PYTORCH TUTORIAL

An explanation of the example code

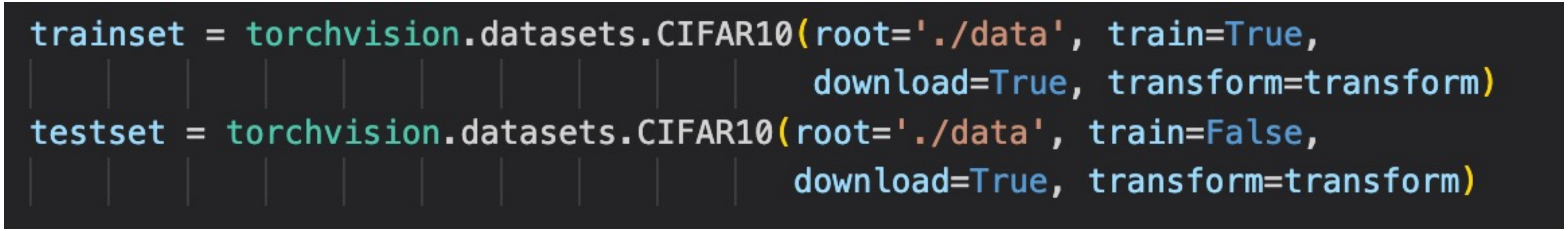
OUTLINE

1. Dataset and Dataloader
2. Building a network
3. Loss function and optimizer
4. Training!
5. Evaluation

DATASET AND DATALOADER

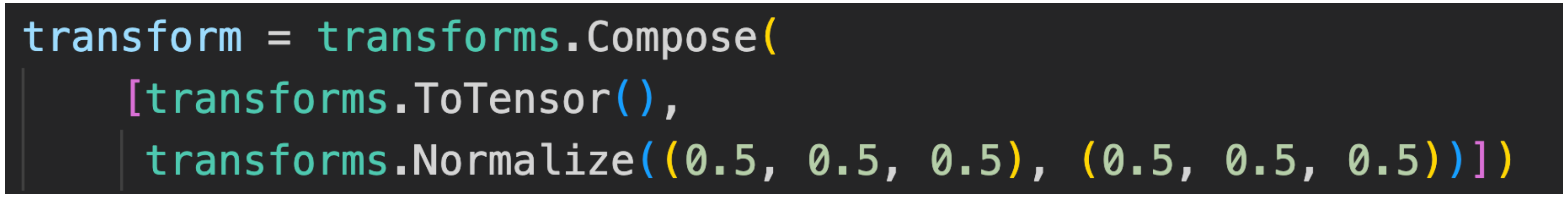
FIRST STEP : GET A DATASET

Get your dataset from pytorch built-in dataset.



```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
```

Use transform to convert an image (numpy array or PIL image) to tensor and do some augmentation(Random flip, color jitter ...) to extend your dataset.



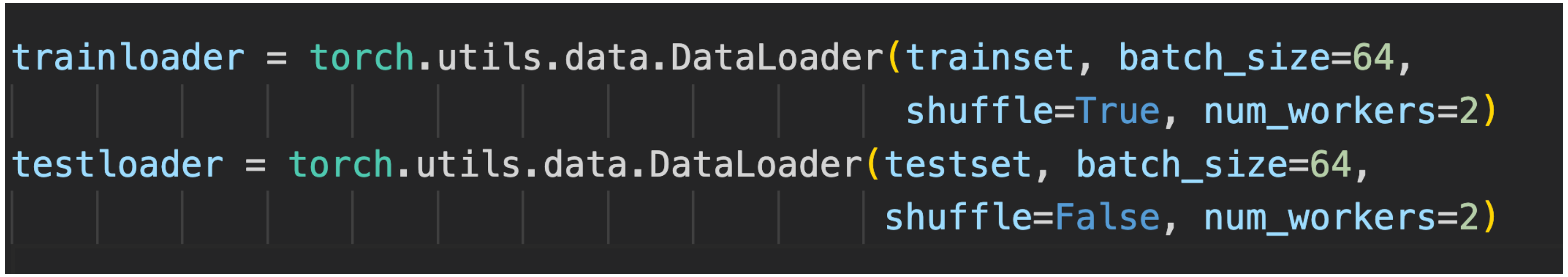
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

# tips: you can concatenate serval dataset by using torch.utils.data.ConcatDataset()

DATASET AND DATALOADER

SECOND STEP: DATALOADER

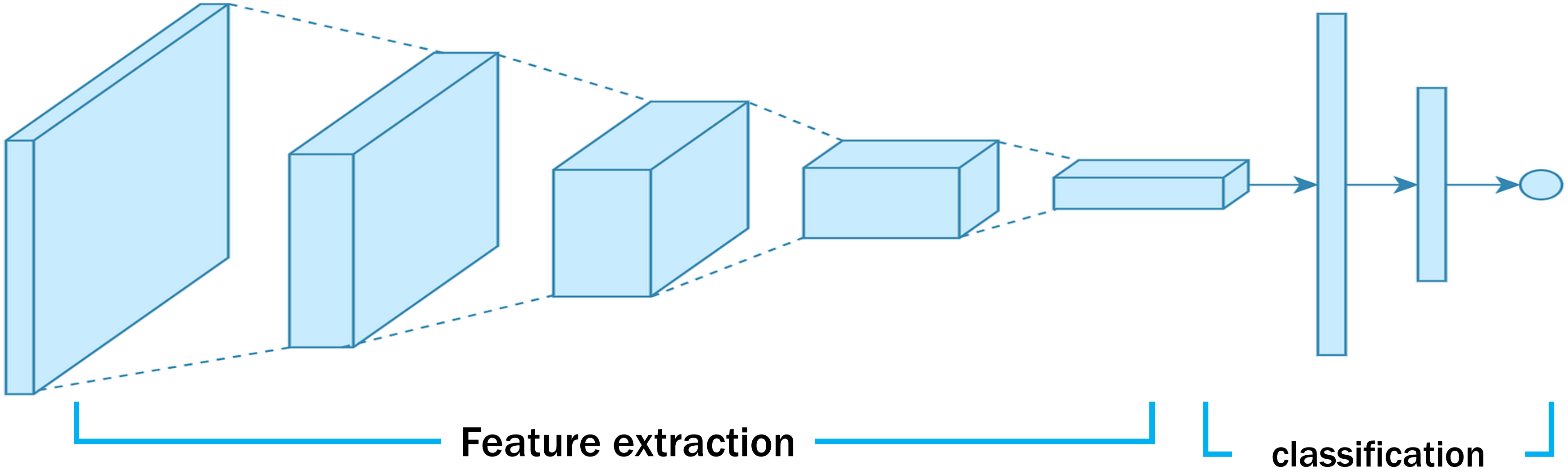
DataLoader is a utility that helps with data loading and batching!



```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                         shuffle=False, num_workers=2)
```

What is num_worker?   How to set batch size?

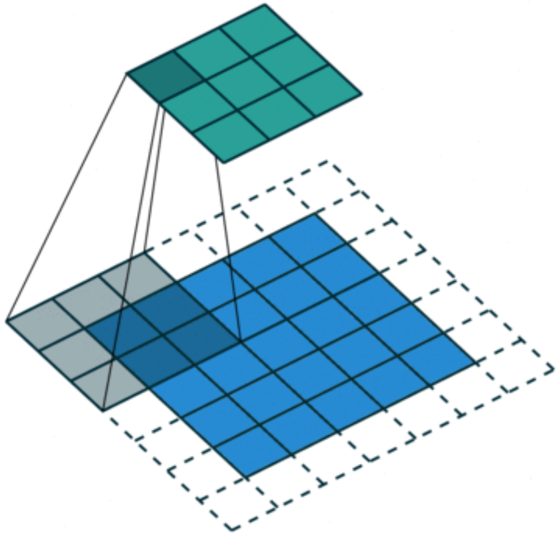
BUILDING A NETWORK



Feature extraction: Use convolution layer
Classification: Use fully connect layer

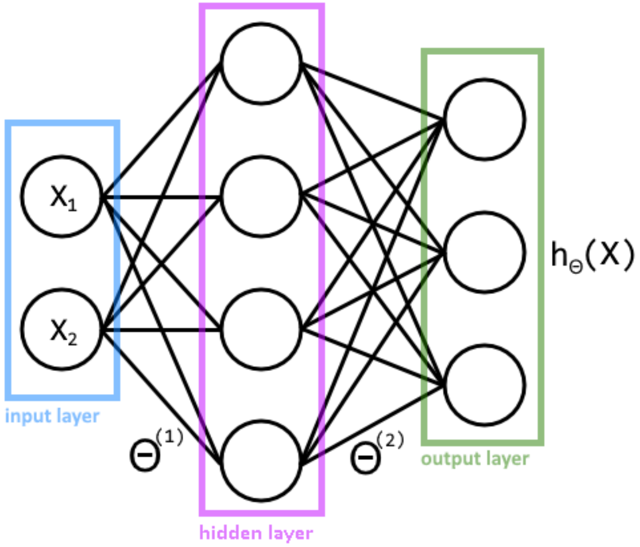
BUILDING A NETWORK

CONVOLUTION AND FULLY CONNECT



Convolutional operations involve small, learnable kernels. These filters are matrices of weights.

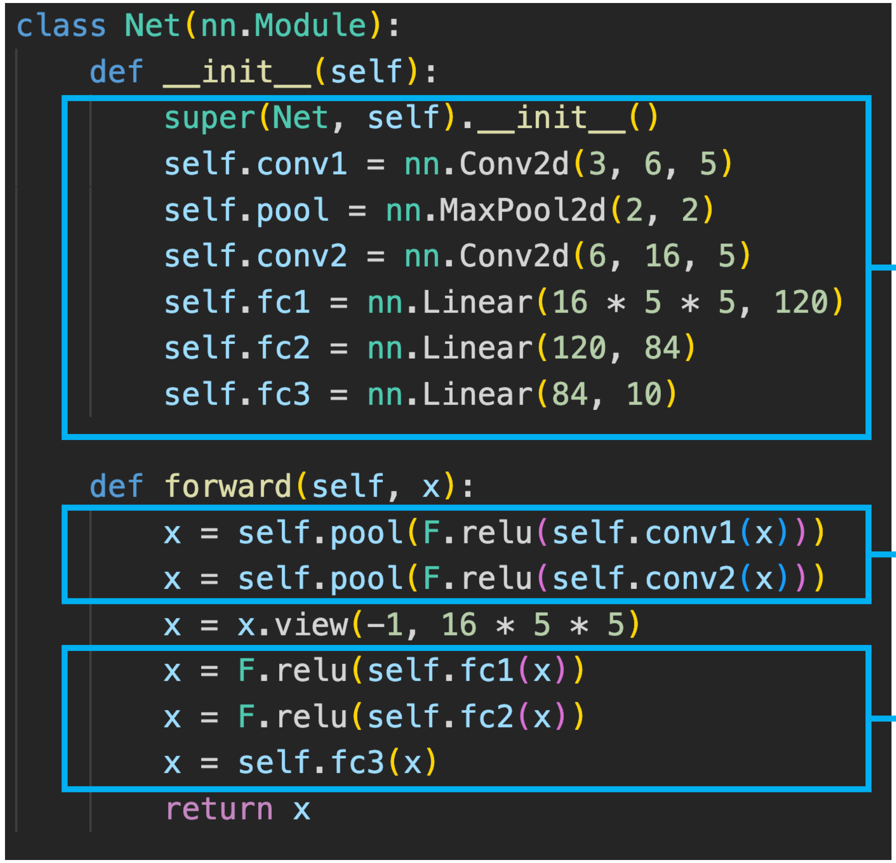
nn.Conv2d(in_channels=1, out_channels=16, size=(3, 3))
shape of weights = (16, 1, 3, 3)



2-layer fully-connected netual network

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

BUILDING A NETWORK



```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

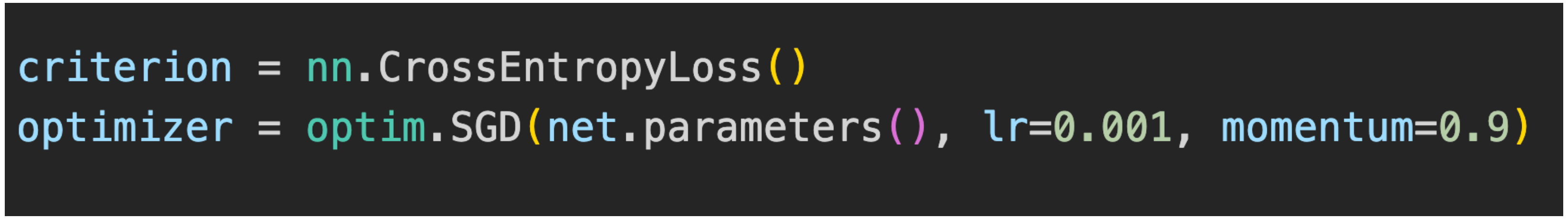
Define all module you need.

A convolution network with two layers.

A fully connect network with three layers.

LOSS FUNCTION AND OPTIMIZER

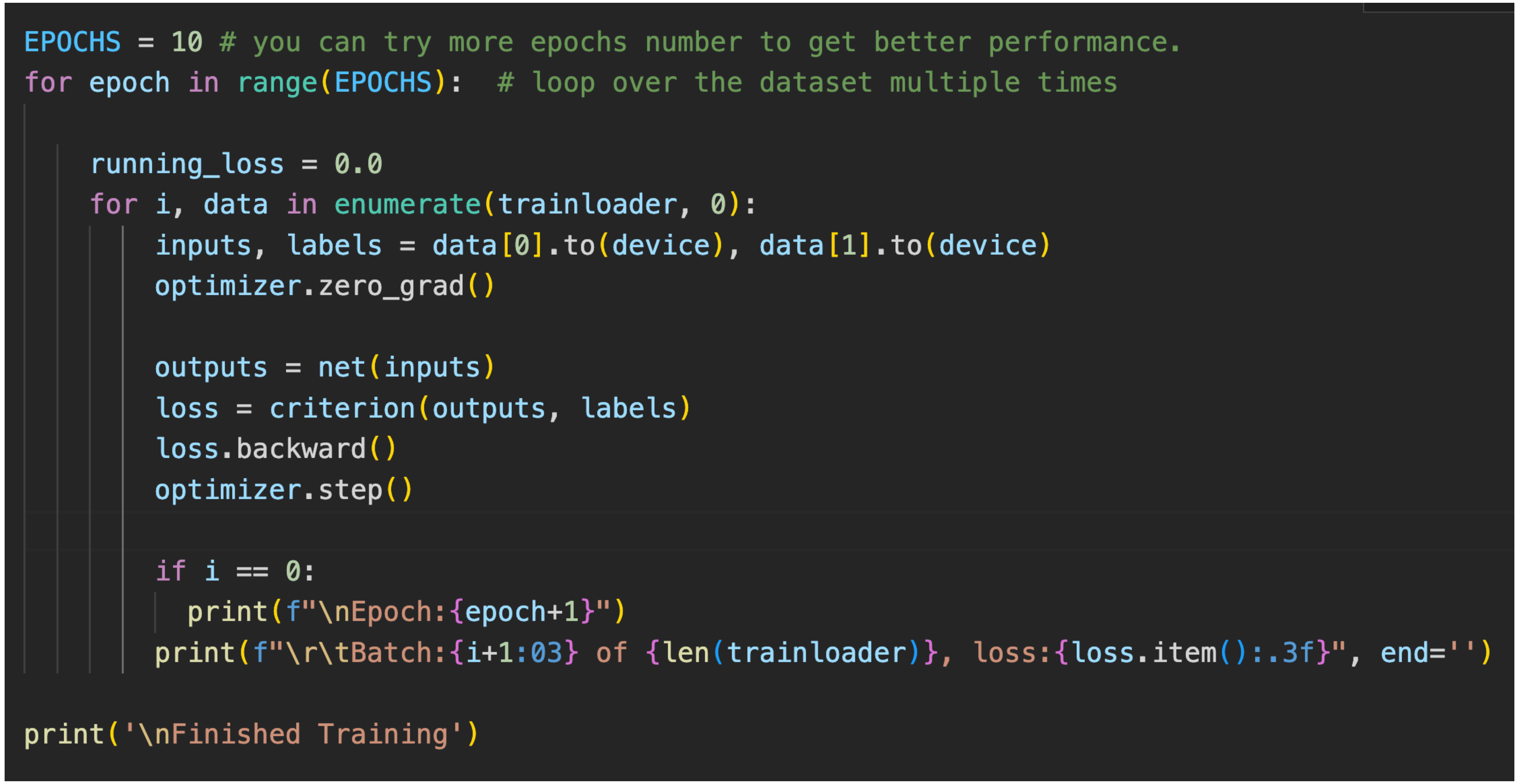
You can try other loss function like nn.L1Loss() or nn.MSELoss().



```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

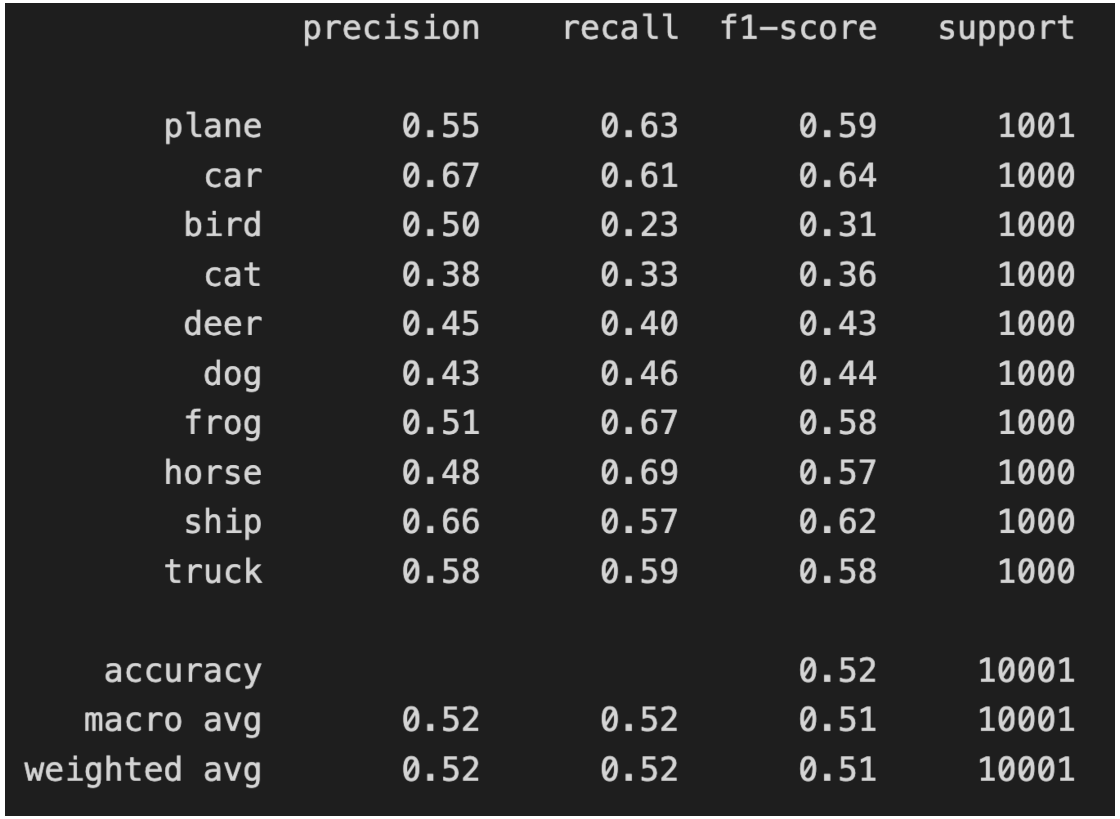
You may also try other optimizer like Adam, RMSprop, etc.

TRAINING!



```python
EPOCHS = 10 # you can try more epochs number to get better performance.
for epoch in range(EPOCHS):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()

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

        if i == 0:
          print(f"\nEpoch:{epoch+1}")
        print(f"\r\tBatch:{i+1:03} of {len(trainloader)}, loss:{loss.item():.3f}", end='')

print('\nFinished Training')
```

EVALUATION



```
              precision    recall  f1-score   support

       plane       0.55      0.63      0.59      1001
         car       0.67      0.61      0.64      1000
        bird       0.50      0.23      0.31      1000
         cat       0.38      0.33      0.36      1000
        deer       0.45      0.40      0.43      1000
         dog       0.43      0.46      0.44      1000
        frog       0.51      0.67      0.58      1000
       horse       0.48      0.69      0.57      1000
        ship       0.66      0.57      0.62      1000
       truck       0.58      0.59      0.58      1000

    accuracy                           0.52     10001
   macro avg       0.52      0.52      0.51     10001
weighted avg       0.52      0.52      0.51     10001
```

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f1\_score = \frac{2 \times precision \times recall}{precision + recall}$$