: 87.03%, Test Loss: 0.383157089827167
Epoch: 13/30
782it [00:10, 77.75it/s]
```

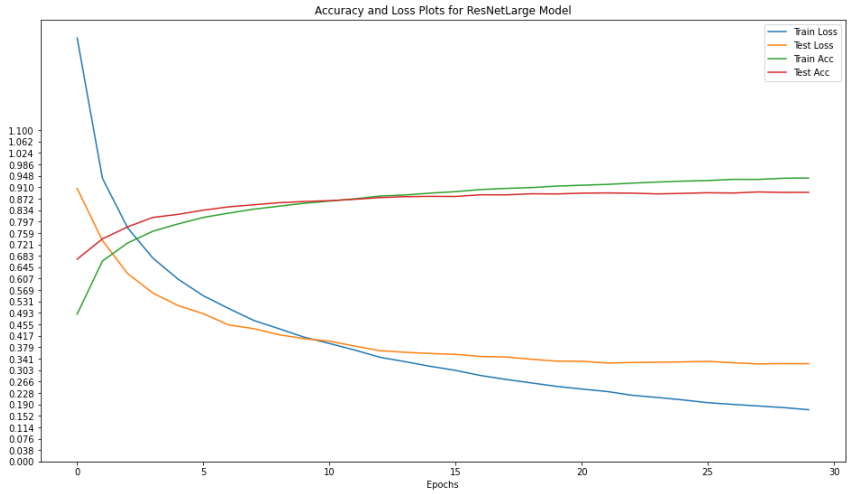
```
Train Accuracy: 88.11%, Train Loss: 0.3458830364753523
Test Accuracy: 87.59%, Test Loss: 0.36779645512438125
Epoch: 14/30
782it [00:09, 78.64it/s]
Train Accuracy: 88.46%, Train Loss: 0.3313629613889148
Test Accuracy: 87.90%, Test Loss: 0.3625057391869794
Epoch: 15/30
782it [00:09, 79.02it/s]
Train Accuracy: 89.08%, Train Loss: 0.3158440920988769
Test Accuracy: 88.01%, Test Loss: 0.35847265520103416
Epoch: 16/30
782it [00:10, 77.83it/s]
Train Accuracy: 89.57%, Train Loss: 0.3024188351086186
Test Accuracy: 87.97%, Test Loss: 0.3555400956683098
Epoch: 17/30
782it [00:10, 78.17it/s]
Train Accuracy: 90.27%, Train Loss: 0.28528198902792945
Test Accuracy: 88.53%, Test Loss: 0.3485969147010214
Epoch: 18/30
782it [00:09, 78.50it/s]
Train Accuracy: 90.64%, Train Loss: 0.2724381180675438
Test Accuracy: 88.51%, Test Loss: 0.3466635327430288
Epoch: 19/30
782it [00:09, 78.40it/s]
Train Accuracy: 90.91%, Train Loss: 0.2608195971554655
Test Accuracy: 88.87%, Test Loss: 0.3394074491254843
Epoch: 20/30
782it [00:09, 78.74it/s]
Train Accuracy: 91.40%, Train Loss: 0.2494079852309983
Test Accuracy: 88.81%, Test Loss: 0.3330639451742172
Epoch: 21/30
782it [00:10, 77.81it/s]
Train Accuracy: 91.70%, Train Loss: 0.24052330293237706
Test Accuracy: 89.07%, Test Loss: 0.3327259635374804
Epoch: 22/30
782it [00:09, 78.45it/s]
Train Accuracy: 91.98%, Train Loss: 0.23239734841277226
Test Accuracy: 89.14%, Test Loss: 0.32689990112735967
Epoch: 23/30
782it [00:10, 78.02it/s]
Train Accuracy: 92.40%, Train Loss: 0.21973636169986957
Test Accuracy: 89.07%, Test Loss: 0.32861825183128857
Epoch: 24/30
782it [00:09, 78.22it/s]
Train Accuracy: 92.76%, Train Loss: 0.21252951015006094
Test Accuracy: 88.82%, Test Loss: 0.329613656327603
Epoch: 25/30
782it [00:09, 79.11it/s]
Train Accuracy: 93.05%, Train Loss: 0.20455673771917515
Test Accuracy: 89.00%, Test Loss: 0.3305412292195733
Epoch: 26/30
782it [00:10, 77.56it/s]
Train Accuracy: 93.26%, Train Loss: 0.19517021642907348
Test Accuracy: 89.21%, Test Loss: 0.33258719144353444
Epoch: 27/30
782it [00:09, 79.10it/s]
Train Accuracy: 93.64%, Train Loss: 0.18944301981659953
Test Accuracy: 89.11%, Test Loss: 0.32758472826640317
Epoch: 28/30
782it [00:10, 77.56it/s]
Train Accuracy: 93.63%, Train Loss: 0.18426255238673572
Test Accuracy: 89.47%, Test Loss: 0.3239446260082494
Epoch: 29/30
782it [00:09, 79.28it/s]
Train Accuracy: 94.02%, Train Loss: 0.1789445672327143
Test Accuracy: 89.32%, Test Loss: 0.3247904695428101
Epoch: 30/30
782it [00:10, 77.78it/s]
Train Accuracy: 94.10%, Train Loss: 0.17178399384002704
Test Accuracy: 89.34%, Test Loss: 0.3246337286890692
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Dynamic Learning Rate behaviour during the training of ResNetLarge Model')
plt.plot(range(len(lr_tracker)), lr_tracker.values())
plt.xlabel('Batches')
plt.ylabel('Learning rate')
plt.show()
```



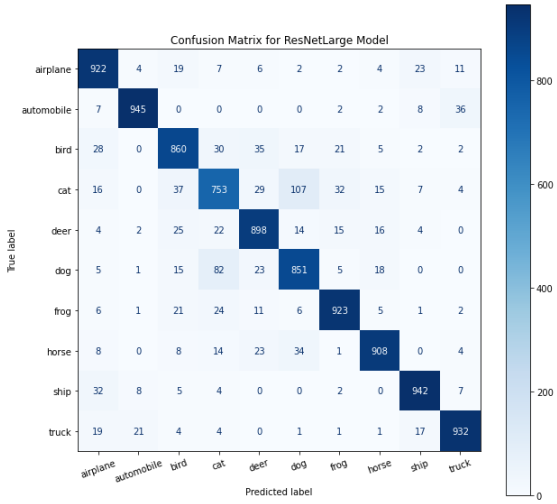
In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))

plt.title('Accuracy and Loss Plots for ResNetLarge Model')
plt.plot(train_loss_history_large, label='Train Loss')
plt.plot(val_loss_history_large, label='Test Loss')
plt.plot(train_acc_history_large, label='Train Acc')
plt.plot(val_acc_history_large, label='Test Acc')
plt.legend()
plt.xlabel("Epochs")
yticks = np.linspace(0, 1.1, num=30)
```

```
ax.set_yticks(yticks)
plt.show()
```

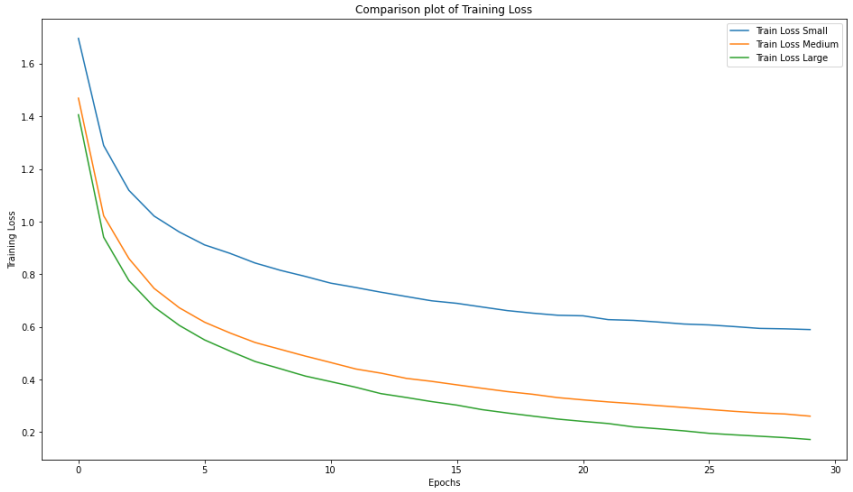


In [ ]:
```
# Calculate the confusion matrix
cm = confusion_matrix(y_true_large, y_pred_large)
fig = plt.figure(figsize = (10, 10));
ax = fig.add_subplot(1, 1, 1);
ax.set_title('Confusion Matrix for ResNetLarge Model')
cm = ConfusionMatrixDisplay(cm, display_labels = classes);
cm.plot(values_format = 'd', cmap = 'Blues', ax = ax)
plt.xticks(rotation = 20)
```

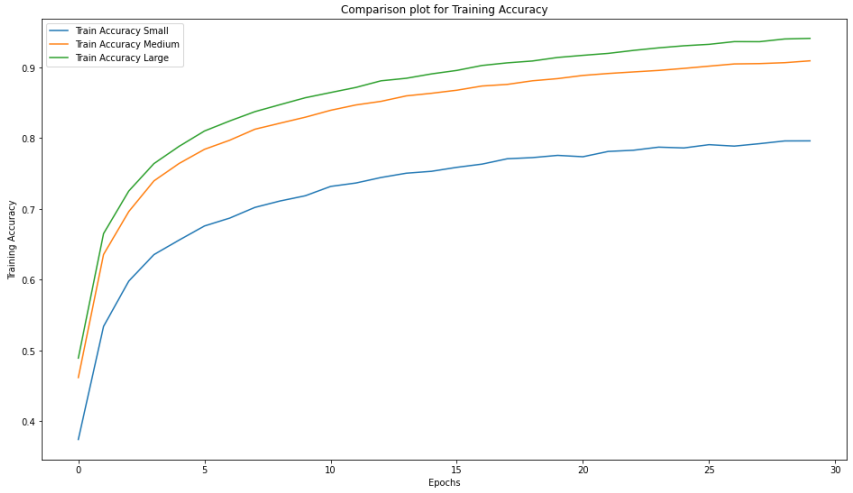
Out[ ]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'airplane'),
  Text(1, 0, 'automobile'),
  Text(2, 0, 'bird'),
  Text(3, 0, 'cat'),
  Text(4, 0, 'deer'),
  Text(5, 0, 'dog'),
  Text(6, 0, 'frog'),
  Text(7, 0, 'horse'),
  Text(8, 0, 'ship'),
  Text(9, 0, 'truck')])
```



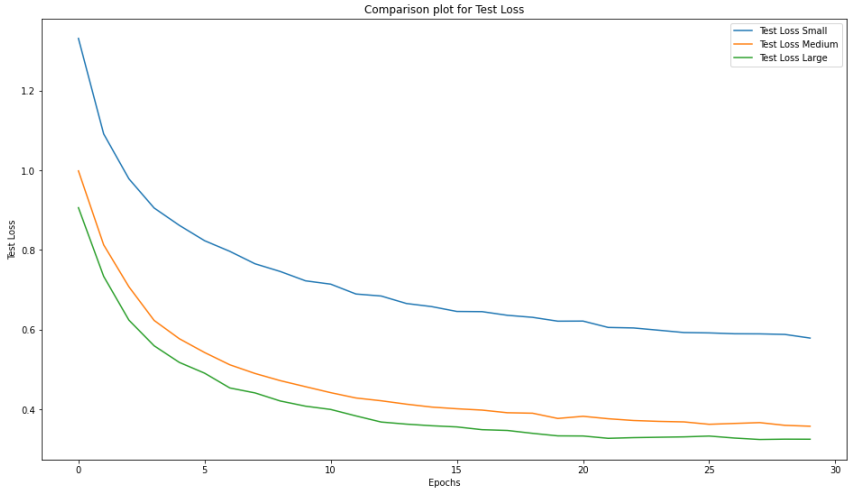
In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Comparison plot of Training Loss')
plt.plot(train_loss_history_small, label='Train Loss Small')
plt.plot(train_loss_history_medium, label='Train Loss Medium')
plt.plot(train_loss_history_large, label='Train Loss Large')
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Training Loss")
plt.show()
```



In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Comparison plot for Training Accuracy')
plt.plot(train_acc_history_small, label='Train Accuracy Small')
plt.plot(train_acc_history_medium, label='Train Accuracy Medium')
plt.plot(train_acc_history_large, label='Train Accuracy Large')
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Training Accuracy")
plt.show()
```



In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Comparison plot for Test Loss')
plt.plot(val_loss_history_small, label='Test Loss Small')
plt.plot(val_loss_history_medium, label='Test Loss Medium')
plt.plot(val_loss_history_large, label='Test Loss Large')
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Test Loss")
plt.show()
```



In [ ]:
```
fig, ax = plt.subplots(figsize=(16,9))
plt.title('Comparison plot for Test Accuracy')
plt.plot(val_acc_history_small, label='Test Accuracy Small')
plt.plot(val_acc_history_medium, label='Test Accuracy Medium')
plt.plot(val_acc_history_large, label='Test Accuracy Large')
plt.legend()
plt.xlabel("Epochs")
```

```
plt.ylabel("Test Accuracy")
plt.show()
```