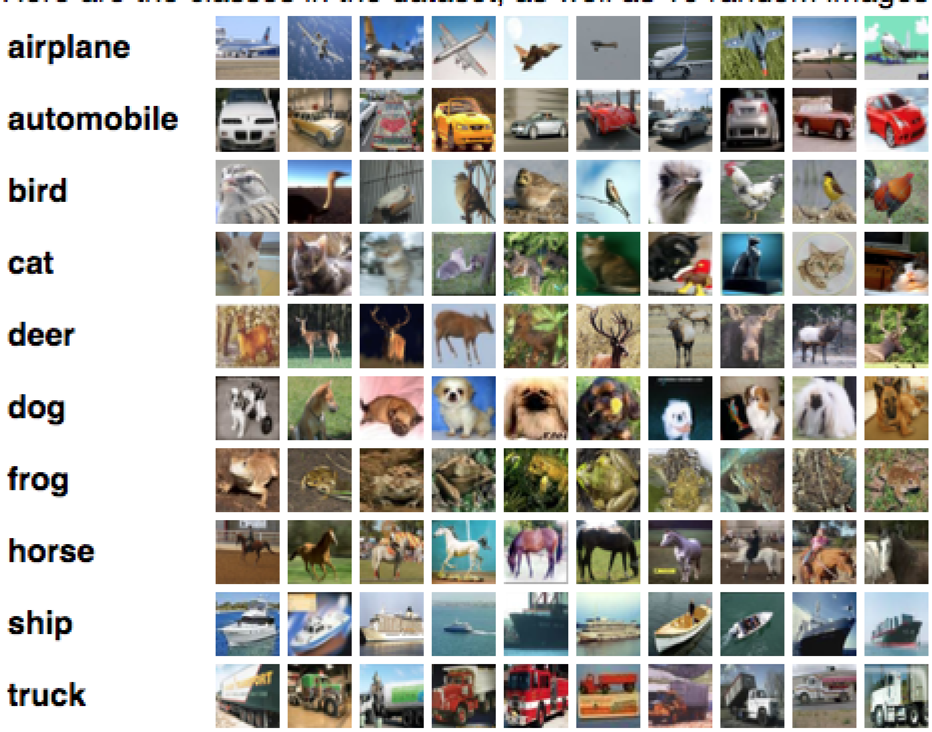
```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as dataset
from torchvision.datasets import FashionMNIST
import matplotlib.pyplot as plt
import numpy as np



# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cuda')
```

CRFAR10 Dataset

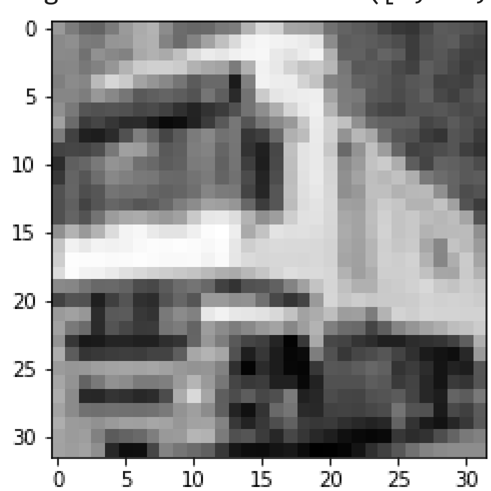
Here are the classes in the dataset, as well as 10 random images from each:




```
batch_size = 128

transform = transforms.Compose([
    transforms.ToTensor()
])


# CIFAR 10 dataset
```

```
# CIFAR-10 dataset
train_dataset = dataset.CIFAR10(root='../../data/', train=True, transform=transform, downl
test_dataset = dataset.CIFAR10(root='../../data/', train=False, transform=transform, downl


print('Train dataset size = ',len(train_dataset))
print('Test dataset size = ',len(test_dataset))
img, label = train_dataset[1]
print('Image size = ',img.shape, '(', label, ')')
plt.imshow(img[0,:,:], cmap='gray')



# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)


del train_dataset
del test_dataset
```

```
Files already downloaded and verified
Files already downloaded and verified
Train dataset size =  50000
Test dataset size =  10000
Image size =  torch.Size([3, 32, 32]) ( 9 )
```



Neural Network Model

Layer	Operations	Input Size	Output Size
Layer 1	conv3x3 + BatchNorm + Relu	3 x 32 x 32	128 x 32 x 32

Layer	Operations	Input Size	Output Size
Layer 2	conv3x3 + BatchNorm + Relu + maxpool	128 x 32 x 32	256 x 16 x 16
Layer 3	conv3x3 + BatchNorm + Relu	256 x 16 x 16	128 x 16 x 16
Layer 4	conv3x3 + BatchNorm + Relu + maxpool	128 x 16 x 16	64 x 8 x 8
Layer 5	Fully connected	1 x 4096	1 x 512
Layer 6	Fully connected	1 x 512	1 x 10

```
class Model(nn.Module):
    def __init__(self, num_classes):
        super(Model, self).__init__()



        self.conv1 = nn.Conv2d(3, 128, kernel_size= 5, padding= 2, stride= 1)
        self.conv2 = nn.Conv2d(128, 256, kernel_size= 5, padding= 2, stride= 1)
        self.conv3 = nn.Conv2d(256, 128, kernel_size= 5, padding= 2, stride= 1)
        self.conv4 = nn.Conv2d(128, 64, kernel_size= 5, padding= 2, stride= 1)
        self.n= np.int(3*32*32 * 64/3 * 1/4 * 1/4)

        self.linear1  = nn.Linear(self.n , hidden_size)
        self.linear2  = nn.Linear (hidden_size, num_classes)

        self.bn1 = nn.BatchNorm2d(128)
        self.bn2 = nn.BatchNorm2d(256)
        self.bn3 = nn.BatchNorm2d(64)
        self.bn4 = nn.BatchNorm2d(128)

        self.relu=nn.ReLU()
        self.max_pool = nn.MaxPool2d(2, stride=2)


        self.init()


    def init(self):
        nn.init.xavier_uniform_(self.conv1.weight)
        nn.init.xavier_uniform_(self.conv2.weight)
        nn.init.xavier_uniform_(self.conv3.weight)
        nn.init.xavier_uniform_(self.conv4.weight)
        nn.init.xavier_uniform_(self.linear1.weight)
        nn.init.xavier_uniform_(self.linear2.weight)


    def forward(self, x):

        out = self.relu(self.bn1(self.conv1(x)))
        out = self.max_pool(self.relu(self.bn2(self.conv2(out))))

        out = self.relu(self.bn4(self.conv3(out)))

```

```
        out = self.max_pool(self.relu(self.bn3(self.conv4(out))))


        out = out.view(out.size(0), -1)

        out = nn.functional.dropout(out, 0.5)

        out = self.linear1(out)

        out = self.linear2(out)

        return out


```

```
num_classes = 10
learning_rate = 0.001
hidden_size = 512


model1 = Model(num_classes).to(device)
model2 = Model(num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer1 = torch.optim.Adam(model1.parameters(), lr=learning_rate)
optimizer2 = torch.optim.SGD(model2.parameters(), lr=learning_rate, momentum=0.8, weight_d

```

```
# Train the model
def Train(model, optimizer, num_epochs):
    total_step = len(train_loader)
    loss_val = []
    count = []

    model.train()
    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(train_loader):
            images = images.to(device)
            labels = labels.to(device)

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



            if (i+1) % 100 == 0:
```

```
                count.append(i+1 + epoch*total_step)
                loss_val.append(loss.item())
                print('Epoch [%d/%d], Step [%d/%d], Loss: %.4f'%(epoch+1, num_epochs, i+1,

    return count, loss_val
```

```
# Test the model


def Test(model):

    model.eval()

    correct = 0
    total = 0

    actual_labels = []
    predicted_labels = []

    for images, labels in test_loader:

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

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

        labelsCPU = labels.data.cpu().numpy()
        predictedCPU = predicted.data.cpu().numpy()
        predicted_labels.append(predictedCPU)
        actual_labels.append(labelsCPU)



    print('Accuracy of the model = %f'%(100 * correct / total))

```

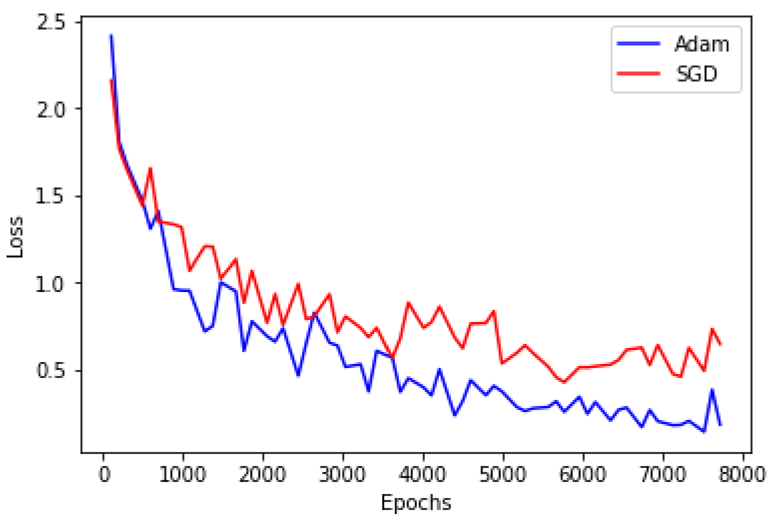
```
num_epochs = 20
count, loss1 = Train(model1, optimizer1, num_epochs)

count, loss2 = Train(model2, optimizer2, num_epochs)
```

```
Epoch [1/20], Step [200/391], Loss: 1.7652
Epoch [1/20], Step [300/391], Loss: 1.6426
Epoch [2/20], Step [100/391], Loss: 1.4393

Epoch [2/20], Step [200/391], Loss: 1.6557
Epoch [2/20], Step [300/391], Loss: 1.3479
Epoch [3/20], Step [100/391], Loss: 1.3322
Epoch [3/20], Step [200/391], Loss: 1.3158
```

```
Epoch [3/20], Step [300/391], Loss: 1.0677
Epoch [4/20], Step [100/391], Loss: 1.2057
Epoch [4/20], Step [200/391], Loss: 1.2035
Epoch [4/20], Step [300/391], Loss: 1.0225
Epoch [5/20], Step [100/391], Loss: 1.1315
Epoch [5/20], Step [200/391], Loss: 0.8841
Epoch [5/20], Step [300/391], Loss: 1.0673
Epoch [6/20], Step [100/391], Loss: 0.7674
Epoch [6/20], Step [200/391], Loss: 0.9342
Epoch [6/20], Step [300/391], Loss: 0.7556
Epoch [7/20], Step [100/391], Loss: 0.9919
Epoch [7/20], Step [200/391], Loss: 0.7885
Epoch [7/20], Step [300/391], Loss: 0.8058
Epoch [8/20], Step [100/391], Loss: 0.9330
Epoch [8/20], Step [200/391], Loss: 0.7137
Epoch [8/20], Step [300/391], Loss: 0.8047
Epoch [9/20], Step [100/391], Loss: 0.7390
Epoch [9/20], Step [200/391], Loss: 0.6854
Epoch [9/20], Step [300/391], Loss: 0.7398
Epoch [10/20], Step [100/391], Loss: 0.5653
Epoch [10/20], Step [200/391], Loss: 0.6788
Epoch [10/20], Step [300/391], Loss: 0.8845
Epoch [11/20], Step [100/391], Loss: 0.7393
Epoch [11/20], Step [200/391], Loss: 0.7727
Epoch [11/20], Step [300/391], Loss: 0.8616
Epoch [12/20], Step [100/391], Loss: 0.6837
Epoch [12/20], Step [200/391], Loss: 0.6214
Epoch [12/20], Step [300/391], Loss: 0.7636
Epoch [13/20], Step [100/391], Loss: 0.7667
Epoch [13/20], Step [200/391], Loss: 0.8357
Epoch [13/20], Step [300/391], Loss: 0.5339
Epoch [14/20], Step [100/391], Loss: 0.5962
Epoch [14/20], Step [200/391], Loss: 0.6393
Epoch [14/20], Step [300/391], Loss: 0.5944
Epoch [15/20], Step [100/391], Loss: 0.5152
Epoch [15/20], Step [200/391], Loss: 0.4546
Epoch [15/20], Step [300/391], Loss: 0.4263
Epoch [16/20], Step [100/391], Loss: 0.5127
Epoch [16/20], Step [200/391], Loss: 0.5115
Epoch [16/20], Step [300/391], Loss: 0.5177
Epoch [17/20], Step [100/391], Loss: 0.5281
Epoch [17/20], Step [200/391], Loss: 0.5542
Epoch [17/20], Step [300/391], Loss: 0.6127
Epoch [18/20], Step [100/391], Loss: 0.6253
Epoch [18/20], Step [200/391], Loss: 0.5253
Epoch [18/20], Step [300/391], Loss: 0.6404
Epoch [19/20], Step [100/391], Loss: 0.4736
Epoch [19/20], Step [200/391], Loss: 0.4581
Epoch [19/20], Step [300/391], Loss: 0.6244
Epoch [20/20], Step [100/391], Loss: 0.4918
Epoch [20/20], Step [200/391], Loss: 0.7330
Epoch [20/20], Step [300/391], Loss: 0.6470
```

```
import matplotlib.pyplot as plt

fig = plt.figure()
plt.plot(count, loss1, color='blue', label='Adam')
```

```
plt.plot(count, loss2, color='red', label='SGD')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

print('Adam Optimizer')
Test(model1)

print('SGD Optimizer')
Test(model2)
```




```
Adam Optimizer
Accuracy of the model = 80.459999
SGD Optimizer
Accuracy of the model = 73.739998
```