Trabalho 3 - Redes Neurais

Fazer um script em jupyter notebook para classificação de imagens.

Você deve plotar as curvas de erro de treino e validação.

Use a metodologia de validação Holdout.

Ao final, você deve gerar métricas para o conjunto de teste.

Métricas de classificação: acurácia e f1 por classe. (consulte o scikit-learn)

Imports

Primeiro é preciso importar as bibliotecas e funções a serem utilizadas.

In [1]:
```
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor
import torch.utils.data as data_utils
from torch import nn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classifica
```

Pegando os datasets

Utilizando o dataset CIFAR10 que possui link direto com o torchvision. É um dataset constituidos por imagens coloridas de veículos e animais onde existem 10 classes diferentes. O objetivo é criar um classificador a partir de uma CNN.

Link para o dataset

In [2]:
```
train_dataset_base = datasets.CIFAR10(
    root = "data",
    train = True,
    transform = ToTensor(),
)

test_dataset = datasets.CIFAR10(
    root = "data",
    train = False,
    transform = ToTensor()
)

print("Shape of the train_dataset: ", train_dataset_base.data.shape)
print("Shape of the test_dataset: ", test_dataset.data.shape)
print("Classes: ", train_dataset_base.classes)
```

```
Shape of the train_dataset:  (50000, 32, 32, 3)
Shape of the test_dataset:  (10000, 32, 32, 3)
Classes:  ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'hor
se', 'ship', 'truck']
```

Dividindo e Transformando o Dataset

In [3]:

```python
# Criando um dataset de validação a partir do treino
train_dataset, valid_dataset = data_utils.random_split(train_dataset_base, [4000

# Criando os dataloaders com o batch_size de 64
train_loader = data_utils.DataLoader(train_dataset, batch_size=64, shuffle=True)
valid_loader = data_utils.DataLoader(valid_dataset, batch_size=64, shuffle=True)
test_loader = data_utils.DataLoader(test_dataset, batch_size=64, shuffle=True)
```

Definindo o device

Como minha placa de vídeo é AMD não possuo Cuda.

In [4]:

```python
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('Using {} device'.format(device))
```

```
Using cpu device
```

Definindo a Rede Neural

Criei uma classe que herda nn.Module e defini as camadas da rede aqui. Usei um modelo base de CNN apenas para facilitar.

In [5]:

```python
class NeuralNetwork(nn.Module):

    def __init__(self):
        super(NeuralNetwork, 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(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Jogando a Rede para a CPU

In [6]:

```python
model = NeuralNetwork().to(device)
print(model)
```

```
NeuralNetwork(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=F
alse)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

Definindo a loss function e o otimizador

Learning rate inicial de 0.01 pois foi a com melhor desempenho.

In [7]:
```python
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Treinando e Validando a Rede

Funções para implementar o treino e a validação

Realizam os ajustes de pesos das redes e calculam a perda e acurácia do algoritmo

In [8]:
```python
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)

    loss_sum = 0
    i = 0
    for X, y in dataloader:
        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss_sum += loss.item() * len(X)

    avg_loss = loss_sum / size
    print(f"Train Error: Avg loss = {avg_loss:>8f}")
    return avg_loss
```

In [9]:
```python
def valid_loop(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    loss, correct = 0, 0

    with torch.no_grad():
      for X, y in dataloader:
          pred = model(X)
          loss += loss_fn(pred, y).item() * len(X)
          correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    avg_loss = loss / size
    acc      = correct / size
    print(f"Test Error: Accuracy = {100*acc:>0.1f}%, Avg loss = {avg_loss:>8f}")
```

```
    return avg_loss, acc
```

Loop do Treino

Utilizei 50 epochs pois o desempenho não melhorava muito depois disso. Salvei o modelo com melhor desempenho de perda na validação.

In [10]:
```
n_epoch = 50
loss_valid_history = []
acc_history = []
loss_train_history = []
min_loss_valid = np.Inf


for epoch in range(n_epoch):
    print(f"Epoch {epoch+1}\n-----------------------------------")
    loss_train = train_loop(train_loader, model, loss_fn, optimizer)
    loss_valid, accuracy = valid_loop(valid_loader, model, loss_fn)
    if  loss_valid <= min_loss_valid:
        print(f"Validation loss decreased ({min_loss_valid:>0.6f} --> {loss_vali
        min_loss_valid = loss_valid
        torch.save(model.state_dict(), 'models/best_model_trab3.pt')
    print("-----------------------------------\n")


    loss_train_history.append(loss_train)
    loss_valid_history.append(loss_valid)
    acc_history.append(accuracy)
```

```
Epoch 1
-----------------------------------
Train Error: Avg loss = 2.301983
Test Error: Accuracy = 12.0%, Avg loss = 2.299056
Validation loss decreased (inf --> 2.299056).  Saving model ...
-----------------------------------

Epoch 2
-----------------------------------
Train Error: Avg loss = 2.292651
Test Error: Accuracy = 16.4%, Avg loss = 2.277575
Validation loss decreased (2.299056 --> 2.277575).  Saving model ...
-----------------------------------

Epoch 3
-----------------------------------
Train Error: Avg loss = 2.209371
Test Error: Accuracy = 23.3%, Avg loss = 2.132837
Validation loss decreased (2.277575 --> 2.132837).  Saving model ...
-----------------------------------

Epoch 4
-----------------------------------
Train Error: Avg loss = 2.039506
Test Error: Accuracy = 28.0%, Avg loss = 1.994714
Validation loss decreased (2.132837 --> 1.994714).  Saving model ...
-----------------------------------

Epoch 5
-----------------------------------
Train Error: Avg loss = 1.946053
Test Error: Accuracy = 30.0%, Avg loss = 1.905836
Validation loss decreased (1.994714 --> 1.905836).  Saving model ...
-----------------------------------

Epoch 6
-----------------------------------
Train Error: Avg loss = 1.864871
Test Error: Accuracy = 34.8%, Avg loss = 1.819797
Validation loss decreased (1.905836 --> 1.819797).  Saving model ...
-----------------------------------

Epoch 7
-----------------------------------
Train Error: Avg loss = 1.784232
Test Error: Accuracy = 37.3%, Avg loss = 1.754910
Validation loss decreased (1.819797 --> 1.754910).  Saving model ...
-----------------------------------

Epoch 8
-----------------------------------
Train Error: Avg loss = 1.720086
Test Error: Accuracy = 39.7%, Avg loss = 1.687446
Validation loss decreased (1.754910 --> 1.687446).  Saving model ...
-----------------------------------

Epoch 9
-----------------------------------
Train Error: Avg loss = 1.655481
Test Error: Accuracy = 42.1%, Avg loss = 1.631101
```

```
Validation loss decreased (1.687446 --> 1.631101).  Saving model ...
-----------------------------------

Epoch 10
-----------------------------------
Train Error: Avg loss = 1.604435
Test Error: Accuracy = 43.7%, Avg loss = 1.574025
Validation loss decreased (1.631101 --> 1.574025).  Saving model ...
-----------------------------------

Epoch 11
-----------------------------------
Train Error: Avg loss = 1.560346
Test Error: Accuracy = 45.2%, Avg loss = 1.545079
Validation loss decreased (1.574025 --> 1.545079).  Saving model ...
-----------------------------------

Epoch 12
-----------------------------------
Train Error: Avg loss = 1.521284
Test Error: Accuracy = 45.1%, Avg loss = 1.538583
Validation loss decreased (1.545079 --> 1.538583).  Saving model ...
-----------------------------------

Epoch 13
-----------------------------------
Train Error: Avg loss = 1.493416
Test Error: Accuracy = 47.3%, Avg loss = 1.475665
Validation loss decreased (1.538583 --> 1.475665).  Saving model ...
-----------------------------------

Epoch 14
-----------------------------------
Train Error: Avg loss = 1.462759
Test Error: Accuracy = 48.1%, Avg loss = 1.459460
Validation loss decreased (1.475665 --> 1.459460).  Saving model ...
-----------------------------------

Epoch 15
-----------------------------------
Train Error: Avg loss = 1.436622
Test Error: Accuracy = 48.1%, Avg loss = 1.457191
Validation loss decreased (1.459460 --> 1.457191).  Saving model ...
-----------------------------------

Epoch 16
-----------------------------------
Train Error: Avg loss = 1.412990
Test Error: Accuracy = 48.9%, Avg loss = 1.432100
Validation loss decreased (1.457191 --> 1.432100).  Saving model ...
-----------------------------------

Epoch 17
-----------------------------------
Train Error: Avg loss = 1.393898
Test Error: Accuracy = 48.6%, Avg loss = 1.437844
-----------------------------------

Epoch 18
-----------------------------------
```

```
Train Error: Avg loss = 1.371760
Test Error: Accuracy = 50.9%, Avg loss = 1.391630
Validation loss decreased (1.432100 --> 1.391630).  Saving model ...
-----------------------------------

Epoch 19
-----------------------------------
Train Error: Avg loss = 1.354982
Test Error: Accuracy = 50.1%, Avg loss = 1.393070
-----------------------------------

Epoch 20
-----------------------------------
Train Error: Avg loss = 1.336949
Test Error: Accuracy = 51.7%, Avg loss = 1.366473
Validation loss decreased (1.391630 --> 1.366473).  Saving model ...
-----------------------------------

Epoch 21
-----------------------------------
Train Error: Avg loss = 1.319537
Test Error: Accuracy = 51.3%, Avg loss = 1.366601
-----------------------------------

Epoch 22
-----------------------------------
Train Error: Avg loss = 1.304318
Test Error: Accuracy = 51.3%, Avg loss = 1.373809
-----------------------------------

Epoch 23
-----------------------------------
Train Error: Avg loss = 1.288919
Test Error: Accuracy = 52.2%, Avg loss = 1.347676
Validation loss decreased (1.366473 --> 1.347676).  Saving model ...
-----------------------------------

Epoch 24
-----------------------------------
Train Error: Avg loss = 1.271301
Test Error: Accuracy = 52.1%, Avg loss = 1.335282
Validation loss decreased (1.347676 --> 1.335282).  Saving model ...
-----------------------------------

Epoch 25
-----------------------------------
Train Error: Avg loss = 1.258200
Test Error: Accuracy = 52.1%, Avg loss = 1.342228
-----------------------------------

Epoch 26
-----------------------------------
Train Error: Avg loss = 1.244255
Test Error: Accuracy = 52.9%, Avg loss = 1.327705
Validation loss decreased (1.335282 --> 1.327705).  Saving model ...
-----------------------------------

Epoch 27
-----------------------------------
Train Error: Avg loss = 1.229821
```

```
Test Error: Accuracy = 53.1%, Avg loss = 1.310165
Validation loss decreased (1.327705 --> 1.310165).  Saving model ...
-----------------------------------

Epoch 28
-----------------------------------
Train Error: Avg loss = 1.212977
Test Error: Accuracy = 52.9%, Avg loss = 1.316234
-----------------------------------

Epoch 29
-----------------------------------
Train Error: Avg loss = 1.201382
Test Error: Accuracy = 53.5%, Avg loss = 1.306172
Validation loss decreased (1.310165 --> 1.306172).  Saving model ...
-----------------------------------

Epoch 30
-----------------------------------
Train Error: Avg loss = 1.188452
Test Error: Accuracy = 54.3%, Avg loss = 1.281335
Validation loss decreased (1.306172 --> 1.281335).  Saving model ...
-----------------------------------

Epoch 31
-----------------------------------
Train Error: Avg loss = 1.175794
Test Error: Accuracy = 54.7%, Avg loss = 1.282952
-----------------------------------

Epoch 32
-----------------------------------
Train Error: Avg loss = 1.159714
Test Error: Accuracy = 55.0%, Avg loss = 1.275567
Validation loss decreased (1.281335 --> 1.275567).  Saving model ...
-----------------------------------

Epoch 33
-----------------------------------
Train Error: Avg loss = 1.150332
Test Error: Accuracy = 54.2%, Avg loss = 1.292983
-----------------------------------

Epoch 34
-----------------------------------
Train Error: Avg loss = 1.137209
Test Error: Accuracy = 54.8%, Avg loss = 1.282207
-----------------------------------

Epoch 35
-----------------------------------
Train Error: Avg loss = 1.123812
Test Error: Accuracy = 54.1%, Avg loss = 1.303904
-----------------------------------

Epoch 36
-----------------------------------
Train Error: Avg loss = 1.109850
Test Error: Accuracy = 55.0%, Avg loss = 1.270842
Validation loss decreased (1.275567 --> 1.270842).  Saving model ...
```

```
-----------------------------------

Epoch 37
-----------------------------------
Train Error: Avg loss = 1.100081
Test Error: Accuracy = 54.7%, Avg loss = 1.279844
-----------------------------------

Epoch 38
-----------------------------------
Train Error: Avg loss = 1.088623
Test Error: Accuracy = 56.0%, Avg loss = 1.261419
Validation loss decreased (1.270842 --> 1.261419).  Saving model ...
-----------------------------------

Epoch 39
-----------------------------------
Train Error: Avg loss = 1.076307
Test Error: Accuracy = 55.7%, Avg loss = 1.254097
Validation loss decreased (1.261419 --> 1.254097).  Saving model ...
-----------------------------------

Epoch 40
-----------------------------------
Train Error: Avg loss = 1.063146
Test Error: Accuracy = 56.2%, Avg loss = 1.248550
Validation loss decreased (1.254097 --> 1.248550).  Saving model ...
-----------------------------------

Epoch 41
-----------------------------------
Train Error: Avg loss = 1.056052
Test Error: Accuracy = 55.3%, Avg loss = 1.276219
-----------------------------------

Epoch 42
-----------------------------------
Train Error: Avg loss = 1.044621
Test Error: Accuracy = 55.7%, Avg loss = 1.269160
-----------------------------------

Epoch 43
-----------------------------------
Train Error: Avg loss = 1.031281
Test Error: Accuracy = 56.2%, Avg loss = 1.260565
-----------------------------------

Epoch 44
-----------------------------------
Train Error: Avg loss = 1.020718
Test Error: Accuracy = 55.7%, Avg loss = 1.270957
-----------------------------------

Epoch 45
-----------------------------------
Train Error: Avg loss = 1.009055
Test Error: Accuracy = 56.7%, Avg loss = 1.265475
-----------------------------------

Epoch 46
```

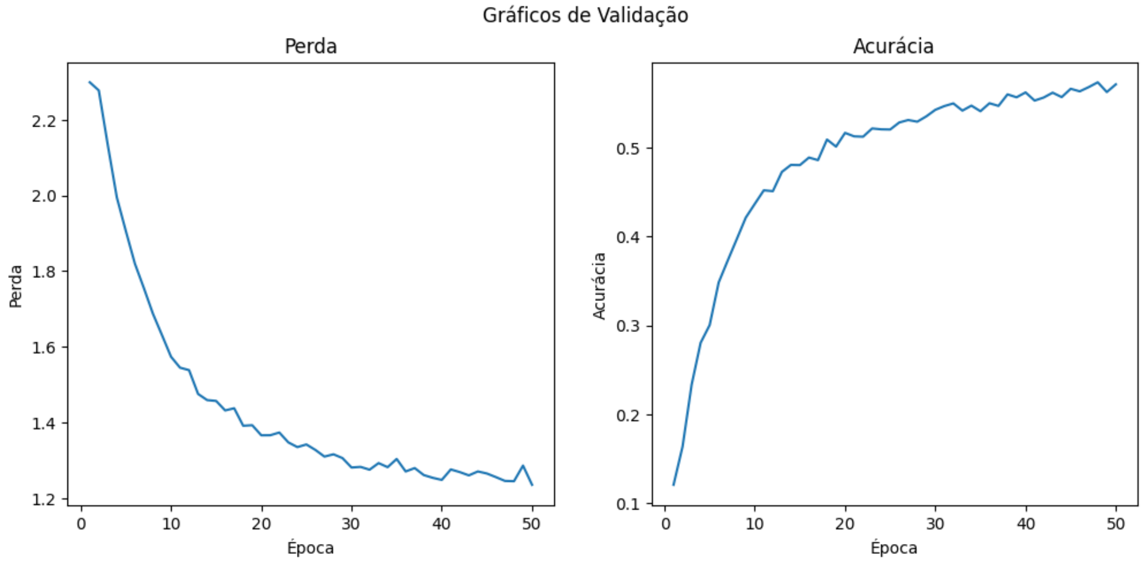
```
-----------------------------------
Train Error: Avg loss = 0.997246
Test Error: Accuracy = 56.4%, Avg loss = 1.255921
-----------------------------------

Epoch 47
-----------------------------------
Train Error: Avg loss = 0.990585
Test Error: Accuracy = 56.9%, Avg loss = 1.245865
Validation loss decreased (1.248550 --> 1.245865).  Saving model ...
-----------------------------------

Epoch 48
-----------------------------------
Train Error: Avg loss = 0.979505
Test Error: Accuracy = 57.4%, Avg loss = 1.245168
Validation loss decreased (1.245865 --> 1.245168).  Saving model ...
-----------------------------------

Epoch 49
-----------------------------------
Train Error: Avg loss = 0.967405
Test Error: Accuracy = 56.3%, Avg loss = 1.286391
-----------------------------------

Epoch 50
-----------------------------------
Train Error: Avg loss = 0.959653
Test Error: Accuracy = 57.2%, Avg loss = 1.235641
Validation loss decreased (1.245168 --> 1.235641).  Saving model ...
-----------------------------------
```

Desempenho no treino e na validação

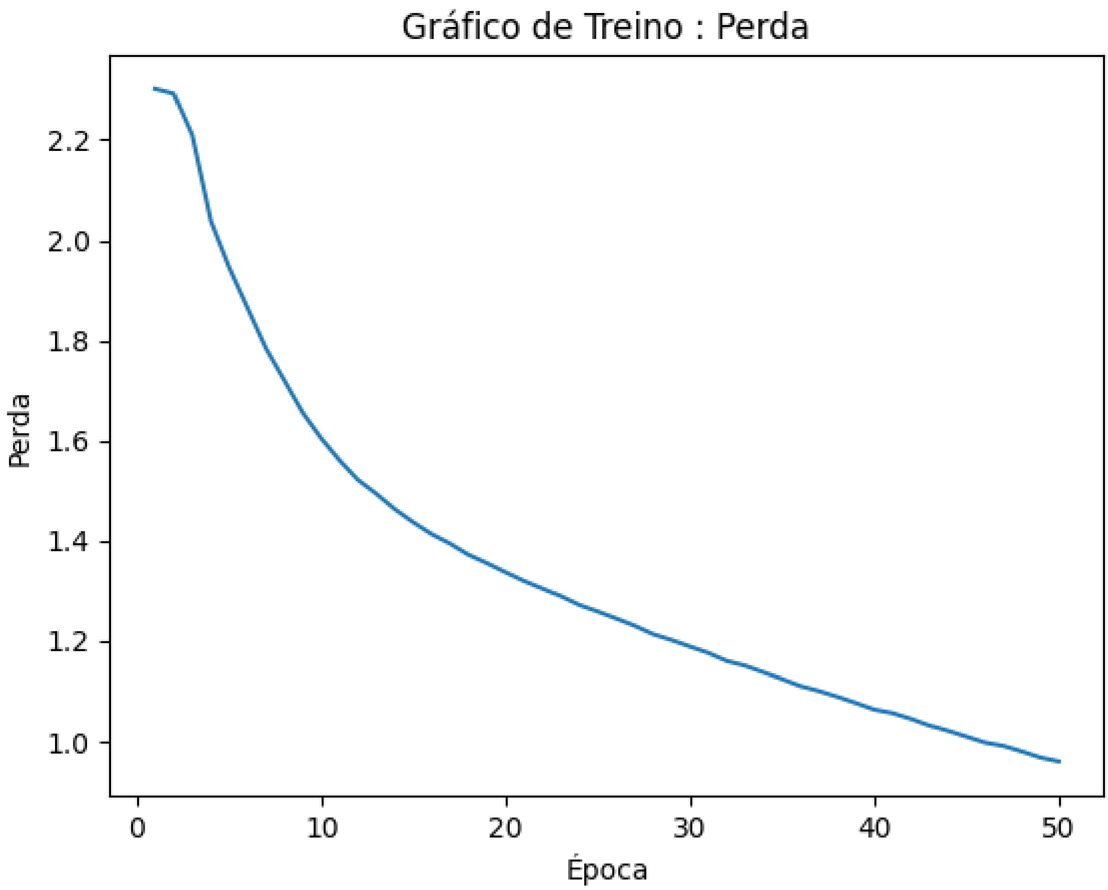
Gráficos que mostram como foi a redução da loss function de acordo com as epócas de treino, tanto na validação quanto do próprio treino.

In [11]:
```
epoch_vec = list(range(1, n_epoch+1))

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Gráficos de Validação')
fig.set_size_inches(12, 5)

ax1.set_title('Perda')
ax1.set_xlabel('Época')
ax1.set_ylabel('Perda')
ax1.plot(epoch_vec, loss_valid_history, label='loss')

ax2.set_title('Acurácia')
ax2.set_xlabel('Época')
ax2.set_ylabel('Acurácia')
ax2.plot(epoch_vec, acc_history, label='acuracy')

fig.show()
```

```
C:\Users\sergi\AppData\Local\Temp\ipykernel_15276\2620626654.py:17: UserWarnin
g: Matplotlib is currently using module://matplotlib_inline.backend_inline, whi
ch is a non-GUI backend, so cannot show the figure.
  fig.show()
```



In [12]:
```python
plt.title('Gráfico de Treino : Perda')
plt.plot(epoch_vec, loss_train_history, label='loss')
plt.xlabel('Época')
plt.ylabel('Perda')
plt.show()
```



Testando o Modelo

Pegando o Melhor modelo

In [13]:
```
model.load_state_dict(torch.load('models/best_model_trab3.pt'))
```

Out[13]:
```
<All keys matched successfully>
```

Função que executa os testes

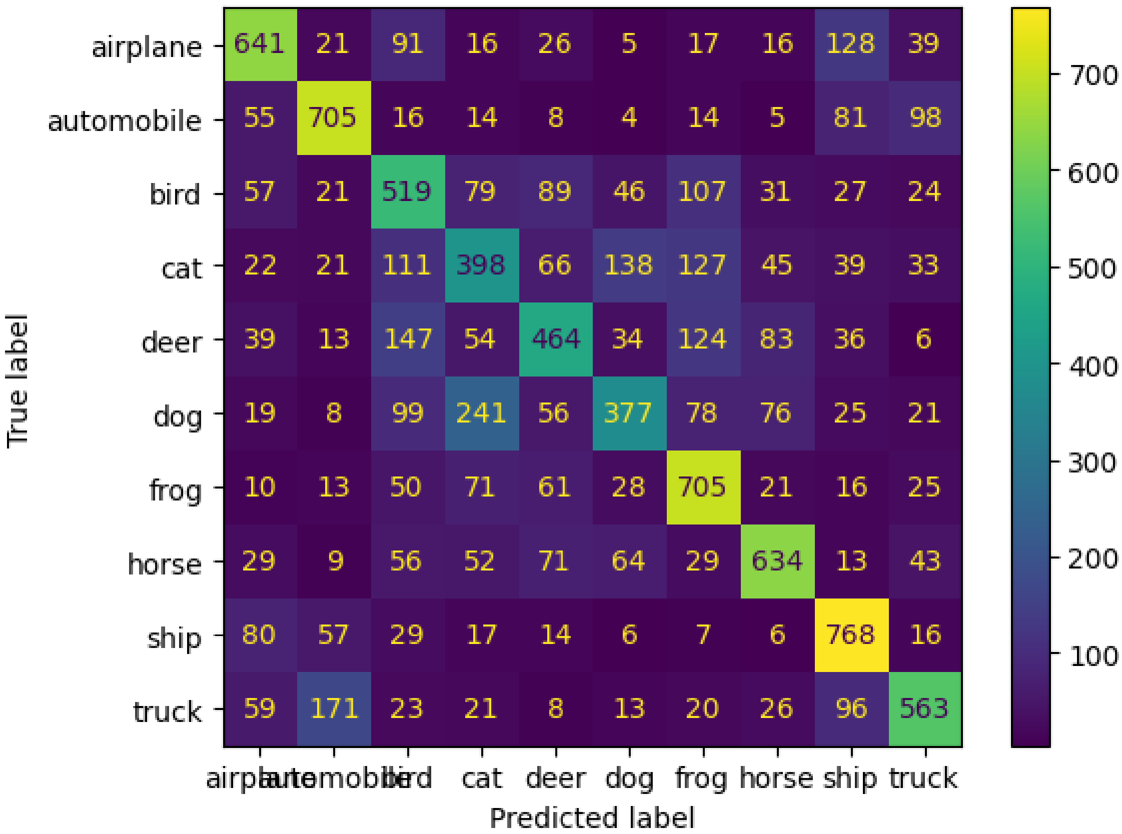
In [14]:
```
def test_loop(dataloader, model):
    y_pred = []
    y_true = []

    for X, y in dataloader:
        pred = model(X)
        pred = (torch.max(torch.exp(pred), 1)[1]).data.cpu().numpy()
        y_pred.extend(pred)

        y = y.data.cpu().numpy()
        y_true.extend(y)

    return y_true, y_pred
```

Resultados

O resultados encontrados são mostrados a seguir, para um problema de classificação com 10 classes uma acurácia acima de 55% deve ser considerada. Além disso, é possível perceber que apenas ao aplicar o modelo da CNN o desempenho entre o trabalho 1 que eu utilizei o mesmo dataset melhorou bastante.

In [15]:
```
y_true, y_pred = test_loop(test_loader, model)

cm = confusion_matrix(y_true, y_pred)
ConfusionMatrixDisplay(cm, display_labels = test_dataset.classes).plot()
report = classification_report(y_true, y_pred, output_dict=True)

for i in range(len(test_dataset.classes)):
    print(f"{test_dataset.classes[i]}:\n\tPrecision: {report[str(i)]['precision'

print(f"\nAcurácia: {report['accuracy']*100:>.02f}%")
mcc = matthews_corrcoef(y_true, y_pred)
print(f"\nMCC: {mcc:>.02f}")
```

```
airplane:
        Precision: 63.40%
        Recall: 64.10%
        F1-Score: 63.75%
automobile:
        Precision: 67.85%
        Recall: 70.50%
        F1-Score: 69.15%
bird:
        Precision: 45.49%
        Recall: 51.90%
        F1-Score: 48.48%
cat:
        Precision: 41.33%
        Recall: 39.80%
        F1-Score: 40.55%
deer:
        Precision: 53.77%
        Recall: 46.40%
        F1-Score: 49.81%
dog:
        Precision: 52.73%
        Recall: 37.70%
        F1-Score: 43.97%
frog:
        Precision: 57.41%
        Recall: 70.50%
        F1-Score: 63.29%
horse:
        Precision: 67.23%
        Recall: 63.40%
        F1-Score: 65.26%
ship:
        Precision: 62.49%
        Recall: 76.80%
        F1-Score: 68.91%
truck:
        Precision: 64.86%
        Recall: 56.30%
        F1-Score: 60.28%

Acurácia: 57.74%

MCC: 0.53
```



True label
Predicted label
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck