Projeto

December 19, 2021

1 Projeto

1.1 Configurações Iniciais

1.1.1 Bibliotecas

Pytorch

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, random_split,␣
↪SubsetRandomSampler, WeightedRandomSampler
```

Torch Vision

[2]:
```
import torchvision
from torchvision import models
from torchvision.transforms import ToTensor
from torchvision.utils import make_grid
from torchvision import transforms
from torchvision.utils import save_image
from torchvision.datasets import CIFAR10
from torchvision.datasets import CIFAR100
```

Extras

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
%matplotlib inline
```

1.1.2 Configurações Globais

[4]:
```
torch.manual_seed(7)
```

[4]:
```
<torch._C.Generator at 0x7f3c26a30ab0>
```

1.1.3 Funções Auxiliares

[5]:
```
def plot_data(batch,classes,limit = 20,figsize = (10,10),n = 25, nrow = 5):
  images,labels = batch
  fig,ax = plt.subplots(figsize = figsize)
  ax.imshow(make_grid(images[:limit],nrow = 5).permute(1,2,0))
  #title = [classes[int(i)] for i in labels]
  #ax.set_title(str(title))
  print(classes[int(i)] for i in labels)
  plt.show()
```

[6]:
```
def get_batch(dataloader):
  for batch in dataloader:
    return batch
```

[7]:
```
def get_device():
  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  #print("Device: ", device)
  return device
```

[8]:
```
def export_to_onnx(model):
  dummy_dataloader = DataLoader(torch.randn(1, 3, 32, 32),batch_size = 1)
  dummy_input = next(iter(dummy_dataloader))
  dummy_input = dummy_input.to(get_device())
  torch.onnx.export(model,dummy_input,'model.onnx')
```

[9]:
```
def predict_image(model,image,classes = None):
  model.eval()
  image = image.to(get_device())
  output = model(image.unsqueeze(0))
  _,prediction = torch.max(output,1)
  if classes == None:
    return prediction.item()
  return classes[int(prediction.item())]
```

[10]:
```
def predict(model,data):
  prediction = [predict_image(model,image) for image,label in data]
  return prediction
```

[11]:
```
def fit(model,dl_train,num_epochs = 5, loss_function = nn.CrossEntropyLoss(),␣
↪lr = 1e-2, optimizer = None,steps = 2000, tupla=False):
```

```
  device = get_device()
  model = model.to(device)
  if optimizer == None:
    optimizer = torch.optim.SGD(model.parameters(),lr = lr)

  model.train() # sets the model for training mode
  for epoch in range(num_epochs):
    running_loss = 0.0
    for i, (data,targets) in enumerate(dl_train):
      data = data.to(device) # send the images to the GPU (if available)
      targets = targets.to(device) # send the labels to the GPU (if available)
      optimizer.zero_grad()

      outputs = model(data) # get the predicted results
      loss = loss_function(outputs,targets) #calculates the loss
      loss.backward() # calculates the derivative of the loss
      optimizer.step() # updates the parameters
      running_loss += loss.item()

      if (i+1) % steps == 0:
        print("Epoch: ", "[" + str(epoch+1) + "/" + str(num_epochs)+ "]" + ":",␣
↪"Loss :", running_loss/steps, "Step:", i+1)
        running_loss = 0.0
  print('The Training is Finished')
```

[12]:
```
def test(model,dl_test):
  device = get_device()
  correct = 0
  total = 0
  model.eval() #sets model to evaluation model (the dropout layer works␣
↪differently in evaluation)
  with torch.no_grad():
    for data,labels in dl_test:
      data = data.to(device) # sends data to GPU (if available)
      labels = labels.to(device) # sends labels to GPU (if available)
      outputs = model(data) # Gets the outputs of the model
      _,predicted = torch.max(outputs.data,1) #Gets which numbers where␣
↪predicted by the model
      total += labels.size(0)
      correct += (predicted == labels).sum().item()
  print("Accuracy = ",correct,'/', total,(correct/total)*100,'%')
```

[13]:
```
def plot_confusion_matrix(y_test,y_pred, classes, title = 'Confusion␣
↪Matrix',figsize = (10,10), fontsize = 20,fmt = 'd'):
  cm = confusion_matrix(y_test,y_pred)
  plt.figure(figsize = figsize)
  plt.title(title,fontsize = fontsize + int(0.5*fontsize))
```

```
  s = sns.heatmap(cm, annot = True, xticklabels = classes, yticklabels =␣
↪classes,fmt = fmt)
  s.set_xlabel('Predicted Label',fontsize = fontsize)
  s.set_ylabel('True Label', fontsize = fontsize)
  plt.show()
```

[14]:
```
def show_image(image):
  plt.imshow(image.view(32,32,3))
```

[15]:
```
def get_image(dataset):
  for image,label in dataset:
    return (image,label)
```

1.2 Cifar10

1.2.1 Transformações

[16]:
```
transform_train = transforms.Compose([#transforms.Resize((227,227)),
                                      transforms.RandomHorizontalFlip(p=0.7),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.4914,0.4822,0.
↪4465], std=[0.247,0.243,0.261])
                                      ])
transform_test = transforms.Compose([#transforms.Resize((227,227)),
                                     transforms.ToTensor(),
                                     transforms.Normalize(mean=[0.4914,0.4822,0.
↪4465], std=[0.247,0.243,0.261])
                                     ])
```

[17]:
```
%cd data
!rm -rf *
%cd ..
!ls
```

```
[Errno 2] No such file or directory: 'data'
/content
/
bin      datalab  home   lib64  opt         root  srv               tmp    var
boot     dev      lib    media  proc        run   sys               tools
content  etc      lib32  mnt    python-apt  sbin  tensorflow-1.15.2  usr
```

[18]:
```
ds_train = CIFAR10('data/', train = True, download = True,transform =␣
↪transform_train)
ds_test =  CIFAR10('data/', train = False, download = True, transform =␣
↪transform_test)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
```

```
data/cifar-10-python.tar.gz

  0%|          | 0/170498071 [00:00<?, ?it/s]

Extracting data/cifar-10-python.tar.gz to data/
Files already downloaded and verified
```

[19]:
```
classes = ds_train.classes
print(classes)
```

```
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
'ship', 'truck']
```

[20]:
```
print(ds_train)
```

```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: data/
    Split: Train
    StandardTransform
Transform: Compose(
               RandomHorizontalFlip(p=0.7)
               ToTensor()
               Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.247, 0.243,
0.261])
           )
```

[21]:
```
print(ds_train.data.shape)
```

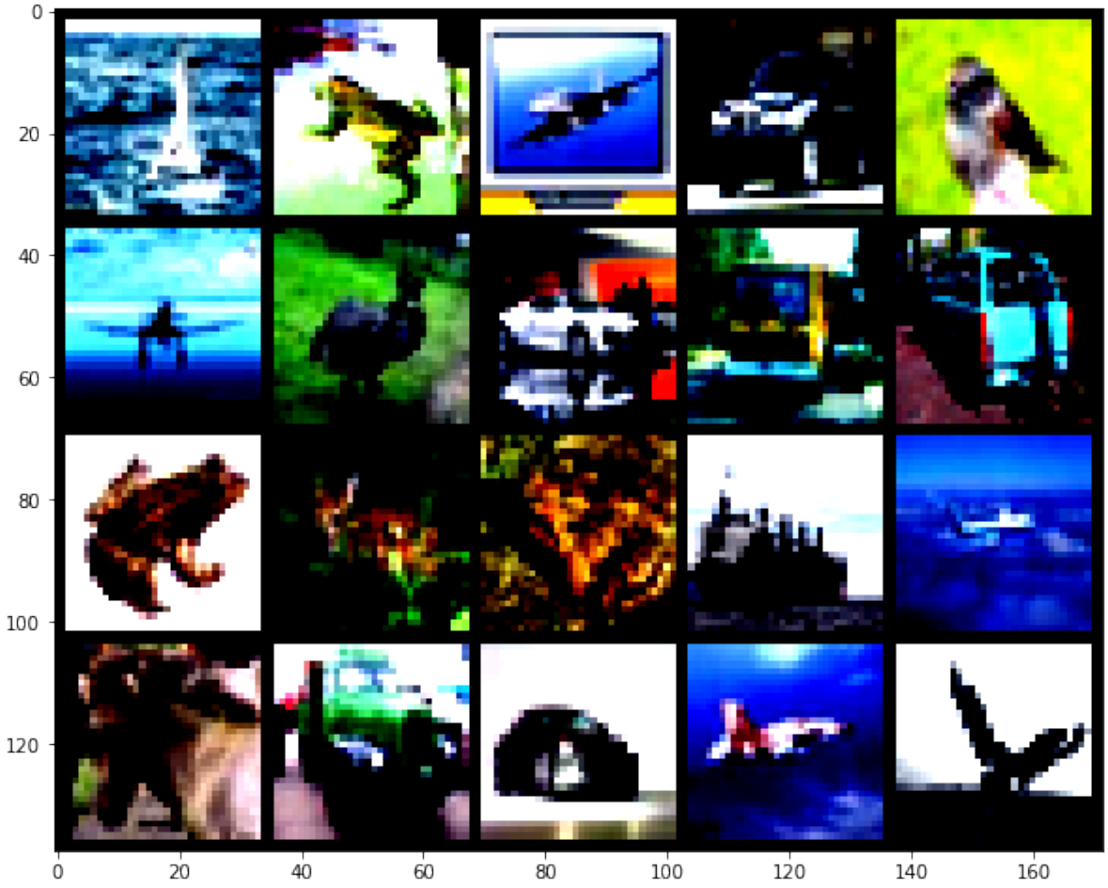
```
(50000, 32, 32, 3)
```

1.2.2 Dataloaders

[22]:
```
batch_size = 20
dl_train = DataLoader(ds_train, batch_size = batch_size, shuffle = True)
dl_test =  DataLoader(ds_test, batch_size = batch_size, shuffle = True)
```

[23]:
```
batch = get_batch(dl_train)
plot_data(batch,classes)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).

<generator object plot_data.<locals>.<genexpr> at 0x7f3c138d7bd0>
```



1.2.3 Modelos

O Nosso Modelo

[24]:
```
class CNN10(nn.Module):
  def __init__(self):
    super(CNN10,self).__init__()
    self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 24, kernel_size = 3)
    self.conv1_bn = nn.BatchNorm2d(24)
    self.conv2 = nn.Conv2d(in_channels = 24, out_channels = 72, kernel_size = 3)
    self.conv2_bn = nn.BatchNorm2d(72)
    self.conv3 = nn.Conv2d(in_channels = 72, out_channels = 72, kernel_size = 3)
    self.conv3_bn = nn.BatchNorm2d(72)
    self.conv4 = nn.Conv2d(in_channels = 72, out_channels = 216, kernel_size =␣
 ↪3)
    self.conv4_bn = nn.BatchNorm2d(216)
    self.pool = nn.MaxPool2d(kernel_size = 2)
```

```
    self.dropout = nn.Dropout(p = 0.2)
    self.fc1 = nn.Linear(216*5*5, 600)
    self.fc1_bn = nn.BatchNorm1d(600)
    self.fc2 = nn.Linear(600,200)
    self.fc2_bn = nn.BatchNorm1d(200)
    self.fc3 = nn.Linear(200,10)

  def forward(self, x):
    x = self.conv1(x)
    x = F.relu(self.conv1_bn(x))
    x = self.conv2(x)
    x = F.relu(self.conv2_bn(x))
    x = self.pool(x)

    x = self.conv3(x)
    x = F.relu(self.conv3_bn(x))
    x = self.conv4(x)
    x = F.relu(self.conv4_bn(x))
    x = self.pool(x)

    x = x.view(-1,216*5*5)

    x = F.relu(self.fc1(x))
    #x = F.relu(self.fc1_bn(x))
    x = self.dropout(x)
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    #x = self.fc2_bn(x)

    x = F.relu(self.fc3(x))
    return F.log_softmax(x,dim=1)
```

[25]:
```
cnn10 = CNN10()
print(cnn10)
cnn10 = cnn10.to(get_device())
```

```
CNN10(
  (conv1): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1))
  (conv1_bn): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv2): Conv2d(24, 72, kernel_size=(3, 3), stride=(1, 1))
  (conv2_bn): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv3): Conv2d(72, 72, kernel_size=(3, 3), stride=(1, 1))
  (conv3_bn): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv4): Conv2d(72, 216, kernel_size=(3, 3), stride=(1, 1))
```

```
  (conv4_bn): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=5400, out_features=600, bias=True)
  (fc1_bn): BatchNorm1d(600, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc2): Linear(in_features=600, out_features=200, bias=True)
  (fc2_bn): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc3): Linear(in_features=200, out_features=10, bias=True)
)
```

LeNet

[26]:
```
class LeNet10(nn.Module):
  def __init__(self):
    super(LeNet10,self).__init__()
    self.conv1 = nn.Conv2d(3, 6, kernel_size=5, padding=2)
    self.avgPool = nn.AvgPool2d(kernel_size=2, stride=2)
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
    self.flatten = nn.Flatten()
    self.fc1 = nn.Linear(576, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

  def forward(self, x):
    x = F.sigmoid(self.conv1(x))
    x = self.avgPool(x)
    x = F.sigmoid(self.conv2(x))
    x = self.avgPool(x)
    x = self.flatten(x)
    x = F.sigmoid(self.fc1(x))
    x = F.sigmoid(self.fc2(x))
    x = self.fc3(x)
    #print(x.shape)
    return x
```

[27]:
```
lenet10 = LeNet10()
print(lenet10)
lenet10 = lenet10.to(get_device())
```

```
LeNet10(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (avgPool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (flatten): Flatten(start_dim=1, end_dim=-1)
```

```
  (fc1): Linear(in_features=576, 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)
)
```

AlexNet

[28]:
```
class AlexNet10(nn.Module):
  def __init__(self):
    super(AlexNet10,self).__init__()
    self.conv1 = nn.Conv2d(3, 96, kernel_size=11, padding=1)
    self.conv2 = nn.Conv2d(96, 256, kernel_size=5, padding=2)
    self.conv3 = nn.Conv2d(256, 384, kernel_size=3, padding=1)
    self.conv4 = nn.Conv2d(384, 384, kernel_size=3, padding=1)
    self.conv5 = nn.Conv2d(384, 256, kernel_size=3, padding=1)
    self.pool = nn.MaxPool2d(kernel_size=3, stride=2)
    self.dropout = nn.Dropout(p=0.5)
    self.fc1 = nn.Linear(256*2*2, 4096)
    self.fc2 = nn.Linear(4096, 4096)
    self.fc3 = nn.Linear(4096, 10)

  def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool(x)

    x = F.relu(self.conv2(x))
    x = self.pool(x)

    x = F.relu(self.conv3(x))
    x = F.relu(self.conv4(x))
    x = F.relu(self.conv5(x))
    x = self.pool(x)
    #print(x.shape)
    x = x.view(-1,256*2*2)

    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    x = self.fc3(x)

    return x
```

[29]:
```
alexnet10 = AlexNet10()
print(alexnet10)
alexnet10 = alexnet10.to(get_device())
```

```
AlexNet10(
```

```
  (conv1): Conv2d(3, 96, kernel_size=(11, 11), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv3): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=1024, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=4096, bias=True)
  (fc3): Linear(in_features=4096, out_features=10, bias=True)
)
```

Treino

O Nosso Modelo

[30]:
```
fit(cnn10,dl_train,num_epochs = 10)
```

```
Epoch:  [1/10]: Loss : 1.4822540675699711 Step: 2000
Epoch:  [2/10]: Loss : 1.0126769013553858 Step: 2000
Epoch:  [3/10]: Loss : 0.8352408567816019 Step: 2000
Epoch:  [4/10]: Loss : 0.7342948031947016 Step: 2000
Epoch:  [5/10]: Loss : 0.6635638531036675 Step: 2000
Epoch:  [6/10]: Loss : 0.5987266487404704 Step: 2000
Epoch:  [7/10]: Loss : 0.5532183033265173 Step: 2000
Epoch:  [8/10]: Loss : 0.5123321064449847 Step: 2000
Epoch:  [9/10]: Loss : 0.4725880506336689 Step: 2000
Epoch:  [10/10]: Loss : 0.43718925583176316 Step: 2000
The Training is Finished
```

LeNet

[31]:
```
fit(lenet10,dl_train,num_epochs = 10)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

```
Epoch:  [1/10]: Loss : 2.3053241349458693 Step: 2000
Epoch:  [2/10]: Loss : 2.305205782413483 Step: 2000
Epoch:  [3/10]: Loss : 2.305276039838791 Step: 2000
Epoch:  [4/10]: Loss : 2.3051891129016875 Step: 2000
Epoch:  [5/10]: Loss : 2.304678301334381 Step: 2000
Epoch:  [6/10]: Loss : 2.304960354208946 Step: 2000
Epoch:  [7/10]: Loss : 2.305099113821983 Step: 2000
Epoch:  [8/10]: Loss : 2.304563944339752 Step: 2000
Epoch:  [9/10]: Loss : 2.3047928570508955 Step: 2000
```

```
Epoch:  [10/10]: Loss : 2.3043586086034775 Step: 2000
The Training is Finished
```

AlexNet

[32]:
```
fit(alexnet10,dl_train,num_epochs = 10)
```

```
Epoch:  [1/10]: Loss : 2.146364500939846 Step: 2000
Epoch:  [2/10]: Loss : 1.6609799524247646 Step: 2000
Epoch:  [3/10]: Loss : 1.3868987611830235 Step: 2000
Epoch:  [4/10]: Loss : 1.196623044550419 Step: 2000
Epoch:  [5/10]: Loss : 1.0528814872354268 Step: 2000
Epoch:  [6/10]: Loss : 0.9374853228032589 Step: 2000
Epoch:  [7/10]: Loss : 0.8386177546828986 Step: 2000
Epoch:  [8/10]: Loss : 0.7579978749752044 Step: 2000
Epoch:  [9/10]: Loss : 0.6910740147531033 Step: 2000
Epoch:  [10/10]: Loss : 0.6251023647859693 Step: 2000
The Training is Finished
```

Salvando os Modelos

[33]:
```
torch.save(cnn10,'cnn10.pkl')
torch.save(lenet10,'lenet10.pkl')
torch.save(alexnet10,'alexnet10.pkl')
```

Test

[34]:
```
cnn10 = torch.load('cnn10.pkl')
lenet10 = torch.load('lenet10.pkl')
alexnet10 = torch.load('alexnet10.pkl')
```

Nosso Modelo

[35]:
```
test(cnn10,dl_test) # 10 epochs should be used at least
```

```
Accuracy =  8110 / 10000 81.10000000000001 %
```

LeNet

[36]:
```
test(lenet10,dl_test)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
Accuracy =  1000 / 10000 10.0 %
```

AlexNet

[37]:
```
test(alexnet10,dl_test)
```

```
Accuracy =  7421 / 10000 74.21 %
```

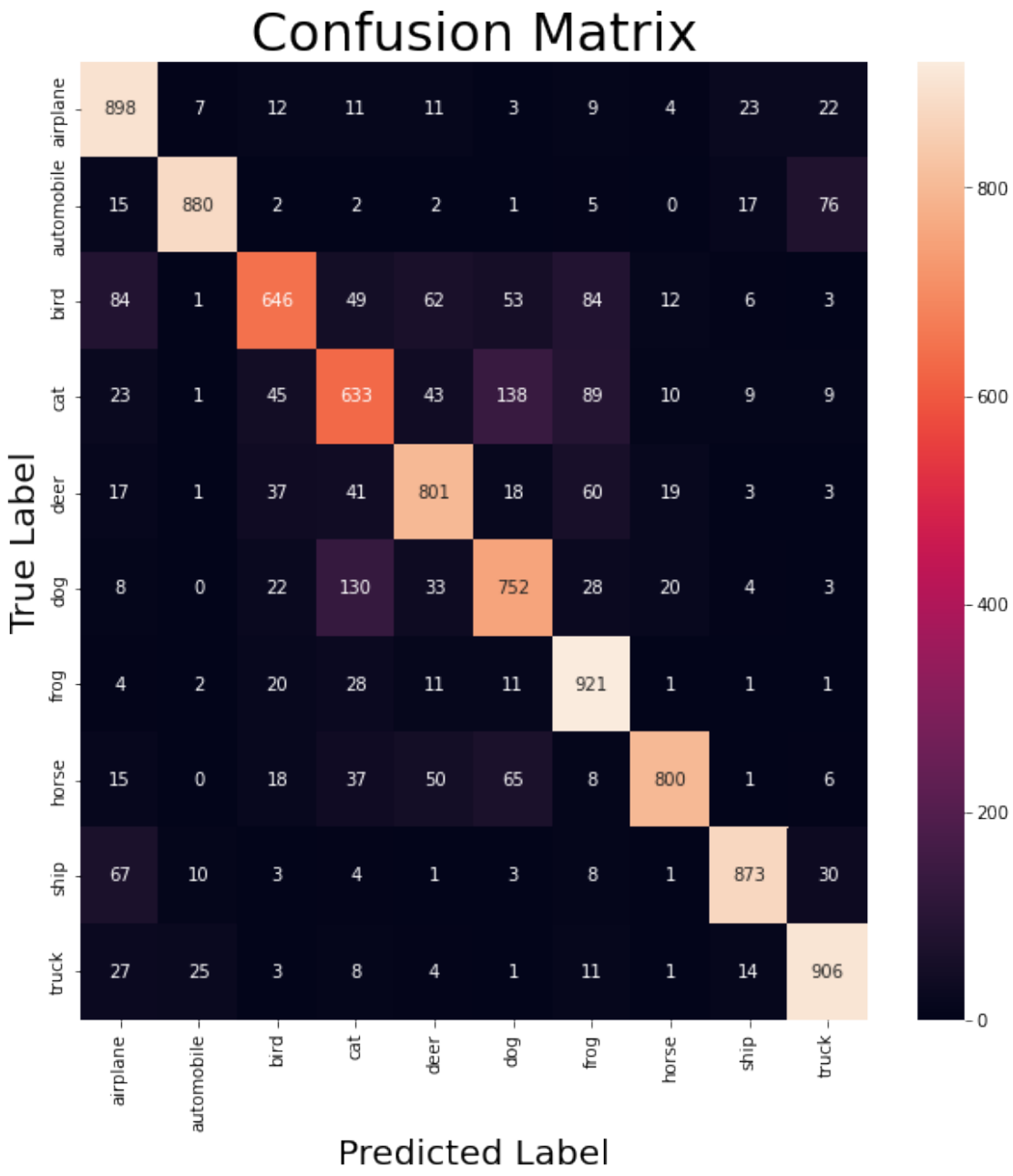
Métricas de Desempenho Nosso Modelo

[38]:
```
y_pred = predict(cnn10,ds_test)
y_test = [label for image,label in ds_test]
```

[39]:
```
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

    airplane       0.78      0.90      0.83      1000
  automobile       0.95      0.88      0.91      1000
        bird       0.80      0.65      0.71      1000
         cat       0.67      0.63      0.65      1000
        deer       0.79      0.80      0.79      1000
         dog       0.72      0.75      0.74      1000
        frog       0.75      0.92      0.83      1000
       horse       0.92      0.80      0.86      1000
        ship       0.92      0.87      0.89      1000
       truck       0.86      0.91      0.88      1000

    accuracy                           0.81     10000
   macro avg       0.82      0.81      0.81     10000
weighted avg       0.82      0.81      0.81     10000
```

[40]:
```
plot_confusion_matrix(y_test,y_pred,classes)
```



LeNet

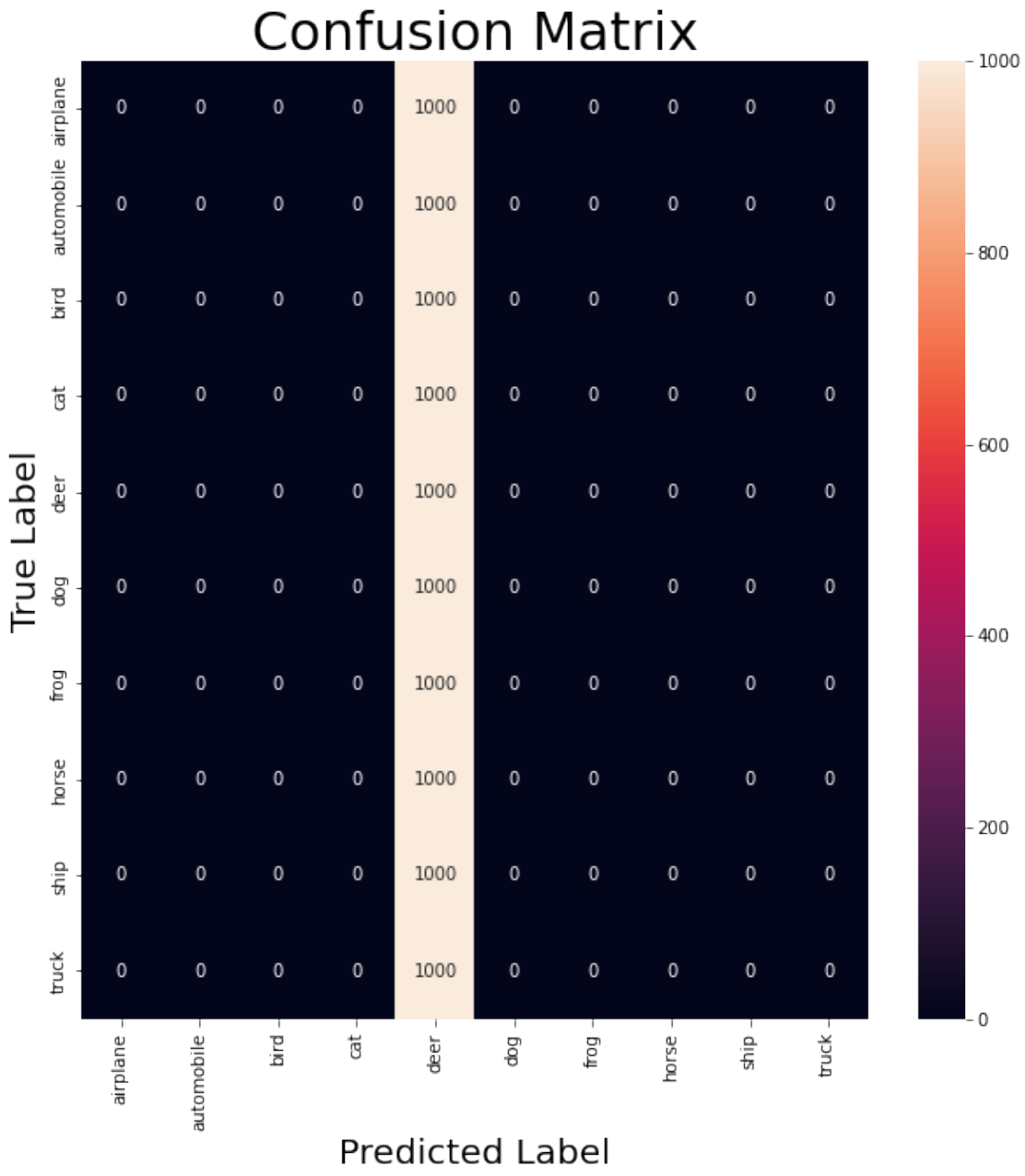
[41]:
```
y_pred = predict(lenet10,ds_test)
y_test = [label for image,label in ds_test]
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

[42]:
```
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

    airplane       0.00      0.00      0.00      1000
  automobile       0.00      0.00      0.00      1000
        bird       0.00      0.00      0.00      1000
         cat       0.00      0.00      0.00      1000
        deer       0.10      1.00      0.18      1000
         dog       0.00      0.00      0.00      1000
        frog       0.00      0.00      0.00      1000
       horse       0.00      0.00      0.00      1000
        ship       0.00      0.00      0.00      1000
       truck       0.00      0.00      0.00      1000

    accuracy                           0.10     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.10      0.02     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[43]:
```
plot_confusion_matrix(y_test,y_pred,classes)
```



AlexNet

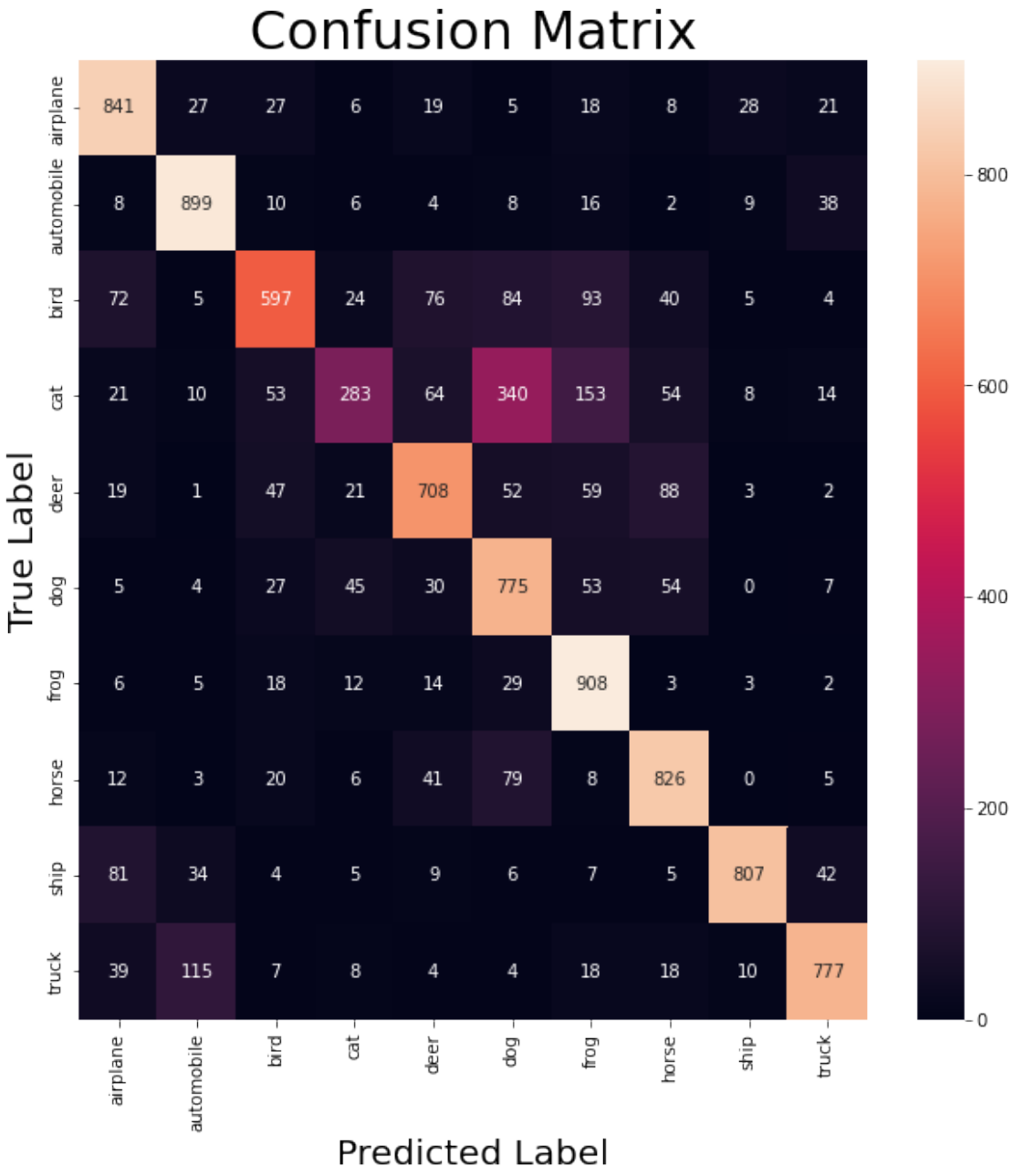
[44]:
```
y_pred = predict(alexnet10,ds_test)
y_test = [label for image,label in ds_test]
```

[45]:
```
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

    airplane       0.76      0.84      0.80      1000
  automobile       0.82      0.90      0.85      1000
        bird       0.74      0.60      0.66      1000
         cat       0.68      0.28      0.40      1000
        deer       0.73      0.71      0.72      1000
         dog       0.56      0.78      0.65      1000
        frog       0.68      0.91      0.78      1000
       horse       0.75      0.83      0.79      1000
        ship       0.92      0.81      0.86      1000
       truck       0.85      0.78      0.81      1000

    accuracy                           0.74     10000
   macro avg       0.75      0.74      0.73     10000
weighted avg       0.75      0.74      0.73     10000
```

[46]:
```
plot_confusion_matrix(y_test,y_pred,classes)
```



1.3 Cifar 100

1.3.1 Análise Exploratória de Dados

Quantidade de Imagens Por Classes

[ ]:
```
%cd data
!rm -rf *
%cd ..
```

```
!ls
```

```
[Errno 2] No such file or directory: 'data'
/content
/
bin      datalab  home   lib64  opt         root  srv                 tmp    var
boot     dev      lib    media  proc        run   sys                 tools
content  etc      lib32  mnt    python-apt  sbin  tensorflow-1.15.2  usr
```

[ ]:
```
ds_train = CIFAR100('data/', train = True, download = True,transform =␣
↪transforms.ToTensor())
ds_test =  CIFAR100('data/', train = False, download = True, transform =␣
↪transforms.ToTensor())
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to
data/cifar-100-python.tar.gz

  0%|          | 0/169001437 [00:00<?, ?it/s]

Extracting data/cifar-100-python.tar.gz to data/
Files already downloaded and verified
```

Classes

[ ]:
```
classes = ds_train.classes
```

[ ]:
```
print(ds_train.classes)
```

```
['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle',
'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',
'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',
'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster', 'house',
'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard',
'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse', 'mushroom',
'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pear', 'pickup_truck',
'pine_tree', 'plain', 'plate', 'poppy', 'porcupine', 'possum', 'rabbit',
'raccoon', 'ray', 'road', 'rocket', 'rose', 'sea', 'seal', 'shark', 'shrew',
'skunk', 'skyscraper', 'snail', 'snake', 'spider', 'squirrel', 'streetcar',
'sunflower', 'sweet_pepper', 'table', 'tank', 'telephone', 'television',
'tiger', 'tractor', 'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale',
'willow_tree', 'wolf', 'woman', 'worm']
```

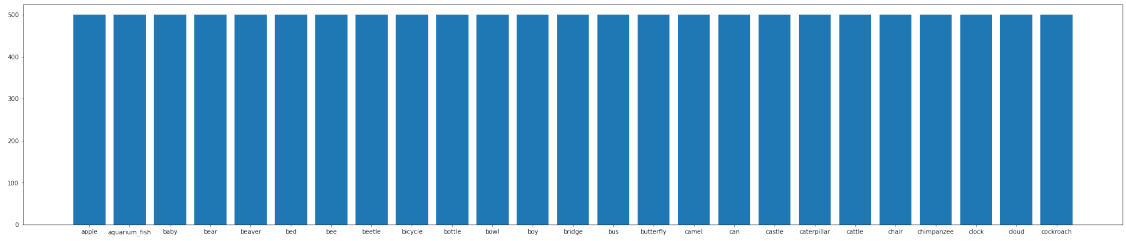
Dataset de Treino

[ ]:
```
data = {}

for image,label in ds_train:
  if label not in data:
    data[label] = []
```

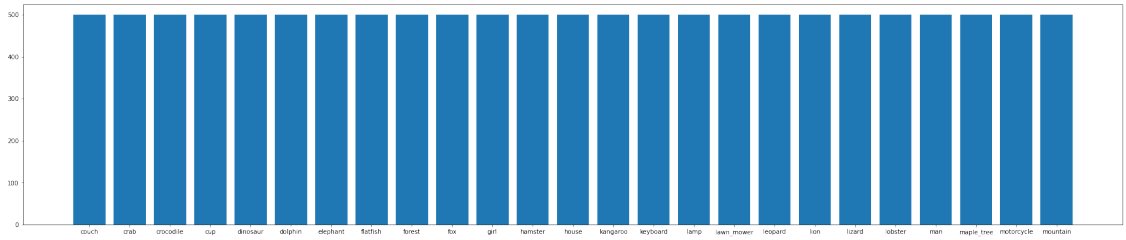
```
  data[label].append(image)
```

[ ]:
```
qtd = [len(data[i]) for i in range(100)]
```

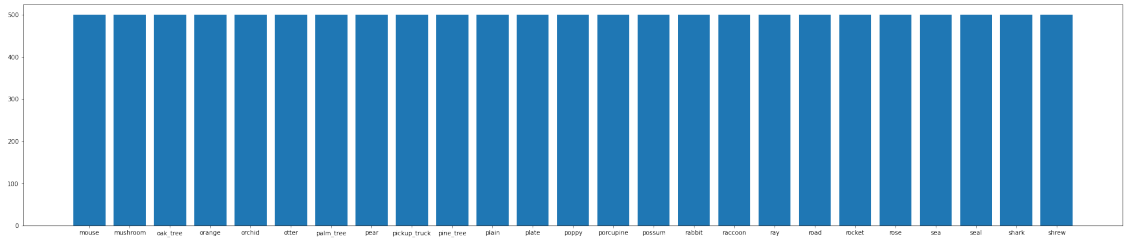
[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[0:25], qtd[0:25])
plt.show()
```



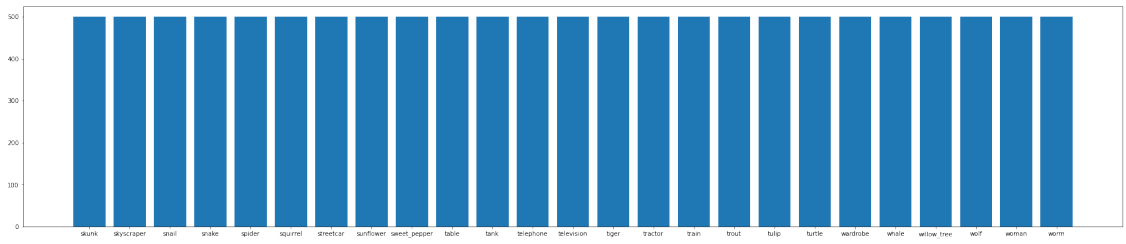
[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[25:50], qtd[25:50])
plt.show()
```



[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[50:75], qtd[50:75])
plt.show()
```



[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[75:100], qtd[75:100])
plt.show()
```

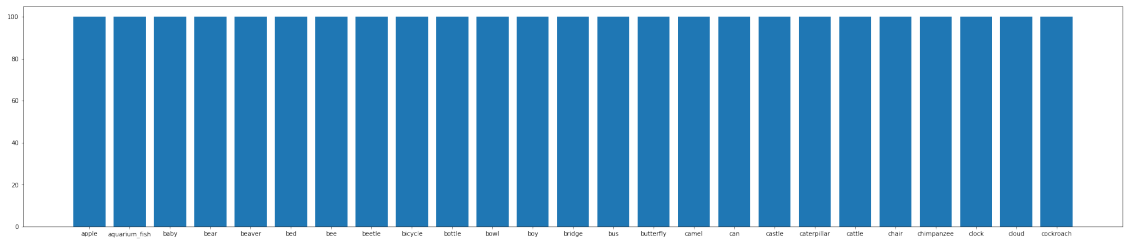


Dataset de Teste

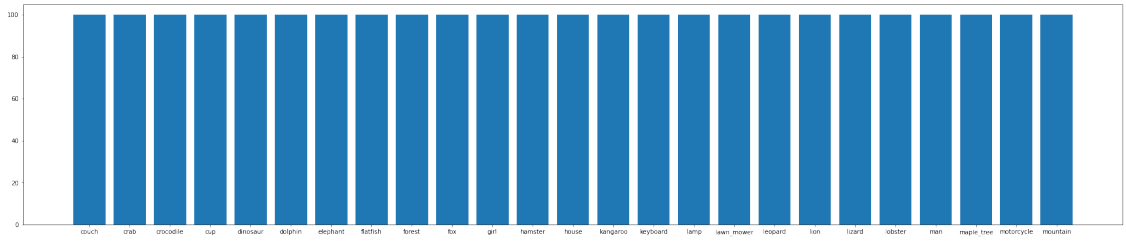
[ ]:
```
data_test = {}

for image,label in ds_test:
  if label not in data_test:
    data_test[label] = []
  data_test[label].append(image)
```

[ ]:
```
qtd = [len(data_test[i]) for i in range(100)]
```

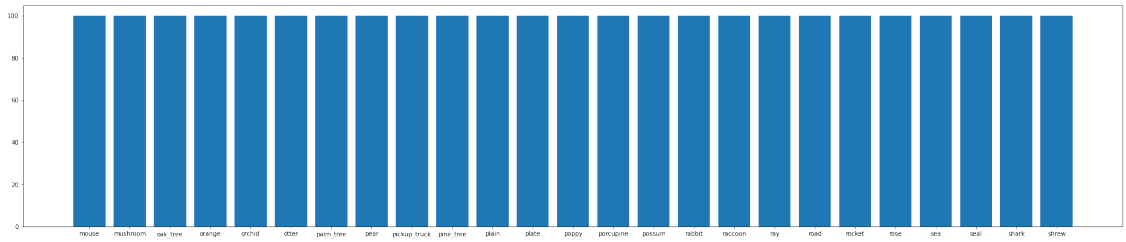
[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[0:25], qtd[0:25])
plt.show()
```



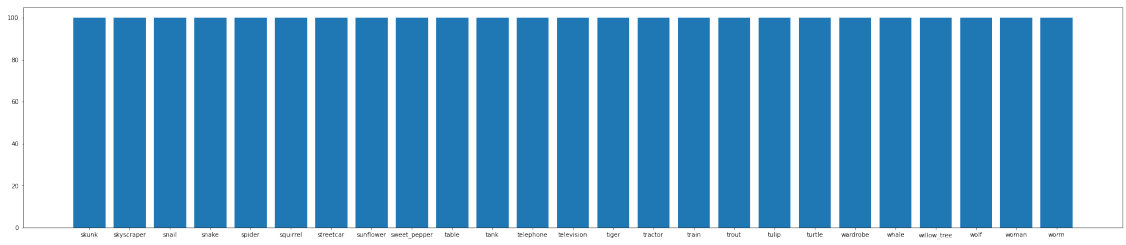
[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[25:50], qtd[25:50])
plt.show()
```



[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[50:75], qtd[50:75])
plt.show()
```



[ ]:
```
fig =  plt.figure(figsize = (25,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(classes[75:100], qtd[75:100])
plt.show()
```



Imagens Médias Função responsável por separar as classes em um dicionário

[ ]:
```
def separate_classes(dataset):
  data = {}
  for image,label in dataset:
    if label not in data:
      data[label] = []
    data[label].append(image)
```

```
  for key in data:
    data[key] = torch.stack(data[key])
  return data
```

Função responsável por calcular a imagem média de cada classe

[ ]:
```
def calculate_mean(dataset):
  data = separate_classes(dataset)
  mean = {}
  for label, images in data.items():
    mean[label] = images.mean(0)
  return mean
```

Sem Normalização

[ ]:
```
%cd data
!rm -rf *
%cd ..
!ls
```

```
/data
/
bin      data     etc   lib32  mnt   python-apt  sbin  tensorflow-1.15.2  usr
boot     datalab  home  lib64  opt   root        srv   tmp                var
content  dev      lib   media  proc  run         sys   tools
```

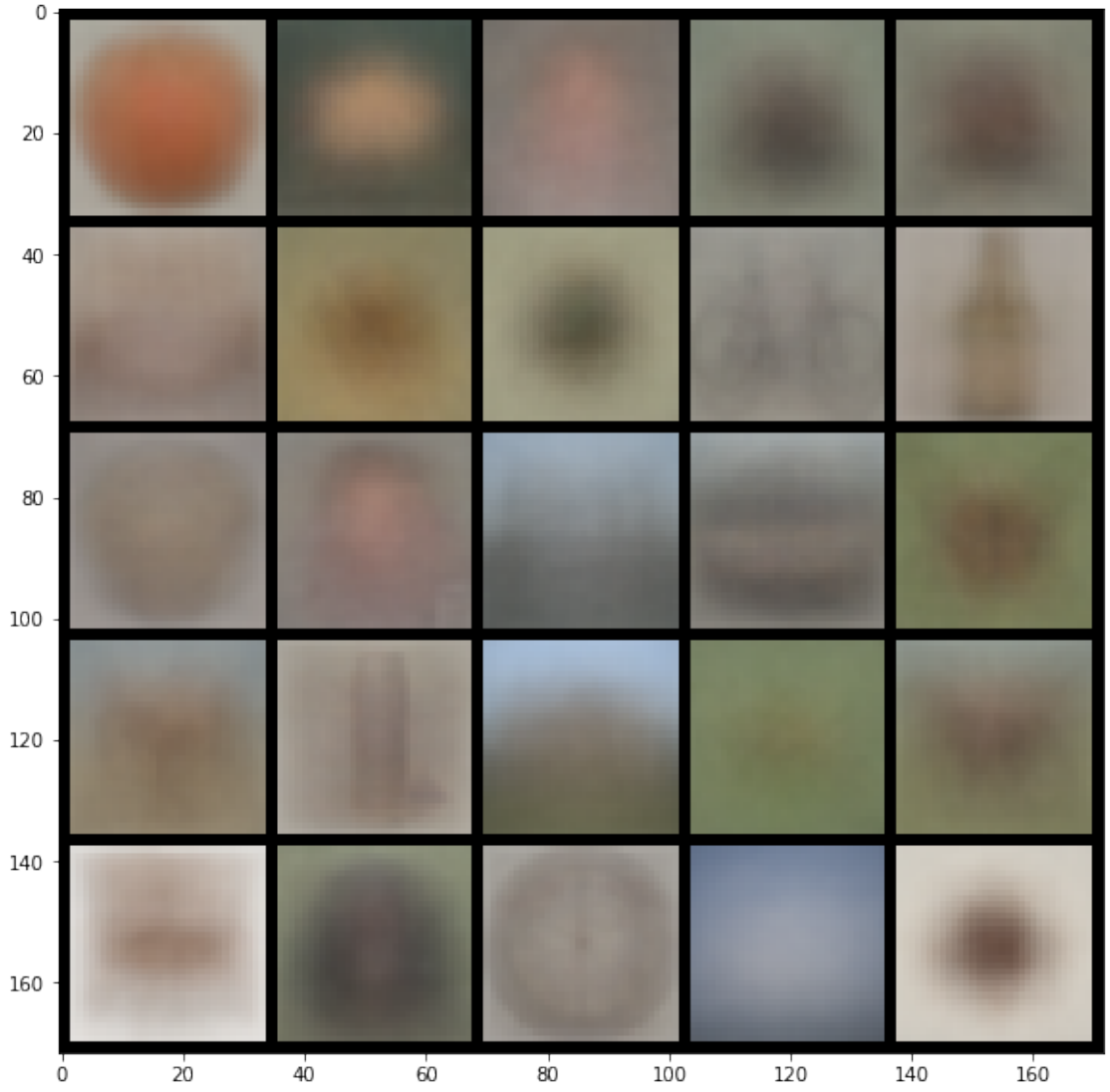
[ ]:
```
ds_train = CIFAR100('data/', train = True, download = True,transform =␣
 ↪transforms.ToTensor())
ds_test =  CIFAR100('data/', train = False, download = True, transform =␣
 ↪transforms.ToTensor())
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to
data/cifar-100-python.tar.gz

  0%|          | 0/169001437 [00:00<?, ?it/s]

Extracting data/cifar-100-python.tar.gz to data/
Files already downloaded and verified
```

[ ]:
```
mean = calculate_mean(ds_train)
```

[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[0:25],nrow = 5).permute(1,2,0))
#title = [classes[int(i)] for i in range(25)]
#ax.set_title(str(title))
print([classes[int(i)] for i in range(25)])
plt.show()
```

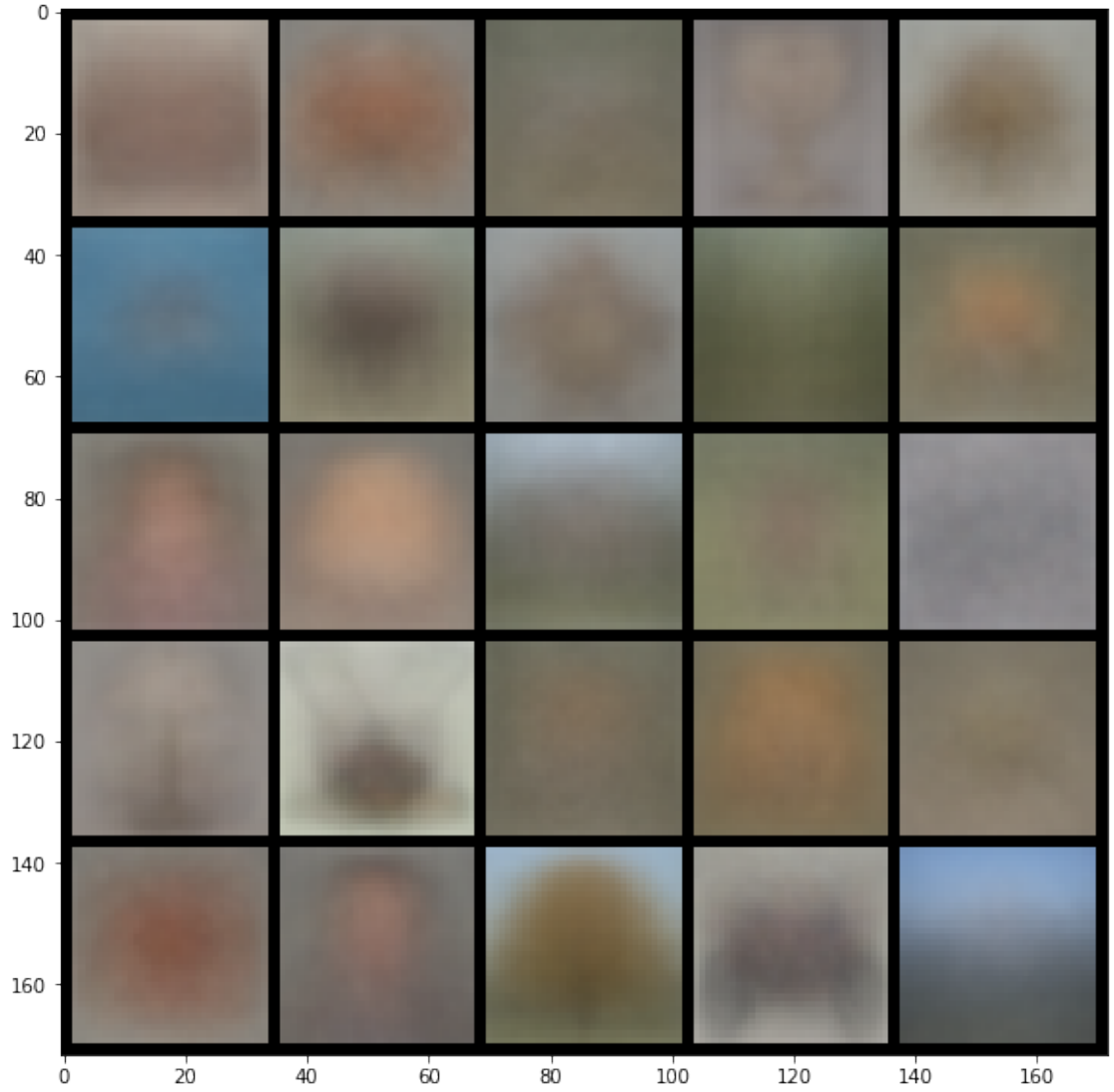
```
['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle',
'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',
'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
'cloud', 'cockroach']
```



[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[25:50],nrow = 5).permute(1,2,0))
#title = [classes[i] for i in range(25,50)]
#ax.set_title(str(title))
print([classes[i] for i in range(25,50)])
plt.show()
```

```
['couch', 'crab', 'crocodile', 'cup', 'dinosaur', 'dolphin', 'elephant',
```

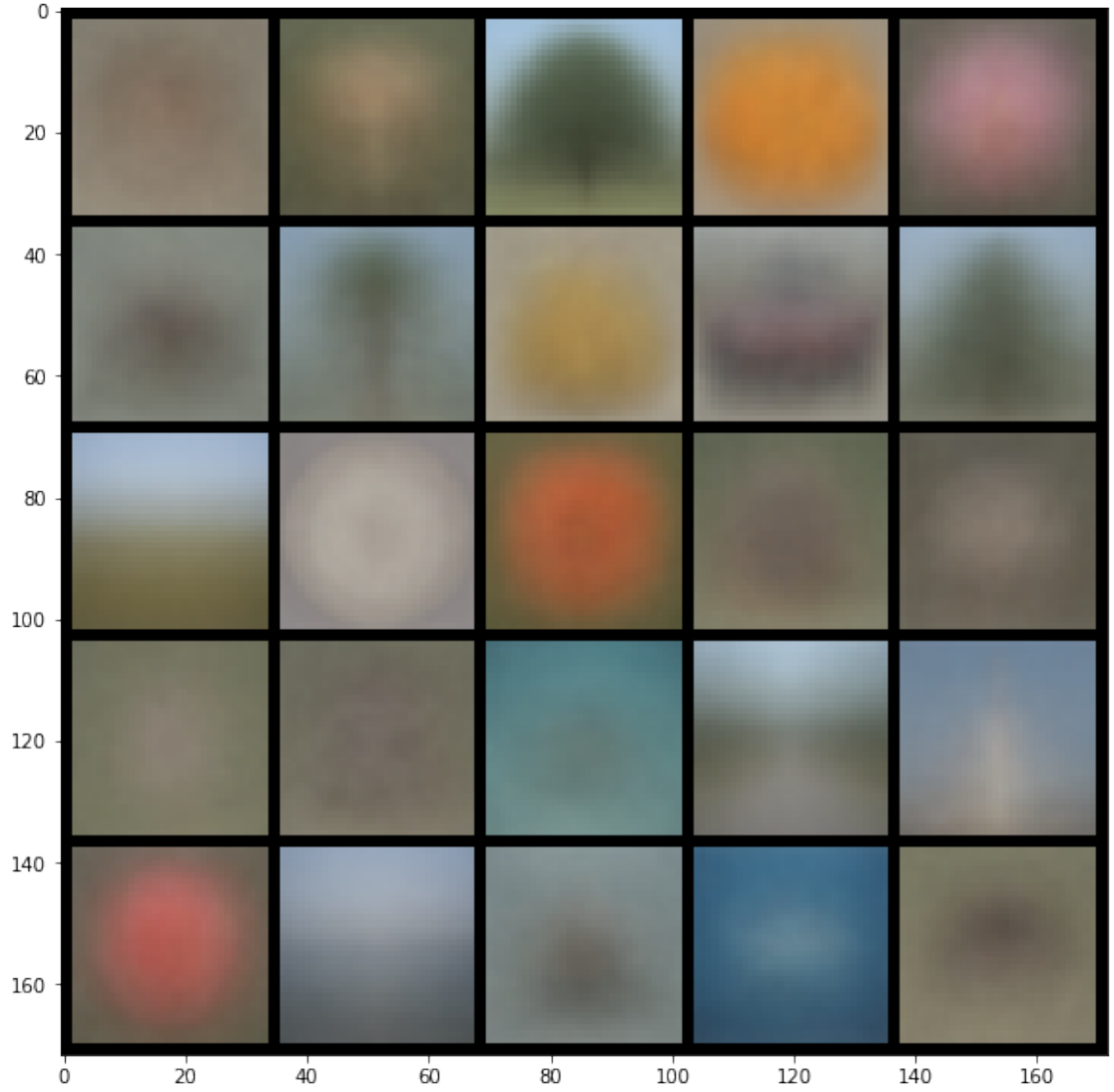
```
'flatfish', 'forest', 'fox', 'girl', 'hamster', 'house', 'kangaroo', 'keyboard',
'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster', 'man',
'maple_tree', 'motorcycle', 'mountain']
```



[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[50:75],nrow = 5).permute(1,2,0))
#title = [classes[i] for i in range(50,75)]
#ax.set_title(str(title))
print([classes[i] for i in range(50,75)])
plt.show()
```

```
['mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree',
'pear', 'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',
```

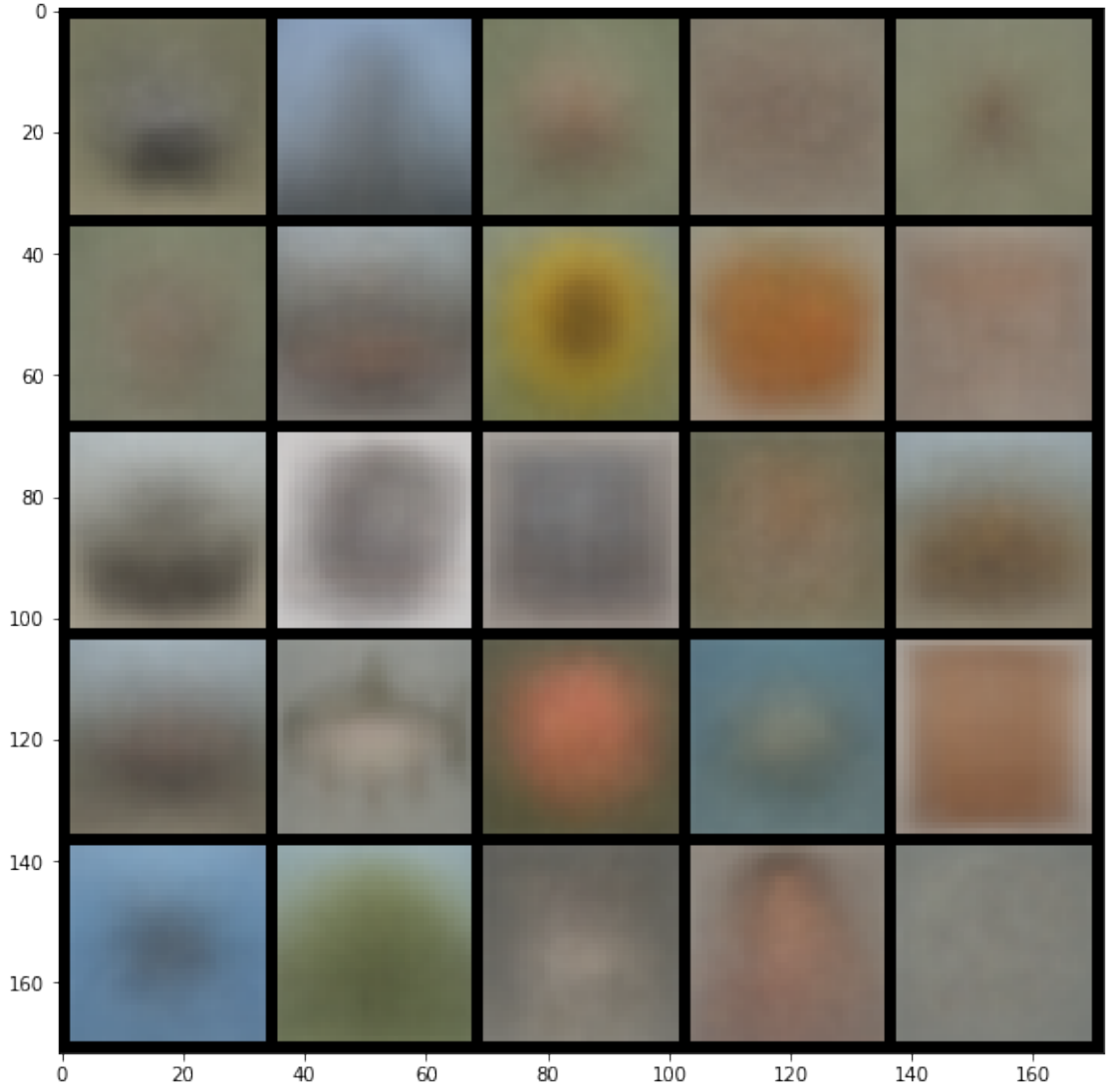
```
'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose', 'sea', 'seal',
'shark', 'shrew']
```



[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[75:100],nrow = 5).permute(1,2,0))
#title = [classes[i] for i in range(75,100)]
#ax.set_title(str(title))
print([classes[i] for i in range(75,100)])
plt.show()
```

```
['skunk', 'skyscraper', 'snail', 'snake', 'spider', 'squirrel', 'streetcar',
'sunflower', 'sweet_pepper', 'table', 'tank', 'telephone', 'television',
'tiger', 'tractor', 'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale',
```

```
'willow_tree', 'wolf', 'woman', 'worm']
```



Imagens Médias Normalizadas

```
[ ]: transform_train = transforms.Compose([#transforms.Resize((227,227)),
                                           #transforms.RandomHorizontalFlip(p=0.7),
                                           transforms.ToTensor(),
                                           transforms.Normalize(mean=[0.5071, 0.4867,␣
     ↪0.4408], std=[0.2675, 0.2565, 0.2761])
                                           ])
     transform_test = transforms.Compose([#transforms.Resize((227,227)),
                                          transforms.ToTensor(),
                                          transforms.Normalize(mean=[0.5071, 0.4867,␣
     ↪0.4408], std=[0.2675, 0.2565, 0.2761])
```

```
])
```

[ ]:
```
%cd data
!rm -rf *
%cd ..
!ls
```

```
/data
/
bin      data     etc   lib32  mnt   python-apt  sbin  tensorflow-1.15.2  usr
boot     datalab  home  lib64  opt   root        srv   tmp                var
content  dev      lib   media  proc  run         sys   tools
```

[ ]:
```
ds_train = CIFAR100('data/', train = True, download = True,transform =␣
↪transform_train)
ds_test =  CIFAR100('data/', train = False, download = True, transform =␣
↪transform_test)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to
data/cifar-100-python.tar.gz
```

```
  0%|          | 0/169001437 [00:00<?, ?it/s]
```

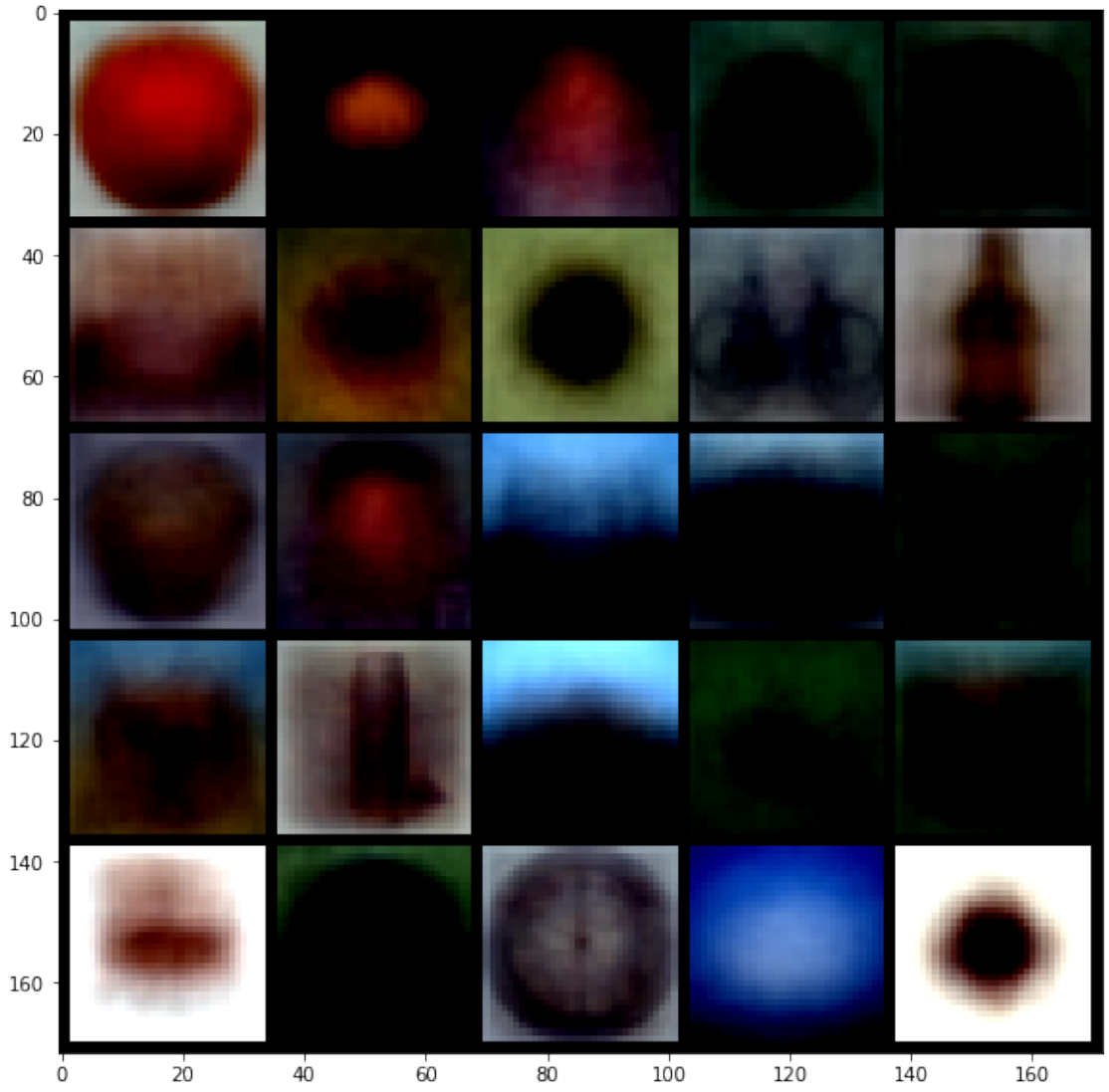
```
Extracting data/cifar-100-python.tar.gz to data/
Files already downloaded and verified
```

[ ]:
```
mean = calculate_mean(ds_train)
```

[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[0:25],nrow = 5).permute(1,2,0))
#title = [classes[int(i)] for i in range(25)]
#ax.set_title(str(title))
print([classes[int(i)] for i in range(25)])
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle',
'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',
'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',
'cloud', 'cockroach']
```

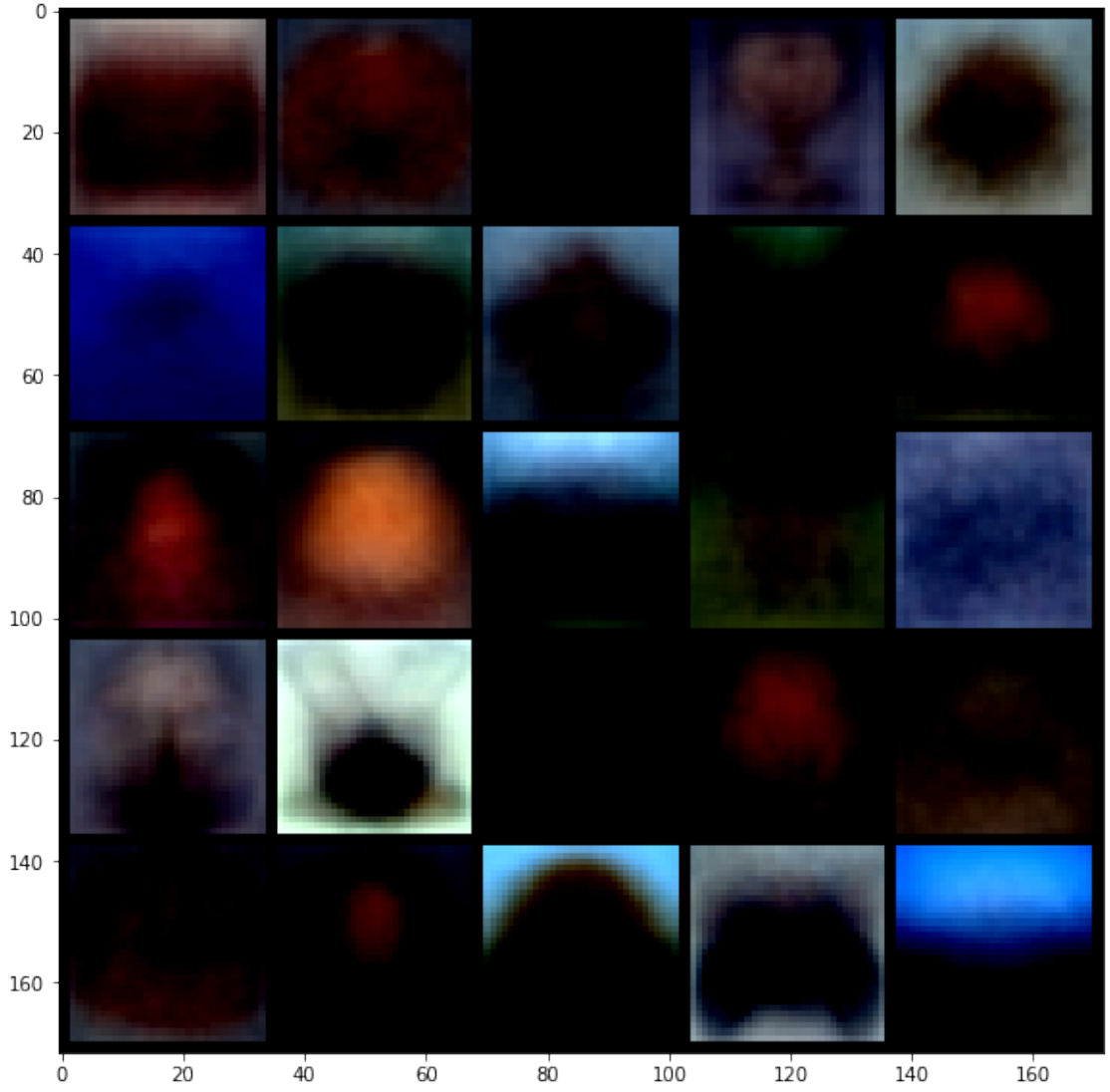


[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[25:50],nrow = 5).permute(1,2,0))
#title = [classes[i] for i in range(25,50)]
#ax.set_title(str(title))
print([classes[i] for i in range(25,50)])
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
['couch', 'crab', 'crocodile', 'cup', 'dinosaur', 'dolphin', 'elephant',
'flatfish', 'forest', 'fox', 'girl', 'hamster', 'house', 'kangaroo', 'keyboard',
'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster', 'man',
```

'maple_tree', 'motorcycle', 'mountain']

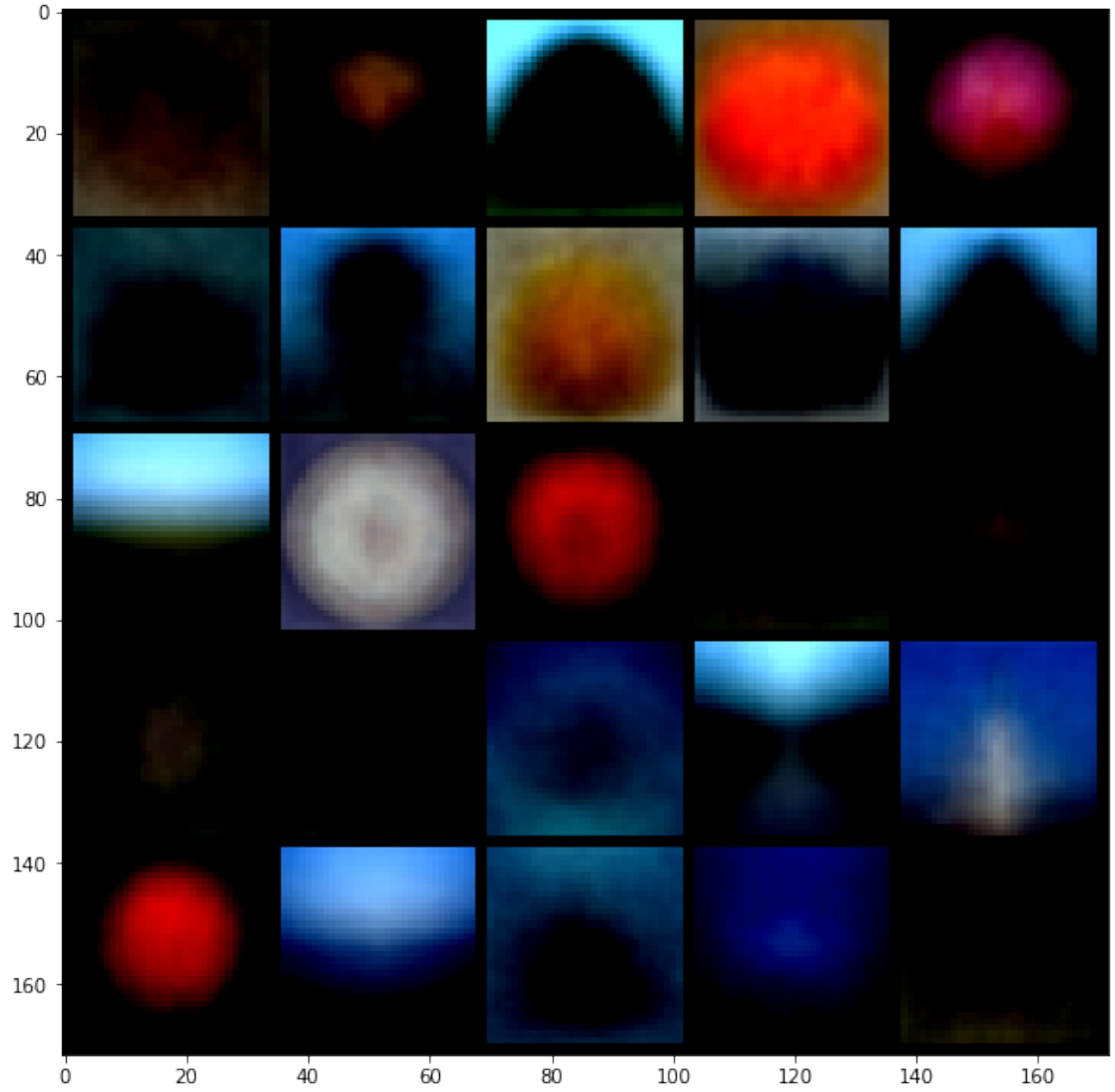


[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[50:75],nrow = 5).permute(1,2,0))
#title = [classes[i] for i in range(50,75)]
#ax.set_title(str(title))
print([classes[i] for i in range(50,75)])
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

['mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree',

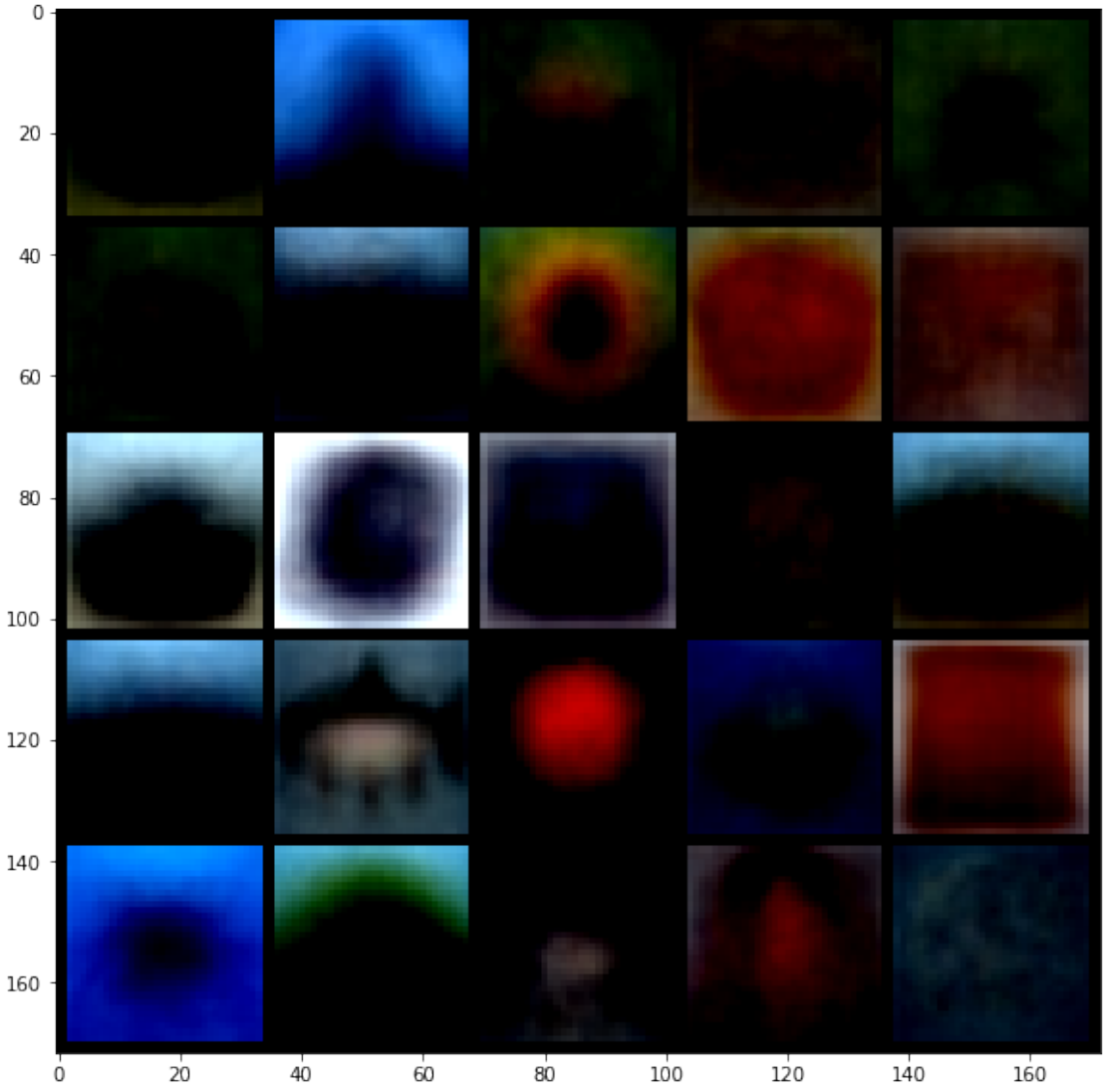
```
'pear', 'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine',
'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose', 'sea', 'seal',
'shark', 'shrew']
```



[ ]:
```
fig,ax = plt.subplots(figsize = (10,10))
mean_images = [mean[i] for i in range(100)]
ax.imshow(make_grid(mean_images[75:100],nrow = 5).permute(1,2,0))
#title = [classes[i] for i in range(75,100)]
#ax.set_title(str(title))
print([classes[i] for i in range(75,100)])
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
['skunk', 'skyscraper', 'snail', 'snake', 'spider', 'squirrel', 'streetcar',
'sunflower', 'sweet_pepper', 'table', 'tank', 'telephone', 'television',
'tiger', 'tractor', 'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale',
'willow_tree', 'wolf', 'woman', 'worm']
```



1.3.2 Experimentos

Pixel Similarity Baseado na abordagem do fastai, iremos criar um modelo básico que não utiliza aprendizado de máquina, para ter uma precisão como base para verificar o desempenho dos próximos modelos. Esse método basicamente cálcula uma imagem média para cada classe do conjunto de treino, esta imagem média é basicamente uma imagem formada pela média de cada pixel das imagens de uma classe.

Para fazer a predição esta arquitetura basicamente cálcula a distância de uma imagem a imagem

média, e escolhe a classe com a menor distância

Podemos usar o erro quadrático médio ou o valor absoluto das diferenças como valor da distância total entre uma imagem e a média da sua classe

[ ]:
```
def distance(image,mean,dist):
  if dist == 'mse':
      return torch.tensor([((image - mean[i])**2).mean().sqrt() for i in␣
↪range(100)])
  elif dist == 'abs':
      return torch.tensor([(image - mean[i]).abs().mean() for i in range(100)])
  return None
```

E escolher a classe que possui a menor distância

[ ]:
```
def simple_predict(image,mean,dist):
  dist = distance(image,mean,dist)
  min_index = 0
  for i in range(1,100):
    if dist[i] < dist[i]:
      min_index = i
  return min_index
```

[ ]:
```
def simple_predict_all(ds_test,mean,dist):
  prediction = []
  for image,label in ds_test:
    prediction.append(simple_predict(image,mean,dist))
  return prediction
```

[ ]:
```
y_true = [label for image, label in ds_test]
y_pred = simple_predict_all(ds_test, mean,'abs')
```

[ ]:
```
print(classification_report(y_true,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.01      1.00      0.02       100
aquarium_fish       0.00      0.00      0.00       100
        baby       0.00      0.00      0.00       100
        bear       0.00      0.00      0.00       100
      beaver       0.00      0.00      0.00       100
         bed       0.00      0.00      0.00       100
         bee       0.00      0.00      0.00       100
      beetle       0.00      0.00      0.00       100
     bicycle       0.00      0.00      0.00       100
      bottle       0.00      0.00      0.00       100
        bowl       0.00      0.00      0.00       100
         boy       0.00      0.00      0.00       100
      bridge       0.00      0.00      0.00       100
```

bus	0.00	0.00	0.00	100
butterfly	0.00	0.00	0.00	100
camel	0.00	0.00	0.00	100
can	0.00	0.00	0.00	100
castle	0.00	0.00	0.00	100
caterpillar	0.00	0.00	0.00	100
cattle	0.00	0.00	0.00	100
chair	0.00	0.00	0.00	100
chimpanzee	0.00	0.00	0.00	100
clock	0.00	0.00	0.00	100
cloud	0.00	0.00	0.00	100
cockroach	0.00	0.00	0.00	100
couch	0.00	0.00	0.00	100
crab	0.00	0.00	0.00	100
crocodile	0.00	0.00	0.00	100
cup	0.00	0.00	0.00	100
dinosaur	0.00	0.00	0.00	100
dolphin	0.00	0.00	0.00	100
elephant	0.00	0.00	0.00	100
flatfish	0.00	0.00	0.00	100
forest	0.00	0.00	0.00	100
fox	0.00	0.00	0.00	100
girl	0.00	0.00	0.00	100
hamster	0.00	0.00	0.00	100
house	0.00	0.00	0.00	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.00	0.00	0.00	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.00	0.00	0.00	100
leopard	0.00	0.00	0.00	100
lion	0.00	0.00	0.00	100
lizard	0.00	0.00	0.00	100
lobster	0.00	0.00	0.00	100
man	0.00	0.00	0.00	100
maple_tree	0.00	0.00	0.00	100
motorcycle	0.00	0.00	0.00	100
mountain	0.00	0.00	0.00	100
mouse	0.00	0.00	0.00	100
mushroom	0.00	0.00	0.00	100
oak_tree	0.00	0.00	0.00	100
orange	0.00	0.00	0.00	100
orchid	0.00	0.00	0.00	100
otter	0.00	0.00	0.00	100
palm_tree	0.00	0.00	0.00	100
pear	0.00	0.00	0.00	100
pickup_truck	0.00	0.00	0.00	100
pine_tree	0.00	0.00	0.00	100
plain	0.00	0.00	0.00	100

```
          plate       0.00      0.00      0.00       100
          poppy       0.00      0.00      0.00       100
       porcupine       0.00      0.00      0.00       100
          possum       0.00      0.00      0.00       100
          rabbit       0.00      0.00      0.00       100
         raccoon       0.00      0.00      0.00       100
             ray       0.00      0.00      0.00       100
            road       0.00      0.00      0.00       100
          rocket       0.00      0.00      0.00       100
            rose       0.00      0.00      0.00       100
             sea       0.00      0.00      0.00       100
            seal       0.00      0.00      0.00       100
           shark       0.00      0.00      0.00       100
           shrew       0.00      0.00      0.00       100
           skunk       0.00      0.00      0.00       100
      skyscraper       0.00      0.00      0.00       100
           snail       0.00      0.00      0.00       100
           snake       0.00      0.00      0.00       100
          spider       0.00      0.00      0.00       100
        squirrel       0.00      0.00      0.00       100
       streetcar       0.00      0.00      0.00       100
       sunflower       0.00      0.00      0.00       100
     sweet_pepper       0.00      0.00      0.00       100
           table       0.00      0.00      0.00       100
            tank       0.00      0.00      0.00       100
       telephone       0.00      0.00      0.00       100
      television       0.00      0.00      0.00       100
           tiger       0.00      0.00      0.00       100
         tractor       0.00      0.00      0.00       100
           train       0.00      0.00      0.00       100
           trout       0.00      0.00      0.00       100
           tulip       0.00      0.00      0.00       100
          turtle       0.00      0.00      0.00       100
        wardrobe       0.00      0.00      0.00       100
           whale       0.00      0.00      0.00       100
      willow_tree       0.00      0.00      0.00       100
            wolf       0.00      0.00      0.00       100
           woman       0.00      0.00      0.00       100
            worm       0.00      0.00      0.00       100

        accuracy                           0.01     10000
       macro avg       0.00      0.01      0.00     10000
    weighted avg       0.00      0.01      0.00     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
```

```
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
y_true = [label for image, label in ds_test]
y_pred = simple_predict_all(ds_test, mean,'mse')
```

[ ]:
```
print(classification_report(y_true,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.01      1.00      0.02       100
aquarium_fish       0.00      0.00      0.00       100
        baby       0.00      0.00      0.00       100
        bear       0.00      0.00      0.00       100
      beaver       0.00      0.00      0.00       100
         bed       0.00      0.00      0.00       100
         bee       0.00      0.00      0.00       100
      beetle       0.00      0.00      0.00       100
     bicycle       0.00      0.00      0.00       100
      bottle       0.00      0.00      0.00       100
        bowl       0.00      0.00      0.00       100
         boy       0.00      0.00      0.00       100
      bridge       0.00      0.00      0.00       100
         bus       0.00      0.00      0.00       100
   butterfly       0.00      0.00      0.00       100
       camel       0.00      0.00      0.00       100
         can       0.00      0.00      0.00       100
      castle       0.00      0.00      0.00       100
  caterpillar       0.00      0.00      0.00       100
      cattle       0.00      0.00      0.00       100
       chair       0.00      0.00      0.00       100
   chimpanzee       0.00      0.00      0.00       100
       clock       0.00      0.00      0.00       100
       cloud       0.00      0.00      0.00       100
   cockroach       0.00      0.00      0.00       100
       couch       0.00      0.00      0.00       100
        crab       0.00      0.00      0.00       100
   crocodile       0.00      0.00      0.00       100
```

cup	0.00	0.00	0.00	100
dinosaur	0.00	0.00	0.00	100
dolphin	0.00	0.00	0.00	100
elephant	0.00	0.00	0.00	100
flatfish	0.00	0.00	0.00	100
forest	0.00	0.00	0.00	100
fox	0.00	0.00	0.00	100
girl	0.00	0.00	0.00	100
hamster	0.00	0.00	0.00	100
house	0.00	0.00	0.00	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.00	0.00	0.00	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.00	0.00	0.00	100
leopard	0.00	0.00	0.00	100
lion	0.00	0.00	0.00	100
lizard	0.00	0.00	0.00	100
lobster	0.00	0.00	0.00	100
man	0.00	0.00	0.00	100
maple_tree	0.00	0.00	0.00	100
motorcycle	0.00	0.00	0.00	100
mountain	0.00	0.00	0.00	100
mouse	0.00	0.00	0.00	100
mushroom	0.00	0.00	0.00	100
oak_tree	0.00	0.00	0.00	100
orange	0.00	0.00	0.00	100
orchid	0.00	0.00	0.00	100
otter	0.00	0.00	0.00	100
palm_tree	0.00	0.00	0.00	100
pear	0.00	0.00	0.00	100
pickup_truck	0.00	0.00	0.00	100
pine_tree	0.00	0.00	0.00	100
plain	0.00	0.00	0.00	100
plate	0.00	0.00	0.00	100
poppy	0.00	0.00	0.00	100
porcupine	0.00	0.00	0.00	100
possum	0.00	0.00	0.00	100
rabbit	0.00	0.00	0.00	100
raccoon	0.00	0.00	0.00	100
ray	0.00	0.00	0.00	100
road	0.00	0.00	0.00	100
rocket	0.00	0.00	0.00	100
rose	0.00	0.00	0.00	100
sea	0.00	0.00	0.00	100
seal	0.00	0.00	0.00	100
shark	0.00	0.00	0.00	100
shrew	0.00	0.00	0.00	100
skunk	0.00	0.00	0.00	100

```
  skyscraper       0.00      0.00      0.00       100
       snail       0.00      0.00      0.00       100
       snake       0.00      0.00      0.00       100
      spider       0.00      0.00      0.00       100
    squirrel       0.00      0.00      0.00       100
   streetcar       0.00      0.00      0.00       100
   sunflower       0.00      0.00      0.00       100
sweet_pepper       0.00      0.00      0.00       100
       table       0.00      0.00      0.00       100
        tank       0.00      0.00      0.00       100
   telephone       0.00      0.00      0.00       100
  television       0.00      0.00      0.00       100
       tiger       0.00      0.00      0.00       100
     tractor       0.00      0.00      0.00       100
       train       0.00      0.00      0.00       100
       trout       0.00      0.00      0.00       100
       tulip       0.00      0.00      0.00       100
      turtle       0.00      0.00      0.00       100
    wardrobe       0.00      0.00      0.00       100
       whale       0.00      0.00      0.00       100
 willow_tree       0.00      0.00      0.00       100
        wolf       0.00      0.00      0.00       100
       woman       0.00      0.00      0.00       100
        worm       0.00      0.00      0.00       100

    accuracy                           0.01     10000
   macro avg       0.00      0.01      0.00     10000
weighted avg       0.00      0.01      0.00     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Ao contrário da abordagem original do fastai para o dataset mnist com apenas duas classe, que teve uma ‘accuracy’ extremamente alta com este método, temos que a ‘accuracy’ que tivemos com o CIFAR 100 que possui muito mais classes, e imagens coloridas, não é melhor do que selecionarmos

uma classe ao acaso

Treinando os Modelos

Dataset

```
[ ]: transform_train = transforms.Compose([#transforms.Resize((227,227)),
                                           transforms.RandomHorizontalFlip(p=0.7),
                                           transforms.ToTensor(),
                                           transforms.Normalize(mean=[0.5071, 0.4867,␣
     ↪0.4408], std=[0.2675, 0.2565, 0.2761])
                                           ])
     transform_test = transforms.Compose([#transforms.Resize((227,227)),
                                          transforms.ToTensor(),
                                          transforms.Normalize(mean=[0.5071, 0.4867,␣
     ↪0.4408], std=[0.2675, 0.2565, 0.2761])
                                          ])
```

```
[ ]: %cd data
     !rm -rf *
     %cd ..
     !ls
```

```
/data
/
alexnet0.pkl     cnn_1.pkl  lenet0.pkl     mnt          sys
alexnet_1ep.pkl  cnn.pkl    lenet_1ep.pkl  opt          tensorflow-1.15.2
alexnet_1.pkl    content    lenet_1.pkl    proc         tmp
alexnet.pkl      data       lenet.pkl      python-apt   tools
bin              datalab    lib            root         usr
boot             dev        lib32          run          var
cnn0.pkl         etc        lib64          sbin
cnn_1ep.pkl      home       media          srv
```

```
[ ]: ds_train = CIFAR100('data/', train = True, download = True,transform =␣
     ↪transform_train)
     ds_test =  CIFAR100('data/', train = False, download = True, transform =␣
     ↪transform_test)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to
data/cifar-100-python.tar.gz

  0%|          | 0/169001437 [00:00<?, ?it/s]

Extracting data/cifar-100-python.tar.gz to data/
Files already downloaded and verified
```

```
[ ]: classes = ds_train.classes
```

[ ]:
```
ds_train.data.shape
```

[ ]:
```
(50000, 32, 32, 3)
```

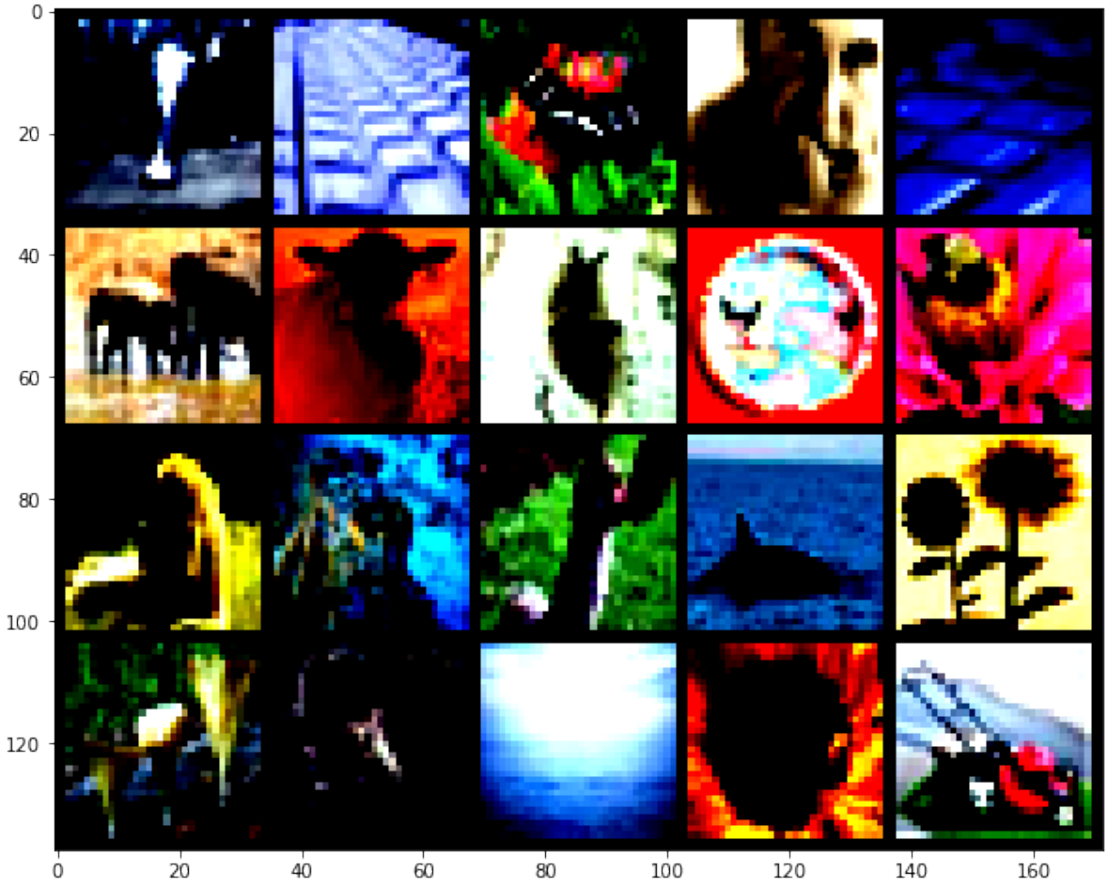
Dataloaders

[ ]:
```
batch_size = 40
dl_train = DataLoader(ds_train, batch_size = batch_size, shuffle = True)
dl_test =  DataLoader(ds_test, batch_size = batch_size, shuffle = True)
```

[ ]:
```
batch = get_batch(dl_train)
plot_data(batch,classes)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
<generator object plot_data.<locals>.<genexpr> at 0x7f40dcf9bf50>
```



Modelos O Modelo Proposto por Nós

[24]:
```
class CNN100(nn.Module):
  def __init__(self):
    super(CNN100,self).__init__()
    self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 24, kernel_size = 3)
    self.conv1_bn = nn.BatchNorm2d(24)
    self.conv2 = nn.Conv2d(in_channels = 24, out_channels = 72, kernel_size = 3)
    self.conv2_bn = nn.BatchNorm2d(72)
    self.conv3 = nn.Conv2d(in_channels = 72, out_channels = 72, kernel_size = 3)
    self.conv3_bn = nn.BatchNorm2d(72)
    self.conv4 = nn.Conv2d(in_channels = 72, out_channels = 216, kernel_size =␣
 ↪3)
    self.conv4_bn = nn.BatchNorm2d(216)
    self.pool = nn.MaxPool2d(kernel_size = 2)
    self.dropout = nn.Dropout(p = 0.2)
    self.fc1 = nn.Linear(216*5*5, 600)
    self.fc1_bn = nn.BatchNorm1d(600)
    self.fc2 = nn.Linear(600,200)
    self.fc2_bn = nn.BatchNorm1d(200)
    self.fc3 = nn.Linear(200,100)

  def forward(self, x):
    x = self.conv1(x)
    x = F.relu(self.conv1_bn(x))
    x = self.conv2(x)
    x = F.relu(self.conv2_bn(x))
    x = self.pool(x)

    x = self.conv3(x)
    x = F.relu(self.conv3_bn(x))
    x = self.conv4(x)
    x = F.relu(self.conv4_bn(x))
    x = self.pool(x)

    x = x.view(-1,216*5*5)

    x = F.relu(self.fc1(x))
    #x = F.relu(self.fc1_bn(x))
    x = self.dropout(x)
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    #x = self.fc2_bn(x)

    x = self.fc3(x)
    return F.log_softmax(x,dim=1)
```

[25]:
```
cnn = CNN100()
cnn = cnn.to(get_device())
torch.save(cnn,'cnn0.pkl')
print(cnn)
```

```
CNN100(
  (conv1): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1))
  (conv1_bn): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv2): Conv2d(24, 72, kernel_size=(3, 3), stride=(1, 1))
  (conv2_bn): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv3): Conv2d(72, 72, kernel_size=(3, 3), stride=(1, 1))
  (conv3_bn): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv4): Conv2d(72, 216, kernel_size=(3, 3), stride=(1, 1))
  (conv4_bn): BatchNorm2d(216, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=5400, out_features=600, bias=True)
  (fc1_bn): BatchNorm1d(600, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc2): Linear(in_features=600, out_features=200, bias=True)
  (fc2_bn): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc3): Linear(in_features=200, out_features=100, bias=True)
)
```

LeNet (Modificado)

[26]:
```
class LeNet(nn.Module):
  def __init__(self):
    super(LeNet,self).__init__()
    self.conv1 = nn.Conv2d(3, 6, kernel_size=5, padding=2)
    self.avgPool = nn.AvgPool2d(kernel_size=2, stride=2)
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
    self.flatten = nn.Flatten()
    self.fc1 = nn.Linear(576, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 100)

  def forward(self, x):
    x = F.sigmoid(self.conv1(x))
    x = self.avgPool(x)
    x = F.sigmoid(self.conv2(x))
    x = self.avgPool(x)
```

```
    x = self.flatten(x)
    x = F.sigmoid(self.fc1(x))
    x = F.sigmoid(self.fc2(x))
    x = self.fc3(x)
    #print(x.shape)
    return x
```

[27]:
```
lenet = LeNet()
lenet = lenet.to(get_device())
torch.save(lenet,'lenet0.pkl')
print(lenet)
```

```
LeNet(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (avgPool): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=576, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=100, bias=True)
)
```

AlexNet (Modificado)

[28]:
```
class AlexNet(nn.Module):
  def __init__(self):
    super(AlexNet,self).__init__()
    self.conv1 = nn.Conv2d(3, 96, kernel_size=11, padding=1)
    self.conv2 = nn.Conv2d(96, 256, kernel_size=5, padding=2)
    self.conv3 = nn.Conv2d(256, 384, kernel_size=3, padding=1)
    self.conv4 = nn.Conv2d(384, 384, kernel_size=3, padding=1)
    self.conv5 = nn.Conv2d(384, 256, kernel_size=3, padding=1)
    self.pool = nn.MaxPool2d(kernel_size=3, stride=2)
    self.dropout = nn.Dropout(p=0.5)
    self.fc1 = nn.Linear(256*2*2, 4096)
    self.fc2 = nn.Linear(4096, 4096)
    self.fc3 = nn.Linear(4096, 100)

  def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool(x)

    x = F.relu(self.conv2(x))
    x = self.pool(x)

    x = F.relu(self.conv3(x))
    x = F.relu(self.conv4(x))
    x = F.relu(self.conv5(x))
```

```
    x = self.pool(x)
    #print(x.shape)
    x = x.view(-1,256*2*2)

    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    x = self.fc3(x)

    return x
```

[29]:
```
alexnet = AlexNet()
alexnet = alexnet.to(get_device())
torch.save(alexnet,'alexnet0.pkl')
print(alexnet)
```

```
AlexNet(
  (conv1): Conv2d(3, 96, kernel_size=(11, 11), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv3): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=1024, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=4096, bias=True)
  (fc3): Linear(in_features=4096, out_features=100, bias=True)
)
```

Treino

[ ]:
```
num_epochs = 50
```

Nosso Modelo

[ ]:
```
fit(cnn,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/50]: Loss : 4.221396705627441 Step: 1000
Epoch:  [2/50]: Loss : 3.442529371023178 Step: 1000
Epoch:  [3/50]: Loss : 3.016068526506424 Step: 1000
Epoch:  [4/50]: Loss : 2.726536409974098 Step: 1000
Epoch:  [5/50]: Loss : 2.5214961433410643 Step: 1000
Epoch:  [6/50]: Loss : 2.3667297565937044 Step: 1000
Epoch:  [7/50]: Loss : 2.2413606016635894 Step: 1000
Epoch:  [8/50]: Loss : 2.131696190714836 Step: 1000
Epoch:  [9/50]: Loss : 2.045403267621994 Step: 1000
Epoch:  [10/50]: Loss : 1.9596656314134597 Step: 1000
```

```
Epoch:  [11/50]: Loss : 1.8683195123672485 Step: 1000
Epoch:  [12/50]: Loss : 1.8028394047021865 Step: 1000
Epoch:  [13/50]: Loss : 1.7309640917778015 Step: 1000
Epoch:  [14/50]: Loss : 1.672695976316929 Step: 1000
Epoch:  [15/50]: Loss : 1.6129718722701072 Step: 1000
Epoch:  [16/50]: Loss : 1.549377133488655 Step: 1000
Epoch:  [17/50]: Loss : 1.5024642004966735 Step: 1000
Epoch:  [18/50]: Loss : 1.443586164832115 Step: 1000
Epoch:  [19/50]: Loss : 1.3880341518521309 Step: 1000
Epoch:  [20/50]: Loss : 1.3480051854252815 Step: 1000
Epoch:  [21/50]: Loss : 1.3038967319130899 Step: 1000
Epoch:  [22/50]: Loss : 1.252393138229847 Step: 1000
Epoch:  [23/50]: Loss : 1.209368919789791 Step: 1000
Epoch:  [24/50]: Loss : 1.1628046661615372 Step: 1000
Epoch:  [25/50]: Loss : 1.11957909989357 Step: 1000
Epoch:  [26/50]: Loss : 1.080538897484541 Step: 1000
Epoch:  [27/50]: Loss : 1.0366282995641232 Step: 1000
Epoch:  [28/50]: Loss : 1.0058936926424504 Step: 1000
Epoch:  [29/50]: Loss : 0.9563290413022041 Step: 1000
Epoch:  [30/50]: Loss : 0.9259406097531319 Step: 1000
Epoch:  [31/50]: Loss : 0.8876181870102883 Step: 1000
Epoch:  [32/50]: Loss : 0.8457021358609199 Step: 1000
Epoch:  [33/50]: Loss : 0.81483776268363 Step: 1000
Epoch:  [34/50]: Loss : 0.7788742383718491 Step: 1000
Epoch:  [35/50]: Loss : 0.7499317646026611 Step: 1000
Epoch:  [36/50]: Loss : 0.7247606095671654 Step: 1000
Epoch:  [37/50]: Loss : 0.6977264957129955 Step: 1000
Epoch:  [38/50]: Loss : 0.6708309221863746 Step: 1000
Epoch:  [39/50]: Loss : 0.6321686197370291 Step: 1000
Epoch:  [40/50]: Loss : 0.6157633363008499 Step: 1000
Epoch:  [41/50]: Loss : 0.5888425379991531 Step: 1000
Epoch:  [42/50]: Loss : 0.5690892262756825 Step: 1000
Epoch:  [43/50]: Loss : 0.5412353737056256 Step: 1000
Epoch:  [44/50]: Loss : 0.5249063689261675 Step: 1000
Epoch:  [45/50]: Loss : 0.5023443355858326 Step: 1000
Epoch:  [46/50]: Loss : 0.4778301683217287 Step: 1000
Epoch:  [47/50]: Loss : 0.4582817324399948 Step: 1000
Epoch:  [48/50]: Loss : 0.44582127273082733 Step: 1000
Epoch:  [49/50]: Loss : 0.4311845659315586 Step: 1000
Epoch:  [50/50]: Loss : 0.4129851536899805 Step: 1000
The Training is Finished
```

[ ]:
```
torch.save(cnn,'cnn.pkl')
```

LeNet

[ ]:
```
fit(lenet,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")

Epoch:  [1/50]: Loss : 4.610275352478028 Step: 1000
Epoch:  [2/50]: Loss : 4.60709143781662 Step: 1000
Epoch:  [3/50]: Loss : 4.607117957592011 Step: 1000
Epoch:  [4/50]: Loss : 4.607108925819397 Step: 1000
Epoch:  [5/50]: Loss : 4.607210103034973 Step: 1000
Epoch:  [6/50]: Loss : 4.607090084075928 Step: 1000
Epoch:  [7/50]: Loss : 4.607166917800903 Step: 1000
Epoch:  [8/50]: Loss : 4.606955875873566 Step: 1000
Epoch:  [9/50]: Loss : 4.607037053585053 Step: 1000
Epoch:  [10/50]: Loss : 4.606995623588562 Step: 1000
Epoch:  [11/50]: Loss : 4.606891127109527 Step: 1000
Epoch:  [12/50]: Loss : 4.607133620738983 Step: 1000
Epoch:  [13/50]: Loss : 4.606977580070495 Step: 1000
Epoch:  [14/50]: Loss : 4.606872906684876 Step: 1000
Epoch:  [15/50]: Loss : 4.606975681304932 Step: 1000
Epoch:  [16/50]: Loss : 4.606828499794006 Step: 1000
Epoch:  [17/50]: Loss : 4.606954144954681 Step: 1000
Epoch:  [18/50]: Loss : 4.60695934677124 Step: 1000
Epoch:  [19/50]: Loss : 4.6069412140846255 Step: 1000
Epoch:  [20/50]: Loss : 4.606876938343048 Step: 1000
Epoch:  [21/50]: Loss : 4.606791066169738 Step: 1000
Epoch:  [22/50]: Loss : 4.606962107181549 Step: 1000
Epoch:  [23/50]: Loss : 4.606927053451538 Step: 1000
Epoch:  [24/50]: Loss : 4.606828117847443 Step: 1000
Epoch:  [25/50]: Loss : 4.606896060943604 Step: 1000
Epoch:  [26/50]: Loss : 4.606832995414734 Step: 1000
Epoch:  [27/50]: Loss : 4.606859262466431 Step: 1000
Epoch:  [28/50]: Loss : 4.606718874454498 Step: 1000
Epoch:  [29/50]: Loss : 4.60677660036087 Step: 1000
Epoch:  [30/50]: Loss : 4.606802321910858 Step: 1000
Epoch:  [31/50]: Loss : 4.606823091030121 Step: 1000
Epoch:  [32/50]: Loss : 4.606867810249328 Step: 1000
Epoch:  [33/50]: Loss : 4.6067352352142334 Step: 1000
Epoch:  [34/50]: Loss : 4.6067351365089415 Step: 1000
Epoch:  [35/50]: Loss : 4.6066831459999085 Step: 1000
Epoch:  [36/50]: Loss : 4.606805233478546 Step: 1000
Epoch:  [37/50]: Loss : 4.606628386974335 Step: 1000
Epoch:  [38/50]: Loss : 4.606787374019623 Step: 1000
Epoch:  [39/50]: Loss : 4.606802604198456 Step: 1000
Epoch:  [40/50]: Loss : 4.606585065841675 Step: 1000
Epoch:  [41/50]: Loss : 4.606699099540711 Step: 1000
Epoch:  [42/50]: Loss : 4.606608563899994 Step: 1000
Epoch:  [43/50]: Loss : 4.606523509025574 Step: 1000
```

```
Epoch:  [44/50]: Loss : 4.6065899238586425 Step: 1000
Epoch:  [45/50]: Loss : 4.606579857349396 Step: 1000
Epoch:  [46/50]: Loss : 4.6065189414024355 Step: 1000
Epoch:  [47/50]: Loss : 4.606556525230408 Step: 1000
Epoch:  [48/50]: Loss : 4.606450908184051 Step: 1000
Epoch:  [49/50]: Loss : 4.606547230243683 Step: 1000
Epoch:  [50/50]: Loss : 4.606608632564544 Step: 1000
The Training is Finished
```

```
[ ]: torch.save(lenet,'lenet.pkl')
```

AlexNet

```
[ ]: fit(alexnet,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/50]: Loss : 4.605053863048553 Step: 1000
Epoch:  [2/50]: Loss : 4.603819807529449 Step: 1000
Epoch:  [3/50]: Loss : 4.554995917797089 Step: 1000
Epoch:  [4/50]: Loss : 4.223932805538177 Step: 1000
Epoch:  [5/50]: Loss : 4.054544434785843 Step: 1000
Epoch:  [6/50]: Loss : 3.8857276592254637 Step: 1000
Epoch:  [7/50]: Loss : 3.7299574971199037 Step: 1000
Epoch:  [8/50]: Loss : 3.5947757971286776 Step: 1000
Epoch:  [9/50]: Loss : 3.465284445285797 Step: 1000
Epoch:  [10/50]: Loss : 3.3228505601882934 Step: 1000
Epoch:  [11/50]: Loss : 3.193504056215286 Step: 1000
Epoch:  [12/50]: Loss : 3.049126019239426 Step: 1000
Epoch:  [13/50]: Loss : 2.941783545255661 Step: 1000
Epoch:  [14/50]: Loss : 2.8360857899188994 Step: 1000
Epoch:  [15/50]: Loss : 2.7406764866113664 Step: 1000
Epoch:  [16/50]: Loss : 2.645433095216751 Step: 1000
Epoch:  [17/50]: Loss : 2.5626494125127794 Step: 1000
Epoch:  [18/50]: Loss : 2.481579114317894 Step: 1000
Epoch:  [19/50]: Loss : 2.406880156636238 Step: 1000
Epoch:  [20/50]: Loss : 2.333530393242836 Step: 1000
Epoch:  [21/50]: Loss : 2.2658006126880648 Step: 1000
Epoch:  [22/50]: Loss : 2.1945547288656235 Step: 1000
Epoch:  [23/50]: Loss : 2.1288208965063093 Step: 1000
Epoch:  [24/50]: Loss : 2.063092941761017 Step: 1000
Epoch:  [25/50]: Loss : 2.0044244683980943 Step: 1000
Epoch:  [26/50]: Loss : 1.9417854435443878 Step: 1000
Epoch:  [27/50]: Loss : 1.878165806889534 Step: 1000
Epoch:  [28/50]: Loss : 1.8241327632665634 Step: 1000
Epoch:  [29/50]: Loss : 1.7684704574346541 Step: 1000
Epoch:  [30/50]: Loss : 1.7114755480289459 Step: 1000
Epoch:  [31/50]: Loss : 1.653062198996544 Step: 1000
Epoch:  [32/50]: Loss : 1.5938974758982658 Step: 1000
Epoch:  [33/50]: Loss : 1.54671443516016 Step: 1000
```

```
Epoch:  [34/50]: Loss : 1.4898958851099013 Step: 1000
Epoch:  [35/50]: Loss : 1.443453662276268 Step: 1000
Epoch:  [36/50]: Loss : 1.397236880660057 Step: 1000
Epoch:  [37/50]: Loss : 1.3422015964388847 Step: 1000
Epoch:  [38/50]: Loss : 1.2835725001096725 Step: 1000
Epoch:  [39/50]: Loss : 1.2301955754756928 Step: 1000
Epoch:  [40/50]: Loss : 1.1837763168215751 Step: 1000
Epoch:  [41/50]: Loss : 1.127120435297489 Step: 1000
Epoch:  [42/50]: Loss : 1.0888645758330822 Step: 1000
Epoch:  [43/50]: Loss : 1.0291140142679214 Step: 1000
Epoch:  [44/50]: Loss : 0.9894203084409237 Step: 1000
Epoch:  [45/50]: Loss : 0.9463449937701225 Step: 1000
Epoch:  [46/50]: Loss : 0.8882468537092209 Step: 1000
Epoch:  [47/50]: Loss : 0.8516607826948166 Step: 1000
Epoch:  [48/50]: Loss : 0.8000303206443786 Step: 1000
Epoch:  [49/50]: Loss : 0.7640866255164146 Step: 1000
Epoch:  [50/50]: Loss : 0.7250659963488579 Step: 1000
The Training is Finished
```

[ ]:
```
torch.save(alexnet,'alexnet.pkl')
```

Salvando os Modelos

[ ]:
```
torch.save(cnn,'cnn.pkl')
torch.save(lenet,'lenet.pkl')
torch.save(alexnet,'alexnet.pkl')

#export_to_onnx(model,num_epochs = 1,steps = 500)
```

Test Carregando os Modelos

[ ]:
```
cnn = torch.load('cnn.pkl')
lenet = torch.load('lenet.pkl')
alexnet = torch.load('alexnet.pkl')
```

Nosso Modelo

[ ]:
```
test(cnn,dl_test)
```

```
Accuracy =  5520 / 10000 55.2 %
```

LeNet

[ ]:
```
test(lenet,dl_test)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

```
Accuracy =  100 / 10000 1.0 %
```

AlexNet

[ ]:
```
test(alexnet,dl_test)
```

```
Accuracy =  4906 / 10000 49.059999999999995 %
```

Métricas de Desempenho Nosso Modelo

[ ]:
```
y_pred = predict(cnn,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.78      0.77      0.77       100
aquarium_fish       0.67      0.71      0.69       100
        baby       0.42      0.42      0.42       100
        bear       0.46      0.22      0.30       100
      beaver       0.32      0.40      0.36       100
         bed       0.36      0.55      0.43       100
         bee       0.62      0.66      0.64       100
      beetle       0.54      0.66      0.59       100
     bicycle       0.79      0.76      0.78       100
      bottle       0.54      0.74      0.62       100
        bowl       0.48      0.32      0.38       100
         boy       0.50      0.34      0.40       100
      bridge       0.56      0.60      0.58       100
         bus       0.51      0.53      0.52       100
   butterfly       0.57      0.54      0.55       100
       camel       0.34      0.59      0.43       100
         can       0.64      0.55      0.59       100
      castle       0.73      0.69      0.71       100
  caterpillar       0.40      0.49      0.44       100
      cattle       0.55      0.41      0.47       100
       chair       0.79      0.74      0.76       100
   chimpanzee       0.66      0.72      0.69       100
       clock       0.66      0.44      0.53       100
       cloud       0.76      0.71      0.74       100
   cockroach       0.76      0.67      0.71       100
       couch       0.46      0.36      0.40       100
        crab       0.59      0.49      0.54       100
   crocodile       0.35      0.49      0.41       100
         cup       0.78      0.70      0.74       100
    dinosaur       0.50      0.50      0.50       100
     dolphin       0.60      0.63      0.61       100
    elephant       0.51      0.53      0.52       100
     flatfish       0.57      0.58      0.57       100
```

forest	0.57	0.56	0.56	100
fox	0.61	0.56	0.58	100
girl	0.31	0.38	0.34	100
hamster	0.60	0.55	0.58	100
house	0.60	0.56	0.58	100
kangaroo	0.39	0.43	0.41	100
keyboard	0.80	0.56	0.66	100
lamp	0.44	0.47	0.46	100
lawn_mower	0.83	0.78	0.80	100
leopard	0.60	0.60	0.60	100
lion	0.73	0.51	0.60	100
lizard	0.27	0.38	0.31	100
lobster	0.48	0.48	0.48	100
man	0.39	0.30	0.34	100
maple_tree	0.59	0.54	0.56	100
motorcycle	0.79	0.84	0.82	100
mountain	0.74	0.66	0.70	100
mouse	0.34	0.32	0.33	100
mushroom	0.65	0.51	0.57	100
oak_tree	0.55	0.64	0.59	100
orange	0.72	0.79	0.76	100
orchid	0.59	0.69	0.64	100
otter	0.25	0.25	0.25	100
palm_tree	0.76	0.81	0.79	100
pear	0.57	0.63	0.60	100
pickup_truck	0.77	0.64	0.70	100
pine_tree	0.52	0.44	0.48	100
plain	0.80	0.73	0.76	100
plate	0.68	0.58	0.63	100
poppy	0.66	0.56	0.61	100
porcupine	0.55	0.52	0.54	100
possum	0.41	0.35	0.38	100
rabbit	0.37	0.29	0.32	100
raccoon	0.62	0.40	0.49	100
ray	0.33	0.37	0.35	100
road	0.71	0.85	0.77	100
rocket	0.72	0.66	0.69	100
rose	0.62	0.54	0.58	100
sea	0.68	0.76	0.72	100
seal	0.31	0.21	0.25	100
shark	0.36	0.49	0.41	100
shrew	0.38	0.40	0.39	100
skunk	0.97	0.59	0.73	100
skyscraper	0.66	0.82	0.73	100
snail	0.38	0.49	0.43	100
snake	0.38	0.39	0.39	100
spider	0.70	0.57	0.63	100
squirrel	0.26	0.42	0.32	100

```
    streetcar       0.44      0.66      0.53       100
    sunflower       0.87      0.78      0.82       100
sweet_pepper       0.45      0.56      0.50       100
        table       0.50      0.48      0.49       100
         tank       0.77      0.61      0.68       100
    telephone       0.54      0.57      0.55       100
   television       0.59      0.61      0.60       100
        tiger       0.62      0.74      0.67       100
      tractor       0.59      0.67      0.63       100
        train       0.54      0.55      0.54       100
        trout       0.63      0.67      0.65       100
        tulip       0.61      0.42      0.50       100
       turtle       0.30      0.35      0.32       100
    wardrobe       0.79      0.81      0.80       100
        whale       0.68      0.54      0.60       100
  willow_tree       0.64      0.35      0.45       100
         wolf       0.56      0.58      0.57       100
        woman       0.27      0.30      0.28       100
         worm       0.54      0.57      0.56       100

     accuracy                           0.55     10000
    macro avg       0.57      0.55      0.55     10000
 weighted avg       0.57      0.55      0.55     10000
```

LeNet

[ ]:
```
y_pred = predict(lenet,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

```
              precision    recall  f1-score   support

       apple       0.00      0.00      0.00       100
aquarium_fish       0.00      0.00      0.00       100
        baby       0.00      0.00      0.00       100
        bear       0.00      0.00      0.00       100
      beaver       0.00      0.00      0.00       100
         bed       0.00      0.00      0.00       100
         bee       0.00      0.00      0.00       100
      beetle       0.00      0.00      0.00       100
     bicycle       0.00      0.00      0.00       100
      bottle       0.00      0.00      0.00       100
        bowl       0.00      0.00      0.00       100
```

boy	0.00	0.00	0.00	100
bridge	0.00	0.00	0.00	100
bus	0.00	0.00	0.00	100
butterfly	0.00	0.00	0.00	100
camel	0.00	0.00	0.00	100
can	0.00	0.00	0.00	100
castle	0.00	0.00	0.00	100
caterpillar	0.00	0.00	0.00	100
cattle	0.00	0.00	0.00	100
chair	0.00	0.00	0.00	100
chimpanzee	0.00	0.00	0.00	100
clock	0.00	0.00	0.00	100
cloud	0.00	0.00	0.00	100
cockroach	0.00	0.00	0.00	100
couch	0.00	0.00	0.00	100
crab	0.00	0.00	0.00	100
crocodile	0.00	0.00	0.00	100
cup	0.00	0.00	0.00	100
dinosaur	0.00	0.00	0.00	100
dolphin	0.00	0.00	0.00	100
elephant	0.00	0.00	0.00	100
flatfish	0.00	0.00	0.00	100
forest	0.00	0.00	0.00	100
fox	0.00	0.00	0.00	100
girl	0.00	0.00	0.00	100
hamster	0.00	0.00	0.00	100
house	0.00	0.00	0.00	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.00	0.00	0.00	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.00	0.00	0.00	100
leopard	0.00	0.00	0.00	100
lion	0.01	1.00	0.02	100
lizard	0.00	0.00	0.00	100
lobster	0.00	0.00	0.00	100
man	0.00	0.00	0.00	100
maple_tree	0.00	0.00	0.00	100
motorcycle	0.00	0.00	0.00	100
mountain	0.00	0.00	0.00	100
mouse	0.00	0.00	0.00	100
mushroom	0.00	0.00	0.00	100
oak_tree	0.00	0.00	0.00	100
orange	0.00	0.00	0.00	100
orchid	0.00	0.00	0.00	100
otter	0.00	0.00	0.00	100
palm_tree	0.00	0.00	0.00	100
pear	0.00	0.00	0.00	100
pickup_truck	0.00	0.00	0.00	100

```
    pine_tree       0.00      0.00      0.00       100
        plain       0.00      0.00      0.00       100
        plate       0.00      0.00      0.00       100
        poppy       0.00      0.00      0.00       100
    porcupine       0.00      0.00      0.00       100
       possum       0.00      0.00      0.00       100
       rabbit       0.00      0.00      0.00       100
      raccoon       0.00      0.00      0.00       100
          ray       0.00      0.00      0.00       100
         road       0.00      0.00      0.00       100
       rocket       0.00      0.00      0.00       100
         rose       0.00      0.00      0.00       100
          sea       0.00      0.00      0.00       100
         seal       0.00      0.00      0.00       100
        shark       0.00      0.00      0.00       100
        shrew       0.00      0.00      0.00       100
        skunk       0.00      0.00      0.00       100
    skyscraper       0.00      0.00      0.00       100
        snail       0.00      0.00      0.00       100
        snake       0.00      0.00      0.00       100
       spider       0.00      0.00      0.00       100
     squirrel       0.00      0.00      0.00       100
    streetcar       0.00      0.00      0.00       100
    sunflower       0.00      0.00      0.00       100
  sweet_pepper       0.00      0.00      0.00       100
        table       0.00      0.00      0.00       100
         tank       0.00      0.00      0.00       100
    telephone       0.00      0.00      0.00       100
   television       0.00      0.00      0.00       100
        tiger       0.00      0.00      0.00       100
      tractor       0.00      0.00      0.00       100
        train       0.00      0.00      0.00       100
        trout       0.00      0.00      0.00       100
        tulip       0.00      0.00      0.00       100
       turtle       0.00      0.00      0.00       100
     wardrobe       0.00      0.00      0.00       100
        whale       0.00      0.00      0.00       100
   willow_tree       0.00      0.00      0.00       100
         wolf       0.00      0.00      0.00       100
        woman       0.00      0.00      0.00       100
         worm       0.00      0.00      0.00       100

     accuracy                           0.01     10000
    macro avg       0.00      0.01      0.00     10000
 weighted avg       0.00      0.01      0.00     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
```

```
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

AlexNet

[ ]:
```
y_pred = predict(alexnet,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.76      0.75      0.75       100
aquarium_fish       0.51      0.69      0.59       100
        baby       0.37      0.44      0.40       100
        bear       0.41      0.28      0.33       100
      beaver       0.35      0.34      0.35       100
         bed       0.34      0.56      0.42       100
         bee       0.57      0.56      0.57       100
      beetle       0.57      0.47      0.51       100
     bicycle       0.78      0.50      0.61       100
      bottle       0.69      0.63      0.66       100
        bowl       0.36      0.31      0.33       100
         boy       0.32      0.40      0.36       100
      bridge       0.75      0.47      0.58       100
         bus       0.40      0.49      0.44       100
   butterfly       0.60      0.40      0.48       100
       camel       0.54      0.40      0.46       100
         can       0.57      0.43      0.49       100
      castle       0.77      0.68      0.72       100
  caterpillar       0.36      0.40      0.38       100
      cattle       0.59      0.26      0.36       100
       chair       0.76      0.65      0.70       100
   chimpanzee       0.63      0.66      0.65       100
       clock       0.57      0.39      0.46       100
       cloud       0.61      0.73      0.66       100
    cockroach       0.63      0.72      0.67       100
```

couch	0.30	0.37	0.33	100
crab	0.49	0.42	0.45	100
crocodile	0.29	0.35	0.32	100
cup	0.57	0.64	0.60	100
dinosaur	0.53	0.41	0.46	100
dolphin	0.40	0.48	0.44	100
elephant	0.58	0.45	0.51	100
flatfish	0.40	0.44	0.42	100
forest	0.52	0.43	0.47	100
fox	0.47	0.37	0.42	100
girl	0.42	0.22	0.29	100
hamster	0.36	0.63	0.45	100
house	0.48	0.37	0.42	100
kangaroo	0.36	0.32	0.34	100
keyboard	0.53	0.72	0.61	100
lamp	0.45	0.33	0.38	100
lawn_mower	0.70	0.62	0.66	100
leopard	0.54	0.44	0.48	100
lion	0.44	0.54	0.49	100
lizard	0.35	0.25	0.29	100
lobster	0.30	0.28	0.29	100
man	0.49	0.23	0.31	100
maple_tree	0.49	0.59	0.54	100
motorcycle	0.78	0.65	0.71	100
mountain	0.67	0.58	0.62	100
mouse	0.26	0.35	0.30	100
mushroom	0.56	0.49	0.52	100
oak_tree	0.54	0.75	0.63	100
orange	0.54	0.84	0.66	100
orchid	0.59	0.51	0.55	100
otter	0.30	0.17	0.22	100
palm_tree	0.61	0.71	0.66	100
pear	0.53	0.60	0.56	100
pickup_truck	0.64	0.54	0.59	100
pine_tree	0.45	0.41	0.43	100
plain	0.80	0.78	0.79	100
plate	0.58	0.63	0.60	100
poppy	0.56	0.68	0.62	100
porcupine	0.47	0.47	0.47	100
possum	0.32	0.30	0.31	100
rabbit	0.32	0.30	0.31	100
raccoon	0.45	0.54	0.49	100
ray	0.30	0.54	0.39	100
road	0.84	0.69	0.76	100
rocket	0.53	0.69	0.60	100
rose	0.44	0.54	0.48	100
sea	0.68	0.67	0.67	100
seal	0.22	0.26	0.24	100

```
           shark       0.36      0.40      0.38       100
           shrew       0.28      0.34      0.31       100
           skunk       0.80      0.76      0.78       100
      skyscraper       0.70      0.62      0.66       100
           snail       0.37      0.48      0.42       100
           snake       0.35      0.25      0.29       100
          spider       0.52      0.44      0.48       100
        squirrel       0.17      0.26      0.20       100
       streetcar       0.49      0.46      0.48       100
       sunflower       0.82      0.77      0.79       100
     sweet_pepper       0.49      0.44      0.46       100
           table       0.42      0.29      0.34       100
            tank       0.72      0.57      0.64       100
       telephone       0.55      0.48      0.51       100
      television       0.59      0.61      0.60       100
           tiger       0.54      0.45      0.49       100
         tractor       0.46      0.50      0.48       100
           train       0.36      0.60      0.45       100
           trout       0.53      0.70      0.61       100
           tulip       0.49      0.21      0.29       100
          turtle       0.18      0.27      0.22       100
        wardrobe       0.72      0.84      0.78       100
           whale       0.58      0.51      0.54       100
      willow_tree       0.48      0.33      0.39       100
            wolf       0.43      0.54      0.48       100
           woman       0.29      0.17      0.21       100
            worm       0.49      0.57      0.53       100

        accuracy                           0.49     10000
       macro avg       0.50      0.49      0.49     10000
    weighted avg       0.50      0.49      0.49     10000
```

Apenas um Epoch

Resetando os Modelos

[ ]:
```
cnn = torch.load('cnn0.pkl')
lenet = torch.load('lenet0.pkl')
alexnet = torch.load('alexnet0.pkl')
```

Treino

[ ]:
```
num_epochs = 1
```

Nosso Modelo

[ ]:
```
fit(cnn,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/1]: Loss : 4.216198087215424 Step: 1000
The Training is Finished
```

LeNet

```
[ ]: fit(lenet,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")

Epoch:  [1/1]: Loss : 4.617201194763184 Step: 1000
The Training is Finished
```

AlexNet

```
[ ]: fit(alexnet,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/1]: Loss : 4.605095304965973 Step: 1000
The Training is Finished
```

Salvando os Modelos

```
[ ]: torch.save(cnn,'cnn_1ep.pkl')
torch.save(lenet,'lenet_1ep.pkl')
torch.save(alexnet,'alexnet_1ep.pkl')
```

Test Carregando os Modelos

```
[ ]: cnn = torch.load('cnn_1ep.pkl')
lenet = torch.load('lenet_1ep.pkl')
alexnet = torch.load('alexnet_1ep.pkl')
```

Nosso Modelo

```
[ ]: test(cnn,dl_test)
```

```
Accuracy =  1485 / 10000 14.85 %
```

LeNet

```
[ ]: test(lenet,dl_test)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")

Accuracy =  100 / 10000 1.0 %
```

AlexNet

[ ]:
```
test(alexnet,dl_test)
```

```
Accuracy =  181 / 10000 1.81 %
```

Métricas de Desempenho Nosso Modelo

[ ]:
```
y_pred = predict(cnn,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.38      0.21      0.27       100
aquarium_fish       0.16      0.21      0.18       100
        baby       0.06      0.36      0.11       100
        bear       0.10      0.10      0.10       100
      beaver       1.00      0.01      0.02       100
         bed       0.00      0.00      0.00       100
         bee       0.31      0.04      0.07       100
      beetle       0.00      0.00      0.00       100
     bicycle       0.19      0.09      0.12       100
      bottle       0.13      0.09      0.11       100
        bowl       0.00      0.00      0.00       100
         boy       0.21      0.03      0.05       100
      bridge       0.19      0.08      0.11       100
         bus       0.12      0.38      0.18       100
   butterfly       0.12      0.10      0.11       100
       camel       0.08      0.13      0.10       100
         can       0.00      0.00      0.00       100
      castle       0.15      0.46      0.22       100
  caterpillar       0.12      0.04      0.06       100
      cattle       0.00      0.00      0.00       100
       chair       0.28      0.38      0.32       100
   chimpanzee       0.18      0.17      0.17       100
       clock       0.12      0.02      0.03       100
       cloud       0.17      0.67      0.27       100
   cockroach       0.19      0.69      0.30       100
       couch       0.33      0.04      0.07       100
        crab       0.12      0.12      0.12       100
   crocodile       0.12      0.01      0.02       100
         cup       0.10      0.19      0.13       100
    dinosaur       0.35      0.06      0.10       100
     dolphin       0.17      0.01      0.02       100
    elephant       0.23      0.03      0.05       100
    flatfish       0.00      0.00      0.00       100
      forest       0.19      0.05      0.08       100
         fox       0.08      0.07      0.08       100
        girl       0.08      0.01      0.02       100
```

hamster	0.18	0.13	0.15	100
house	0.12	0.05	0.07	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.04	0.03	0.03	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.47	0.42	0.44	100
leopard	0.15	0.11	0.13	100
lion	0.10	0.42	0.17	100
lizard	0.00	0.00	0.00	100
lobster	0.14	0.03	0.05	100
man	0.08	0.04	0.05	100
maple_tree	0.46	0.16	0.24	100
motorcycle	0.45	0.29	0.35	100
mountain	0.15	0.36	0.21	100
mouse	0.00	0.00	0.00	100
mushroom	0.22	0.08	0.12	100
oak_tree	0.48	0.62	0.54	100
orange	0.00	0.00	0.00	100
orchid	0.33	0.37	0.35	100
otter	0.08	0.03	0.04	100
palm_tree	0.30	0.09	0.14	100
pear	0.09	0.55	0.16	100
pickup_truck	0.18	0.11	0.14	100
pine_tree	0.25	0.01	0.02	100
plain	0.36	0.62	0.46	100
plate	0.11	0.28	0.16	100
poppy	0.38	0.27	0.32	100
porcupine	0.06	0.39	0.10	100
possum	0.00	0.00	0.00	100
rabbit	0.04	0.09	0.06	100
raccoon	0.06	0.22	0.10	100
ray	0.12	0.04	0.06	100
road	0.30	0.39	0.34	100
rocket	0.00	0.00	0.00	100
rose	0.39	0.12	0.18	100
sea	0.25	0.01	0.02	100
seal	0.00	0.00	0.00	100
shark	0.16	0.62	0.25	100
shrew	0.03	0.01	0.01	100
skunk	0.40	0.27	0.32	100
skyscraper	0.31	0.09	0.14	100
snail	0.08	0.01	0.02	100
snake	0.06	0.05	0.05	100
spider	0.19	0.08	0.11	100
squirrel	0.00	0.00	0.00	100
streetcar	0.00	0.00	0.00	100
sunflower	0.66	0.37	0.47	100
sweet_pepper	0.17	0.03	0.05	100

```
        table       0.07      0.02      0.03       100
         tank       0.15      0.17      0.16       100
    telephone       0.23      0.24      0.23       100
   television       0.00      0.00      0.00       100
        tiger       0.08      0.13      0.10       100
      tractor       0.09      0.05      0.06       100
        train       0.14      0.17      0.15       100
        trout       0.22      0.04      0.07       100
        tulip       0.17      0.01      0.02       100
       turtle       0.05      0.01      0.02       100
     wardrobe       0.30      0.38      0.34       100
        whale       0.29      0.11      0.16       100
  willow_tree       0.12      0.35      0.18       100
         wolf       0.09      0.44      0.15       100
        woman       0.02      0.02      0.02       100
         worm       0.00      0.00      0.00       100

     accuracy                           0.15     10000
    macro avg       0.16      0.15      0.12     10000
 weighted avg       0.16      0.15      0.12     10000
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

LeNet

[ ]:
```
y_pred = predict(lenet,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

	precision	recall	f1-score	support
apple	0.00	0.00	0.00	100
aquarium_fish	0.00	0.00	0.00	100
baby	0.00	0.00	0.00	100
bear	0.00	0.00	0.00	100
beaver	0.00	0.00	0.00	100
bed	0.00	0.00	0.00	100
bee	0.00	0.00	0.00	100
beetle	0.00	0.00	0.00	100
bicycle	0.00	0.00	0.00	100
bottle	0.00	0.00	0.00	100
bowl	0.00	0.00	0.00	100
boy	0.00	0.00	0.00	100
bridge	0.00	0.00	0.00	100
bus	0.00	0.00	0.00	100
butterfly	0.00	0.00	0.00	100
camel	0.00	0.00	0.00	100
can	0.00	0.00	0.00	100
castle	0.00	0.00	0.00	100
caterpillar	0.00	0.00	0.00	100
cattle	0.00	0.00	0.00	100
chair	0.00	0.00	0.00	100
chimpanzee	0.00	0.00	0.00	100
clock	0.00	0.00	0.00	100
cloud	0.00	0.00	0.00	100
cockroach	0.00	0.00	0.00	100
couch	0.00	0.00	0.00	100
crab	0.00	0.00	0.00	100
crocodile	0.00	0.00	0.00	100
cup	0.00	0.00	0.00	100
dinosaur	0.00	0.00	0.00	100
dolphin	0.00	0.00	0.00	100
elephant	0.00	0.00	0.00	100
flatfish	0.00	0.00	0.00	100
forest	0.00	0.00	0.00	100
fox	0.00	0.00	0.00	100
girl	0.00	0.00	0.00	100
hamster	0.00	0.00	0.00	100
house	0.00	0.00	0.00	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.00	0.00	0.00	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.00	0.00	0.00	100
leopard	0.00	0.00	0.00	100
lion	0.00	0.00	0.00	100
lizard	0.00	0.00	0.00	100
lobster	0.00	0.00	0.00	100

man	0.00	0.00	0.00	100
maple_tree	0.00	0.00	0.00	100
motorcycle	0.00	0.00	0.00	100
mountain	0.00	0.00	0.00	100
mouse	0.00	0.00	0.00	100
mushroom	0.00	0.00	0.00	100
oak_tree	0.00	0.00	0.00	100
orange	0.00	0.00	0.00	100
orchid	0.00	0.00	0.00	100
otter	0.00	0.00	0.00	100
palm_tree	0.00	0.00	0.00	100
pear	0.00	0.00	0.00	100
pickup_truck	0.00	0.00	0.00	100
pine_tree	0.00	0.00	0.00	100
plain	0.00	0.00	0.00	100
plate	0.00	0.00	0.00	100
poppy	0.00	0.00	0.00	100
porcupine	0.00	0.00	0.00	100
possum	0.00	0.00	0.00	100
rabbit	0.00	0.00	0.00	100
raccoon	0.00	0.00	0.00	100
ray	0.00	0.00	0.00	100
road	0.00	0.00	0.00	100
rocket	0.00	0.00	0.00	100
rose	0.00	0.00	0.00	100
sea	0.00	0.00	0.00	100
seal	0.00	0.00	0.00	100
shark	0.00	0.00	0.00	100
shrew	0.00	0.00	0.00	100
skunk	0.00	0.00	0.00	100
skyscraper	0.00	0.00	0.00	100
snail	0.00	0.00	0.00	100
snake	0.00	0.00	0.00	100
spider	0.00	0.00	0.00	100
squirrel	0.00	0.00	0.00	100
streetcar	0.00	0.00	0.00	100
sunflower	0.00	0.00	0.00	100
sweet_pepper	0.00	0.00	0.00	100
table	0.00	0.00	0.00	100
tank	0.00	0.00	0.00	100
telephone	0.00	0.00	0.00	100
television	0.00	0.00	0.00	100
tiger	0.00	0.00	0.00	100
tractor	0.00	0.00	0.00	100
train	0.00	0.00	0.00	100
trout	0.00	0.00	0.00	100
tulip	0.01	1.00	0.02	100
turtle	0.00	0.00	0.00	100

```
    wardrobe       0.00      0.00      0.00       100
       whale       0.00      0.00      0.00       100
 willow_tree       0.00      0.00      0.00       100
        wolf       0.00      0.00      0.00       100
       woman       0.00      0.00      0.00       100
        worm       0.00      0.00      0.00       100

    accuracy                           0.01     10000
   macro avg       0.00      0.01      0.00     10000
weighted avg       0.00      0.01      0.00     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

AlexNet

[ ]:
```python
y_pred = predict(alexnet,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.00      0.00      0.00       100
aquarium_fish       0.00      0.00      0.00       100
        baby       0.00      0.00      0.00       100
        bear       0.00      0.00      0.00       100
      beaver       0.00      0.00      0.00       100
         bed       0.00      0.00      0.00       100
         bee       0.00      0.00      0.00       100
      beetle       0.00      0.00      0.00       100
     bicycle       0.00      0.00      0.00       100
      bottle       0.00      0.00      0.00       100
        bowl       0.00      0.00      0.00       100
         boy       0.00      0.00      0.00       100
      bridge       0.00      0.00      0.00       100
```

bus	0.02	0.56	0.04	100
butterfly	0.00	0.00	0.00	100
camel	0.00	0.00	0.00	100
can	0.00	0.00	0.00	100
castle	0.00	0.00	0.00	100
caterpillar	0.00	0.00	0.00	100
cattle	0.00	0.00	0.00	100
chair	0.29	0.02	0.04	100
chimpanzee	0.08	0.05	0.06	100
clock	0.00	0.00	0.00	100
cloud	0.00	0.00	0.00	100
cockroach	0.00	0.00	0.00	100
couch	0.00	0.00	0.00	100
crab	0.00	0.00	0.00	100
crocodile	0.00	0.00	0.00	100
cup	0.00	0.00	0.00	100
dinosaur	0.00	0.00	0.00	100
dolphin	0.00	0.00	0.00	100
elephant	0.00	0.00	0.00	100
flatfish	0.00	0.00	0.00	100
forest	0.00	0.00	0.00	100
fox	0.00	0.00	0.00	100
girl	0.00	0.00	0.00	100
hamster	0.03	0.52	0.05	100
house	0.00	0.00	0.00	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.00	0.00	0.00	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.00	0.00	0.00	100
leopard	0.00	0.00	0.00	100
lion	0.00	0.00	0.00	100
lizard	0.00	0.00	0.00	100
lobster	0.00	0.00	0.00	100
man	0.00	0.00	0.00	100
maple_tree	0.00	0.00	0.00	100
motorcycle	0.00	0.00	0.00	100
mountain	0.00	0.00	0.00	100
mouse	0.00	0.00	0.00	100
mushroom	0.00	0.00	0.00	100
oak_tree	0.00	0.00	0.00	100
orange	0.00	0.00	0.00	100
orchid	0.00	0.00	0.00	100
otter	0.00	0.00	0.00	100
palm_tree	0.00	0.00	0.00	100
pear	0.00	0.00	0.00	100
pickup_truck	0.01	0.43	0.02	100
pine_tree	0.00	0.00	0.00	100
plain	0.00	0.00	0.00	100

```
           plate       0.00      0.00      0.00       100
           poppy       0.00      0.00      0.00       100
       porcupine       0.00      0.00      0.00       100
          possum       0.02      0.23      0.03       100
          rabbit       0.00      0.00      0.00       100
         raccoon       0.00      0.00      0.00       100
             ray       0.00      0.00      0.00       100
            road       0.00      0.00      0.00       100
          rocket       0.00      0.00      0.00       100
            rose       0.00      0.00      0.00       100
             sea       0.00      0.00      0.00       100
            seal       0.00      0.00      0.00       100
           shark       0.00      0.00      0.00       100
           shrew       0.00      0.00      0.00       100
           skunk       0.00      0.00      0.00       100
      skyscraper       0.00      0.00      0.00       100
           snail       0.00      0.00      0.00       100
           snake       0.00      0.00      0.00       100
          spider       0.00      0.00      0.00       100
        squirrel       0.00      0.00      0.00       100
       streetcar       0.00      0.00      0.00       100
       sunflower       0.00      0.00      0.00       100
     sweet_pepper       0.00      0.00      0.00       100
           table       0.00      0.00      0.00       100
            tank       0.00      0.00      0.00       100
       telephone       0.00      0.00      0.00       100
      television       0.00      0.00      0.00       100
           tiger       0.00      0.00      0.00       100
         tractor       0.00      0.00      0.00       100
           train       0.00      0.00      0.00       100
           trout       0.00      0.00      0.00       100
           tulip       0.00      0.00      0.00       100
          turtle       0.00      0.00      0.00       100
        wardrobe       0.00      0.00      0.00       100
           whale       0.00      0.00      0.00       100
      willow_tree       0.00      0.00      0.00       100
            wolf       0.00      0.00      0.00       100
           woman       0.00      0.00      0.00       100
            worm       0.00      0.00      0.00       100

        accuracy                           0.02     10000
       macro avg       0.00      0.02      0.00     10000
    weighted avg       0.00      0.02      0.00     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
```

```
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Limitando os Dados

Resetando os Modelos

[ ]:
```
cnn = torch.load('cnn0.pkl')
lenet = torch.load('lenet0.pkl')
alexnet = torch.load('alexnet0.pkl')
```

Treino

[ ]:
```
num_epochs = 50
```

Nosso Modelo

[ ]:
```
fit(cnn,dl_test,num_epochs = num_epochs,steps = 100)
```

```
Epoch:  [1/50]: Loss : 4.585521955490112 Step: 100
Epoch:  [1/50]: Loss : 4.5166109895706175 Step: 200
Epoch:  [2/50]: Loss : 4.394777455329895 Step: 100
Epoch:  [2/50]: Loss : 4.321760921478272 Step: 200
Epoch:  [3/50]: Loss : 4.130421192646026 Step: 100
Epoch:  [3/50]: Loss : 4.0332274031639095 Step: 200
Epoch:  [4/50]: Loss : 3.8747202587127685 Step: 100
Epoch:  [4/50]: Loss : 3.8110615372657777 Step: 200
Epoch:  [5/50]: Loss : 3.654009313583374 Step: 100
Epoch:  [5/50]: Loss : 3.6276140213012695 Step: 200
Epoch:  [6/50]: Loss : 3.497863392829895 Step: 100
Epoch:  [6/50]: Loss : 3.4518131017684937 Step: 200
Epoch:  [7/50]: Loss : 3.3328936290740967 Step: 100
Epoch:  [7/50]: Loss : 3.315926432609558 Step: 200
Epoch:  [8/50]: Loss : 3.181609282493591 Step: 100
Epoch:  [8/50]: Loss : 3.1602194356918334 Step: 200
Epoch:  [9/50]: Loss : 2.990006895065308 Step: 100
Epoch:  [9/50]: Loss : 3.0531459069252014 Step: 200
Epoch:  [10/50]: Loss : 2.869357798099518 Step: 100
Epoch:  [10/50]: Loss : 2.885706388950348 Step: 200
```

```
Epoch:  [11/50]: Loss : 2.7074213075637816 Step: 100
Epoch:  [11/50]: Loss : 2.7916078996658324 Step: 200
Epoch:  [12/50]: Loss : 2.616760585308075 Step: 100
Epoch:  [12/50]: Loss : 2.6382871985435488 Step: 200
Epoch:  [13/50]: Loss : 2.4328749346733094 Step: 100
Epoch:  [13/50]: Loss : 2.5112745225429536 Step: 200
Epoch:  [14/50]: Loss : 2.3382361733913424 Step: 100
Epoch:  [14/50]: Loss : 2.370347763299942 Step: 200
Epoch:  [15/50]: Loss : 2.265431500673294 Step: 100
Epoch:  [15/50]: Loss : 2.2347524654865265 Step: 200
Epoch:  [16/50]: Loss : 2.077880687713623 Step: 100
Epoch:  [16/50]: Loss : 2.1493660509586334 Step: 200
Epoch:  [17/50]: Loss : 1.9841097021102905 Step: 100
Epoch:  [17/50]: Loss : 2.0327138125896456 Step: 200
Epoch:  [18/50]: Loss : 1.876896151304245 Step: 100
Epoch:  [18/50]: Loss : 1.8930393421649934 Step: 200
Epoch:  [19/50]: Loss : 1.751795210838318 Step: 100
Epoch:  [19/50]: Loss : 1.7534850907325745 Step: 200
Epoch:  [20/50]: Loss : 1.6056821167469024 Step: 100
Epoch:  [20/50]: Loss : 1.6637748634815217 Step: 200
Epoch:  [21/50]: Loss : 1.5193364024162292 Step: 100
Epoch:  [21/50]: Loss : 1.5300781059265136 Step: 200
Epoch:  [22/50]: Loss : 1.342365819811821 Step: 100
Epoch:  [22/50]: Loss : 1.433269771337509 Step: 200
Epoch:  [23/50]: Loss : 1.2677922189235686 Step: 100
Epoch:  [23/50]: Loss : 1.269913094639778 Step: 200
Epoch:  [24/50]: Loss : 1.1523679798841477 Step: 100
Epoch:  [24/50]: Loss : 1.2352930688858033 Step: 200
Epoch:  [25/50]: Loss : 1.0368765729665756 Step: 100
Epoch:  [25/50]: Loss : 1.0956781482696534 Step: 200
Epoch:  [26/50]: Loss : 0.9151980912685395 Step: 100
Epoch:  [26/50]: Loss : 0.955437285900116 Step: 200
Epoch:  [27/50]: Loss : 0.8301325190067291 Step: 100
Epoch:  [27/50]: Loss : 0.8801485866308212 Step: 200
Epoch:  [28/50]: Loss : 0.7351470431685447 Step: 100
Epoch:  [28/50]: Loss : 0.7827376899123192 Step: 200
Epoch:  [29/50]: Loss : 0.644067864716053 Step: 100
Epoch:  [29/50]: Loss : 0.6883971628546715 Step: 200
Epoch:  [30/50]: Loss : 0.5743502920866013 Step: 100
Epoch:  [30/50]: Loss : 0.5884467777609825 Step: 200
Epoch:  [31/50]: Loss : 0.4988648322224617 Step: 100
Epoch:  [31/50]: Loss : 0.5046117988228798 Step: 200
Epoch:  [32/50]: Loss : 0.42720905408263204 Step: 100
Epoch:  [32/50]: Loss : 0.4695203945040703 Step: 200
Epoch:  [33/50]: Loss : 0.368186821937561 Step: 100
Epoch:  [33/50]: Loss : 0.3999771513044834 Step: 200
Epoch:  [34/50]: Loss : 0.31289603501558305 Step: 100
Epoch:  [34/50]: Loss : 0.3568172006309032 Step: 200
```

```
Epoch:  [35/50]: Loss : 0.2784688498079777 Step: 100
Epoch:  [35/50]: Loss : 0.3182652872055769 Step: 200
Epoch:  [36/50]: Loss : 0.23901019424200057 Step: 100
Epoch:  [36/50]: Loss : 0.24725639685988426 Step: 200
Epoch:  [37/50]: Loss : 0.21738560661673545 Step: 100
Epoch:  [37/50]: Loss : 0.24048853814601898 Step: 200
Epoch:  [38/50]: Loss : 0.1832705892622471 Step: 100
Epoch:  [38/50]: Loss : 0.19369059816002845 Step: 200
Epoch:  [39/50]: Loss : 0.1615708514302969 Step: 100
Epoch:  [39/50]: Loss : 0.19550646468997002 Step: 200
Epoch:  [40/50]: Loss : 0.14552062518894673 Step: 100
Epoch:  [40/50]: Loss : 0.16721648398786784 Step: 200
Epoch:  [41/50]: Loss : 0.14805263191461562 Step: 100
Epoch:  [41/50]: Loss : 0.15284639205783607 Step: 200
Epoch:  [42/50]: Loss : 0.12560391549021005 Step: 100
Epoch:  [42/50]: Loss : 0.12299305517226458 Step: 200
Epoch:  [43/50]: Loss : 0.12110808793455362 Step: 100
Epoch:  [43/50]: Loss : 0.1247514221817255 Step: 200
Epoch:  [44/50]: Loss : 0.1014193619042635 Step: 100
Epoch:  [44/50]: Loss : 0.10421645812690258 Step: 200
Epoch:  [45/50]: Loss : 0.09413395496085286 Step: 100
Epoch:  [45/50]: Loss : 0.08621013730764389 Step: 200
Epoch:  [46/50]: Loss : 0.09232778303325176 Step: 100
Epoch:  [46/50]: Loss : 0.09081011682748795 Step: 200
Epoch:  [47/50]: Loss : 0.08199582852423191 Step: 100
Epoch:  [47/50]: Loss : 0.08534581061452627 Step: 200
Epoch:  [48/50]: Loss : 0.06550162659958005 Step: 100
Epoch:  [48/50]: Loss : 0.07313248405233025 Step: 200
Epoch:  [49/50]: Loss : 0.0681732274685055 Step: 100
Epoch:  [49/50]: Loss : 0.06192491206340492 Step: 200
Epoch:  [50/50]: Loss : 0.0594903701171279 Step: 100
Epoch:  [50/50]: Loss : 0.07361003061756492 Step: 200
The Training is Finished
```

[ ]:
```
torch.save(cnn,'cnn_1.pkl')
```

LeNet

[ ]:
```
fit(lenet,dl_test,num_epochs = num_epochs,steps = 100)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")

Epoch:  [1/50]: Loss : 4.642161245346069 Step: 100
Epoch:  [1/50]: Loss : 4.633384165763855 Step: 200
Epoch:  [2/50]: Loss : 4.6211574840545655 Step: 100
Epoch:  [2/50]: Loss : 4.614845385551453 Step: 200
```

```
Epoch:  [3/50]: Loss : 4.610919785499573 Step: 100
Epoch:  [3/50]: Loss : 4.61040488243103 Step: 200
Epoch:  [4/50]: Loss : 4.6080766201019285 Step: 100
Epoch:  [4/50]: Loss : 4.609831328392029 Step: 200
Epoch:  [5/50]: Loss : 4.606768436431885 Step: 100
Epoch:  [5/50]: Loss : 4.6088636684417725 Step: 200
Epoch:  [6/50]: Loss : 4.606490845680237 Step: 100
Epoch:  [6/50]: Loss : 4.608329067230224 Step: 200
Epoch:  [7/50]: Loss : 4.606142420768737 Step: 100
Epoch:  [7/50]: Loss : 4.6085197591781615 Step: 200
Epoch:  [8/50]: Loss : 4.606481184959412 Step: 100
Epoch:  [8/50]: Loss : 4.6080841541290285 Step: 200
Epoch:  [9/50]: Loss : 4.605948758125305 Step: 100
Epoch:  [9/50]: Loss : 4.608695497512818 Step: 200
Epoch:  [10/50]: Loss : 4.606397433280945 Step: 100
Epoch:  [10/50]: Loss : 4.6082532024383545 Step: 200
Epoch:  [11/50]: Loss : 4.606356377601624 Step: 100
Epoch:  [11/50]: Loss : 4.60848641872406 Step: 200
Epoch:  [12/50]: Loss : 4.605964813232422 Step: 100
Epoch:  [12/50]: Loss : 4.608572278022766 Step: 200
Epoch:  [13/50]: Loss : 4.606006464958191 Step: 100
Epoch:  [13/50]: Loss : 4.608436012268067 Step: 200
Epoch:  [14/50]: Loss : 4.606490602493286 Step: 100
Epoch:  [14/50]: Loss : 4.607942209243775 Step: 200
Epoch:  [15/50]: Loss : 4.606551342010498 Step: 100
Epoch:  [15/50]: Loss : 4.608183484077454 Step: 200
Epoch:  [16/50]: Loss : 4.606465516090393 Step: 100
Epoch:  [16/50]: Loss : 4.608008785247803 Step: 200
Epoch:  [17/50]: Loss : 4.606126112937927 Step: 100
Epoch:  [17/50]: Loss : 4.608637638092041 Step: 200
Epoch:  [18/50]: Loss : 4.6063651275634765 Step: 100
Epoch:  [18/50]: Loss : 4.608332848548889 Step: 200
Epoch:  [19/50]: Loss : 4.606361541748047 Step: 100
Epoch:  [19/50]: Loss : 4.608245549201965 Step: 200
Epoch:  [20/50]: Loss : 4.606379871368408 Step: 100
Epoch:  [20/50]: Loss : 4.608355660438537 Step: 200
Epoch:  [21/50]: Loss : 4.606382474899292 Step: 100
Epoch:  [21/50]: Loss : 4.608031234741211 Step: 200
Epoch:  [22/50]: Loss : 4.606344771385193 Step: 100
Epoch:  [22/50]: Loss : 4.607981514930725 Step: 200
Epoch:  [23/50]: Loss : 4.606438345909119 Step: 100
Epoch:  [23/50]: Loss : 4.608064737319946 Step: 200
Epoch:  [24/50]: Loss : 4.606378836631775 Step: 100
Epoch:  [24/50]: Loss : 4.608230895996094 Step: 200
Epoch:  [25/50]: Loss : 4.6061441659927365 Step: 100
Epoch:  [25/50]: Loss : 4.608282823562622 Step: 200
Epoch:  [26/50]: Loss : 4.606385083198547 Step: 100
Epoch:  [26/50]: Loss : 4.608284406661987 Step: 200
```

```
Epoch:  [27/50]: Loss : 4.606132645606994 Step: 100
Epoch:  [27/50]: Loss : 4.608312678337097 Step: 200
Epoch:  [28/50]: Loss : 4.606413297653198 Step: 100
Epoch:  [28/50]: Loss : 4.608007831573486 Step: 200
Epoch:  [29/50]: Loss : 4.606726222038269 Step: 100
Epoch:  [29/50]: Loss : 4.607874851226807 Step: 200
Epoch:  [30/50]: Loss : 4.606262536048889 Step: 100
Epoch:  [30/50]: Loss : 4.608131046295166 Step: 200
Epoch:  [31/50]: Loss : 4.6060568046569825 Step: 100
Epoch:  [31/50]: Loss : 4.608455958366394 Step: 200
Epoch:  [32/50]: Loss : 4.606396493911743 Step: 100
Epoch:  [32/50]: Loss : 4.607971086502075 Step: 200
Epoch:  [33/50]: Loss : 4.606327915191651 Step: 100
Epoch:  [33/50]: Loss : 4.608219332695008 Step: 200
Epoch:  [34/50]: Loss : 4.606200938224792 Step: 100
Epoch:  [34/50]: Loss : 4.6080788803100585 Step: 200
Epoch:  [35/50]: Loss : 4.606625723838806 Step: 100
Epoch:  [35/50]: Loss : 4.607926959991455 Step: 200
Epoch:  [36/50]: Loss : 4.60633355140686 Step: 100
Epoch:  [36/50]: Loss : 4.608342084884644 Step: 200
Epoch:  [37/50]: Loss : 4.606721453666687 Step: 100
Epoch:  [37/50]: Loss : 4.60783139705658 Step: 200
Epoch:  [38/50]: Loss : 4.606283030509949 Step: 100
Epoch:  [38/50]: Loss : 4.608215613365173 Step: 200
Epoch:  [39/50]: Loss : 4.6064786195755 Step: 100
Epoch:  [39/50]: Loss : 4.608042421340943 Step: 200
Epoch:  [40/50]: Loss : 4.605965099334717 Step: 100
Epoch:  [40/50]: Loss : 4.608651075363159 Step: 200
Epoch:  [41/50]: Loss : 4.605826539993286 Step: 100
Epoch:  [41/50]: Loss : 4.608798289299012 Step: 200
Epoch:  [42/50]: Loss : 4.606067790985107 Step: 100
Epoch:  [42/50]: Loss : 4.60810227394104 Step: 200
Epoch:  [43/50]: Loss : 4.606467943191529 Step: 100
Epoch:  [43/50]: Loss : 4.607897510528565 Step: 200
Epoch:  [44/50]: Loss : 4.606053447723388 Step: 100
Epoch:  [44/50]: Loss : 4.608398609161377 Step: 200
Epoch:  [45/50]: Loss : 4.606187133789063 Step: 100
Epoch:  [45/50]: Loss : 4.608052334785461 Step: 200
Epoch:  [46/50]: Loss : 4.60609384059906 Step: 100
Epoch:  [46/50]: Loss : 4.608213896751404 Step: 200
Epoch:  [47/50]: Loss : 4.606144390106201 Step: 100
Epoch:  [47/50]: Loss : 4.608354501724243 Step: 200
Epoch:  [48/50]: Loss : 4.605982422828674 Step: 100
Epoch:  [48/50]: Loss : 4.6085448789596555 Step: 200
Epoch:  [49/50]: Loss : 4.6066834831237795 Step: 100
Epoch:  [49/50]: Loss : 4.607908978462219 Step: 200
Epoch:  [50/50]: Loss : 4.606489596366882 Step: 100
Epoch:  [50/50]: Loss : 4.60810085773468 Step: 200
```

```
The Training is Finished
```

```
[ ]: torch.save(lenet,'lenet_1.pkl')
```

AlexNet

```
[ ]: fit(alexnet,dl_test,num_epochs = num_epochs,steps = 100)
```

```
Epoch:  [1/50]: Loss : 4.605181212425232 Step: 100
Epoch:  [1/50]: Loss : 4.605480585098267 Step: 200
Epoch:  [2/50]: Loss : 4.605196843147278 Step: 100
Epoch:  [2/50]: Loss : 4.6052999019622805 Step: 200
Epoch:  [3/50]: Loss : 4.605170774459839 Step: 100
Epoch:  [3/50]: Loss : 4.605043034553528 Step: 200
Epoch:  [4/50]: Loss : 4.6049329280853275 Step: 100
Epoch:  [4/50]: Loss : 4.604843511581421 Step: 200
Epoch:  [5/50]: Loss : 4.604665412902832 Step: 100
Epoch:  [5/50]: Loss : 4.604871072769165 Step: 200
Epoch:  [6/50]: Loss : 4.604425892829895 Step: 100
Epoch:  [6/50]: Loss : 4.604243221282959 Step: 200
Epoch:  [7/50]: Loss : 4.604097905158997 Step: 100
Epoch:  [7/50]: Loss : 4.604112396240234 Step: 200
Epoch:  [8/50]: Loss : 4.603303899765015 Step: 100
Epoch:  [8/50]: Loss : 4.603698468208313 Step: 200
Epoch:  [9/50]: Loss : 4.60281343460083 Step: 100
Epoch:  [9/50]: Loss : 4.602608375549316 Step: 200
Epoch:  [10/50]: Loss : 4.602212452888489 Step: 100
Epoch:  [10/50]: Loss : 4.601545100212097 Step: 200
Epoch:  [11/50]: Loss : 4.5994162940979 Step: 100
Epoch:  [11/50]: Loss : 4.598297085762024 Step: 200
Epoch:  [12/50]: Loss : 4.595289826393127 Step: 100
Epoch:  [12/50]: Loss : 4.589666242599487 Step: 200
Epoch:  [13/50]: Loss : 4.566724586486816 Step: 100
Epoch:  [13/50]: Loss : 4.527339463233948 Step: 200
Epoch:  [14/50]: Loss : 4.469729685783387 Step: 100
Epoch:  [14/50]: Loss : 4.447470378875733 Step: 200
Epoch:  [15/50]: Loss : 4.363091931343079 Step: 100
Epoch:  [15/50]: Loss : 4.30389370918274 Step: 200
Epoch:  [16/50]: Loss : 4.287655644416809 Step: 100
Epoch:  [16/50]: Loss : 4.245610792636871 Step: 200
Epoch:  [17/50]: Loss : 4.235225148200989 Step: 100
Epoch:  [17/50]: Loss : 4.199109172821045 Step: 200
Epoch:  [18/50]: Loss : 4.18893949508667 Step: 100
Epoch:  [18/50]: Loss : 4.170110771656036 Step: 200
Epoch:  [19/50]: Loss : 4.154323716163635 Step: 100
Epoch:  [19/50]: Loss : 4.125820026397705 Step: 200
Epoch:  [20/50]: Loss : 4.1174522662162785 Step: 100
Epoch:  [20/50]: Loss : 4.110755341053009 Step: 200
```

```
Epoch:  [21/50]: Loss : 4.079150629043579 Step: 100
Epoch:  [21/50]: Loss : 4.085169415473938 Step: 200
Epoch:  [22/50]: Loss : 4.044300622940064 Step: 100
Epoch:  [22/50]: Loss : 4.045781779289245 Step: 200
Epoch:  [23/50]: Loss : 3.9995115280151365 Step: 100
Epoch:  [23/50]: Loss : 4.019740581512451 Step: 200
Epoch:  [24/50]: Loss : 3.9664571475982666 Step: 100
Epoch:  [24/50]: Loss : 3.9604627776145933 Step: 200
Epoch:  [25/50]: Loss : 3.921112642288208 Step: 100
Epoch:  [25/50]: Loss : 3.912289364337921 Step: 200
Epoch:  [26/50]: Loss : 3.8728296160697937 Step: 100
Epoch:  [26/50]: Loss : 3.8670505142211913 Step: 200
Epoch:  [27/50]: Loss : 3.8636799669265747 Step: 100
Epoch:  [27/50]: Loss : 3.8087764263153074 Step: 200
Epoch:  [28/50]: Loss : 3.7722434902191164 Step: 100
Epoch:  [28/50]: Loss : 3.8089625692367552 Step: 200
Epoch:  [29/50]: Loss : 3.7642123198509214 Step: 100
Epoch:  [29/50]: Loss : 3.7755230259895325 Step: 200
Epoch:  [30/50]: Loss : 3.7496522283554077 Step: 100
Epoch:  [30/50]: Loss : 3.699940288066864 Step: 200
Epoch:  [31/50]: Loss : 3.716363387107849 Step: 100
Epoch:  [31/50]: Loss : 3.6754926443099976 Step: 200
Epoch:  [32/50]: Loss : 3.6369338607788086 Step: 100
Epoch:  [32/50]: Loss : 3.674951732158661 Step: 200
Epoch:  [33/50]: Loss : 3.611797556877136 Step: 100
Epoch:  [33/50]: Loss : 3.62436283826828 Step: 200
Epoch:  [34/50]: Loss : 3.5915687680244446 Step: 100
Epoch:  [34/50]: Loss : 3.5966719603538513 Step: 200
Epoch:  [35/50]: Loss : 3.5516582226753233 Step: 100
Epoch:  [35/50]: Loss : 3.555027599334717 Step: 200
Epoch:  [36/50]: Loss : 3.514039926528931 Step: 100
Epoch:  [36/50]: Loss : 3.5493685269355773 Step: 200
Epoch:  [37/50]: Loss : 3.4891528487205505 Step: 100
Epoch:  [37/50]: Loss : 3.481537251472473 Step: 200
Epoch:  [38/50]: Loss : 3.4593779182434083 Step: 100
Epoch:  [38/50]: Loss : 3.4697594046592712 Step: 200
Epoch:  [39/50]: Loss : 3.4345023036003113 Step: 100
Epoch:  [39/50]: Loss : 3.438198745250702 Step: 200
Epoch:  [40/50]: Loss : 3.4280765986442567 Step: 100
Epoch:  [40/50]: Loss : 3.3881680870056154 Step: 200
Epoch:  [41/50]: Loss : 3.3632861757278443 Step: 100
Epoch:  [41/50]: Loss : 3.3645659279823303 Step: 200
Epoch:  [42/50]: Loss : 3.3321446347236634 Step: 100
Epoch:  [42/50]: Loss : 3.349756586551666 Step: 200
Epoch:  [43/50]: Loss : 3.274549763202667 Step: 100
Epoch:  [43/50]: Loss : 3.283732249736786 Step: 200
Epoch:  [44/50]: Loss : 3.2383056020736696 Step: 100
Epoch:  [44/50]: Loss : 3.273409113883972 Step: 200
```

```
Epoch:  [45/50]: Loss : 3.2014054536819456 Step: 100
Epoch:  [45/50]: Loss : 3.2546771502494813 Step: 200
Epoch:  [46/50]: Loss : 3.1924059629440307 Step: 100
Epoch:  [46/50]: Loss : 3.189613642692566 Step: 200
Epoch:  [47/50]: Loss : 3.1687050223350526 Step: 100
Epoch:  [47/50]: Loss : 3.1340513396263123 Step: 200
Epoch:  [48/50]: Loss : 3.1143968653678895 Step: 100
Epoch:  [48/50]: Loss : 3.072686038017273 Step: 200
Epoch:  [49/50]: Loss : 3.052469878196716 Step: 100
Epoch:  [49/50]: Loss : 3.0600576877593992 Step: 200
Epoch:  [50/50]: Loss : 3.054283313751221 Step: 100
Epoch:  [50/50]: Loss : 3.039636936187744 Step: 200
The Training is Finished
```

[ ]:
```
torch.save(alexnet,'alexnet_1.pkl')
```

Salvando os Modelos

[ ]:
```
torch.save(cnn,'cnn_1.pkl')
torch.save(lenet,'lenet_1.pkl')
torch.save(alexnet,'alexnet_1.pkl')
```

Test Carregando os Modelos

[ ]:
```
cnn = torch.load('cnn_1.pkl')
lenet = torch.load('lenet_1.pkl')
alexnet = torch.load('alexnet_1.pkl')
```

Nosso Modelo

[ ]:
```
test(cnn,dl_train)
```

```
Accuracy =  15097 / 50000 30.194 %
```

LeNet

[ ]:
```
test(lenet,dl_train)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

```
Accuracy =  500 / 50000 1.0 %
```

AlexNet

[ ]:
```
test(alexnet,dl_train)
```

```
Accuracy =  9527 / 50000 19.054 %
```

Métricas de Desempenho Nosso Modelo

[ ]:
```
y_pred = predict(cnn,ds_train)
y_test = [label for image,label in ds_train]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.52      0.65      0.58       500
aquarium_fish       0.52      0.23      0.32       500
        baby       0.24      0.13      0.17       500
        bear       0.14      0.25      0.18       500
      beaver       0.12      0.10      0.11       500
         bed       0.28      0.11      0.16       500
         bee       0.27      0.25      0.26       500
      beetle       0.21      0.37      0.27       500
     bicycle       0.61      0.28      0.38       500
      bottle       0.36      0.50      0.42       500
        bowl       0.18      0.20      0.19       500
         boy       0.24      0.11      0.15       500
      bridge       0.30      0.24      0.26       500
         bus       0.32      0.25      0.28       500
   butterfly       0.16      0.36      0.22       500
       camel       0.23      0.14      0.17       500
         can       0.30      0.25      0.27       500
      castle       0.66      0.27      0.38       500
  caterpillar       0.15      0.20      0.17       500
      cattle       0.21      0.21      0.21       500
       chair       0.59      0.51      0.55       500
    chimpanzee       0.40      0.28      0.33       500
       clock       0.31      0.23      0.26       500
       cloud       0.43      0.58      0.49       500
    cockroach       0.35      0.49      0.41       500
       couch       0.26      0.25      0.26       500
        crab       0.24      0.22      0.23       500
   crocodile       0.21      0.16      0.18       500
         cup       0.42      0.55      0.48       500
    dinosaur       0.26      0.27      0.27       500
     dolphin       0.31      0.33      0.32       500
    elephant       0.27      0.35      0.31       500
    flatfish       0.20      0.21      0.20       500
      forest       0.33      0.37      0.35       500
         fox       0.31      0.16      0.21       500
        girl       0.15      0.18      0.16       500
     hamster       0.40      0.31      0.35       500
       house       0.25      0.15      0.19       500
    kangaroo       0.15      0.14      0.15       500
    keyboard       0.30      0.21      0.24       500
```

lamp	0.28	0.22	0.25	500
lawn_mower	0.48	0.52	0.50	500
leopard	0.34	0.30	0.32	500
lion	0.39	0.20	0.26	500
lizard	0.15	0.12	0.13	500
lobster	0.14	0.15	0.15	500
man	0.21	0.17	0.19	500
maple_tree	0.54	0.35	0.42	500
motorcycle	0.39	0.51	0.44	500
mountain	0.44	0.41	0.43	500
mouse	0.11	0.14	0.13	500
mushroom	0.23	0.16	0.19	500
oak_tree	0.48	0.68	0.56	500
orange	0.61	0.49	0.55	500
orchid	0.38	0.38	0.38	500
otter	0.07	0.06	0.06	500
palm_tree	0.48	0.45	0.46	500
pear	0.33	0.37	0.35	500
pickup_truck	0.40	0.35	0.37	500
pine_tree	0.28	0.24	0.26	500
plain	0.60	0.67	0.63	500
plate	0.47	0.54	0.50	500
poppy	0.35	0.47	0.40	500
porcupine	0.35	0.25	0.29	500
possum	0.13	0.19	0.15	500
rabbit	0.10	0.08	0.09	500
raccoon	0.22	0.15	0.18	500
ray	0.19	0.23	0.21	500
road	0.55	0.65	0.60	500
rocket	0.55	0.44	0.49	500
rose	0.39	0.37	0.38	500
sea	0.61	0.48	0.54	500
seal	0.10	0.11	0.11	500
shark	0.21	0.35	0.26	500
shrew	0.18	0.16	0.17	500
skunk	0.43	0.50	0.46	500
skyscraper	0.48	0.59	0.53	500
snail	0.11	0.21	0.14	500
snake	0.11	0.09	0.10	500
spider	0.30	0.32	0.31	500
squirrel	0.09	0.14	0.11	500
streetcar	0.47	0.22	0.30	500
sunflower	0.49	0.67	0.57	500
sweet_pepper	0.22	0.35	0.27	500
table	0.24	0.14	0.17	500
tank	0.42	0.35	0.38	500
telephone	0.21	0.26	0.23	500
television	0.39	0.38	0.39	500

```
        tiger       0.29      0.18      0.22       500
      tractor       0.33      0.30      0.32       500
        train       0.29      0.21      0.24       500
        trout       0.40      0.40      0.40       500
        tulip       0.22      0.16      0.18       500
       turtle       0.12      0.13      0.12       500
    wardrobe        0.70      0.69      0.69       500
        whale       0.29      0.38      0.33       500
  willow_tree       0.31      0.44      0.36       500
         wolf       0.31      0.18      0.23       500
        woman       0.15      0.18      0.16       500
         worm       0.19      0.25      0.21       500

     accuracy                           0.30     50000
    macro avg       0.31      0.30      0.30     50000
 weighted avg       0.31      0.30      0.30     50000
```

LeNet

[ ]:
```
y_pred = predict(lenet,ds_train)
y_test = [label for image,label in ds_train]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
              precision    recall  f1-score   support

       apple       0.00      0.00      0.00       500
aquarium_fish       0.00      0.00      0.00       500
        baby       0.00      0.00      0.00       500
        bear       0.00      0.00      0.00       500
      beaver       0.00      0.00      0.00       500
         bed       0.00      0.00      0.00       500
         bee       0.00      0.00      0.00       500
      beetle       0.00      0.00      0.00       500
     bicycle       0.00      0.00      0.00       500
      bottle       0.01      1.00      0.02       500
        bowl       0.00      0.00      0.00       500
         boy       0.00      0.00      0.00       500
      bridge       0.00      0.00      0.00       500
         bus       0.00      0.00      0.00       500
   butterfly       0.00      0.00      0.00       500
       camel       0.00      0.00      0.00       500
         can       0.00      0.00      0.00       500
      castle       0.00      0.00      0.00       500
```

caterpillar	0.00	0.00	0.00	500
cattle	0.00	0.00	0.00	500
chair	0.00	0.00	0.00	500
chimpanzee	0.00	0.00	0.00	500
clock	0.00	0.00	0.00	500
cloud	0.00	0.00	0.00	500
cockroach	0.00	0.00	0.00	500
couch	0.00	0.00	0.00	500
crab	0.00	0.00	0.00	500
crocodile	0.00	0.00	0.00	500
cup	0.00	0.00	0.00	500
dinosaur	0.00	0.00	0.00	500
dolphin	0.00	0.00	0.00	500
elephant	0.00	0.00	0.00	500
flatfish	0.00	0.00	0.00	500
forest	0.00	0.00	0.00	500
fox	0.00	0.00	0.00	500
girl	0.00	0.00	0.00	500
hamster	0.00	0.00	0.00	500
house	0.00	0.00	0.00	500
kangaroo	0.00	0.00	0.00	500
keyboard	0.00	0.00	0.00	500
lamp	0.00	0.00	0.00	500
lawn_mower	0.00	0.00	0.00	500
leopard	0.00	0.00	0.00	500
lion	0.00	0.00	0.00	500
lizard	0.00	0.00	0.00	500
lobster	0.00	0.00	0.00	500
man	0.00	0.00	0.00	500
maple_tree	0.00	0.00	0.00	500
motorcycle	0.00	0.00	0.00	500
mountain	0.00	0.00	0.00	500
mouse	0.00	0.00	0.00	500
mushroom	0.00	0.00	0.00	500
oak_tree	0.00	0.00	0.00	500
orange	0.00	0.00	0.00	500
orchid	0.00	0.00	0.00	500
otter	0.00	0.00	0.00	500
palm_tree	0.00	0.00	0.00	500
pear	0.00	0.00	0.00	500
pickup_truck	0.00	0.00	0.00	500
pine_tree	0.00	0.00	0.00	500
plain	0.00	0.00	0.00	500
plate	0.00	0.00	0.00	500
poppy	0.00	0.00	0.00	500
porcupine	0.00	0.00	0.00	500
possum	0.00	0.00	0.00	500
rabbit	0.00	0.00	0.00	500

```
      raccoon       0.00      0.00      0.00       500
          ray       0.00      0.00      0.00       500
         road       0.00      0.00      0.00       500
       rocket       0.00      0.00      0.00       500
         rose       0.00      0.00      0.00       500
          sea       0.00      0.00      0.00       500
         seal       0.00      0.00      0.00       500
        shark       0.00      0.00      0.00       500
        shrew       0.00      0.00      0.00       500
        skunk       0.00      0.00      0.00       500
   skyscraper       0.00      0.00      0.00       500
        snail       0.00      0.00      0.00       500
        snake       0.00      0.00      0.00       500
       spider       0.00      0.00      0.00       500
     squirrel       0.00      0.00      0.00       500
    streetcar       0.00      0.00      0.00       500
    sunflower       0.00      0.00      0.00       500
  sweet_pepper       0.00      0.00      0.00       500
        table       0.00      0.00      0.00       500
         tank       0.00      0.00      0.00       500
    telephone       0.00      0.00      0.00       500
   television       0.00      0.00      0.00       500
        tiger       0.00      0.00      0.00       500
      tractor       0.00      0.00      0.00       500
        train       0.00      0.00      0.00       500
        trout       0.00      0.00      0.00       500
        tulip       0.00      0.00      0.00       500
       turtle       0.00      0.00      0.00       500
     wardrobe       0.00      0.00      0.00       500
        whale       0.00      0.00      0.00       500
   willow_tree       0.00      0.00      0.00       500
         wolf       0.00      0.00      0.00       500
        woman       0.00      0.00      0.00       500
         worm       0.00      0.00      0.00       500

     accuracy                           0.01     50000
    macro avg       0.00      0.01      0.00     50000
 weighted avg       0.00      0.01      0.00     50000


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
```

```
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

AlexNet

[ ]:
```
y_pred = predict(alexnet,ds_train)
y_test = [label for image,label in ds_train]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.35      0.46      0.39       500
aquarium_fish       0.19      0.39      0.25       500
        baby       0.15      0.15      0.15       500
        bear       0.13      0.09      0.11       500
      beaver       0.00      0.00      0.00       500
         bed       0.12      0.07      0.09       500
         bee       0.10      0.15      0.12       500
      beetle       0.12      0.24      0.16       500
     bicycle       0.26      0.12      0.16       500
      bottle       0.25      0.11      0.15       500
        bowl       0.08      0.01      0.02       500
         boy       0.07      0.03      0.04       500
      bridge       0.19      0.23      0.21       500
         bus       0.30      0.20      0.24       500
   butterfly       0.12      0.08      0.10       500
       camel       0.12      0.06      0.08       500
         can       0.13      0.15      0.14       500
      castle       0.27      0.35      0.31       500
  caterpillar       0.10      0.14      0.11       500
      cattle       0.12      0.03      0.04       500
       chair       0.30      0.50      0.38       500
   chimpanzee       0.20      0.12      0.15       500
       clock       0.11      0.07      0.09       500
       cloud       0.28      0.55      0.37       500
   cockroach       0.26      0.44      0.32       500
       couch       0.17      0.08      0.10       500
        crab       0.04      0.00      0.01       500
   crocodile       0.12      0.08      0.09       500
         cup       0.11      0.30      0.16       500
    dinosaur       0.22      0.11      0.15       500
     dolphin       0.25      0.34      0.29       500
    elephant       0.16      0.07      0.09       500
```

flatfish	0.09	0.14	0.11	500
forest	0.33	0.11	0.16	500
fox	0.10	0.08	0.08	500
girl	0.07	0.10	0.08	500
hamster	0.13	0.28	0.17	500
house	0.15	0.07	0.10	500
kangaroo	0.06	0.03	0.04	500
keyboard	0.20	0.12	0.15	500
lamp	0.09	0.05	0.07	500
lawn_mower	0.28	0.46	0.35	500
leopard	0.07	0.04	0.05	500
lion	0.08	0.36	0.13	500
lizard	0.15	0.02	0.04	500
lobster	0.06	0.01	0.01	500
man	0.08	0.18	0.11	500
maple_tree	0.35	0.27	0.30	500
motorcycle	0.30	0.22	0.25	500
mountain	0.28	0.14	0.19	500
mouse	0.07	0.02	0.04	500
mushroom	0.12	0.14	0.13	500
oak_tree	0.50	0.50	0.50	500
orange	0.17	0.78	0.28	500
orchid	0.23	0.21	0.22	500
otter	0.04	0.00	0.01	500
palm_tree	0.25	0.25	0.25	500
pear	0.18	0.15	0.16	500
pickup_truck	0.30	0.12	0.18	500
pine_tree	0.37	0.06	0.11	500
plain	0.39	0.51	0.44	500
plate	0.16	0.20	0.18	500
poppy	0.22	0.58	0.32	500
porcupine	0.15	0.02	0.04	500
possum	0.09	0.00	0.01	500
rabbit	0.06	0.06	0.06	500
raccoon	0.16	0.09	0.11	500
ray	0.35	0.10	0.15	500
road	0.28	0.53	0.37	500
rocket	0.15	0.09	0.12	500
rose	0.24	0.24	0.24	500
sea	0.27	0.72	0.40	500
seal	0.07	0.01	0.02	500
shark	0.28	0.39	0.33	500
shrew	0.26	0.02	0.03	500
skunk	0.26	0.47	0.33	500
skyscraper	0.36	0.40	0.38	500
snail	0.07	0.07	0.07	500
snake	0.13	0.03	0.05	500
spider	0.05	0.00	0.00	500

```
          squirrel       0.08      0.01      0.02       500
        streetcar        0.19      0.14      0.16       500
        sunflower        0.26      0.60      0.36       500
      sweet_pepper       0.12      0.10      0.11       500
            table        0.09      0.02      0.03       500
             tank        0.17      0.15      0.16       500
        telephone        0.18      0.09      0.12       500
       television        0.22      0.20      0.21       500
            tiger        0.08      0.09      0.08       500
          tractor        0.18      0.31      0.23       500
            train        0.21      0.04      0.07       500
            trout        0.23      0.39      0.29       500
            tulip        0.07      0.06      0.06       500
           turtle        0.15      0.09      0.11       500
        wardrobe         0.26      0.69      0.37       500
            whale        0.22      0.28      0.25       500
       willow_tree       0.30      0.34      0.32       500
             wolf        0.10      0.18      0.13       500
            woman        0.10      0.03      0.05       500
             worm        0.26      0.05      0.08       500

         accuracy                             0.19     50000
        macro avg        0.18      0.19      0.16     50000
     weighted avg        0.18      0.19      0.16     50000
```

Sem Normalização

Resetando os Modelos

[30]:
```
cnn = torch.load('cnn0.pkl')
lenet = torch.load('lenet0.pkl')
alexnet = torch.load('alexnet0.pkl')
```

Dataset

[31]:
```
%cd data
!rm -rf *
%cd ..
!ls
```

```
/data
/
alexnet0.pkl  data      lenet0.pkl  mnt          run                 tmp
bin           datalab   lib         opt          sbin                tools
boot          dev       lib32       proc         srv                 usr
cnn0.pkl      etc       lib64       python-apt   sys                 var
content       home      media       root         tensorflow-1.15.2
```

[32]:
```
ds_train = CIFAR100('data/', train = True, download = True,transform =␣
↪transforms.ToTensor())
ds_test =  CIFAR100('data/', train = False, download = True, transform =␣
↪transforms.ToTensor())
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to
data/cifar-100-python.tar.gz
  0%|          | 0/169001437 [00:00<?, ?it/s]
Extracting data/cifar-100-python.tar.gz to data/
Files already downloaded and verified
```

[33]:
```
classes = ds_train.classes
```

[34]:
```
ds_train.data.shape
```

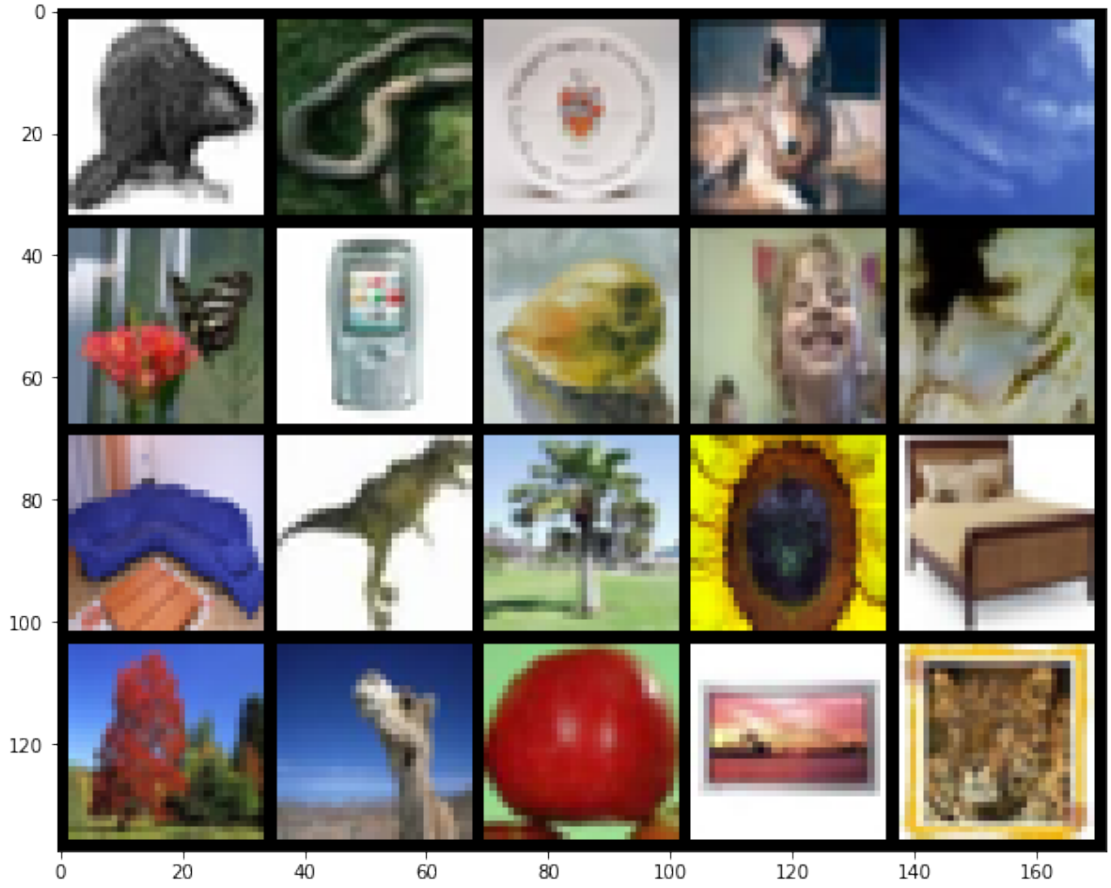
[34]:
```
(50000, 32, 32, 3)
```

Dataloaders

[35]:
```
batch_size = 40
dl_train = DataLoader(ds_train, batch_size = batch_size, shuffle = True)
dl_test =  DataLoader(ds_test, batch_size = batch_size, shuffle = True)
```

[36]:
```
batch = get_batch(dl_train)
plot_data(batch,classes)
```

```
<generator object plot_data.<locals>.<genexpr> at 0x7f377f8a07d0>
```



Treino

[37]:
```
num_epochs = 50
```

Nosso Modelo

[38]:
```
fit(cnn,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/50]: Loss : 4.179013405799866 Step: 1000
Epoch:  [2/50]: Loss : 3.438392668247223 Step: 1000
Epoch:  [3/50]: Loss : 2.996607216119766 Step: 1000
Epoch:  [4/50]: Loss : 2.7001130199432373 Step: 1000
Epoch:  [5/50]: Loss : 2.4788288805484773 Step: 1000
Epoch:  [6/50]: Loss : 2.306931765437126 Step: 1000
Epoch:  [7/50]: Loss : 2.178256085753441 Step: 1000
Epoch:  [8/50]: Loss : 2.0532144074440004 Step: 1000
Epoch:  [9/50]: Loss : 1.9390509141683578 Step: 1000
Epoch:  [10/50]: Loss : 1.8310176330804824 Step: 1000
Epoch:  [11/50]: Loss : 1.7315656394958496 Step: 1000
Epoch:  [12/50]: Loss : 1.6370024831295014 Step: 1000
```

```
Epoch:  [13/50]: Loss : 1.5436512974500656 Step: 1000
Epoch:  [14/50]: Loss : 1.4514065512418748 Step: 1000
Epoch:  [15/50]: Loss : 1.371270197749138 Step: 1000
Epoch:  [16/50]: Loss : 1.2786473959088325 Step: 1000
Epoch:  [17/50]: Loss : 1.1960725781321526 Step: 1000
Epoch:  [18/50]: Loss : 1.111581428706646 Step: 1000
Epoch:  [19/50]: Loss : 1.0397278941273689 Step: 1000
Epoch:  [20/50]: Loss : 0.9595479361116886 Step: 1000
Epoch:  [21/50]: Loss : 0.8854659529030323 Step: 1000
Epoch:  [22/50]: Loss : 0.8118978538066148 Step: 1000
Epoch:  [23/50]: Loss : 0.7556001746058464 Step: 1000
Epoch:  [24/50]: Loss : 0.6902042623013258 Step: 1000
Epoch:  [25/50]: Loss : 0.6211089231967926 Step: 1000
Epoch:  [26/50]: Loss : 0.5654940499961376 Step: 1000
Epoch:  [27/50]: Loss : 0.5173367411494255 Step: 1000
Epoch:  [28/50]: Loss : 0.46565264028310777 Step: 1000
Epoch:  [29/50]: Loss : 0.42319203965365887 Step: 1000
Epoch:  [30/50]: Loss : 0.38160582542419436 Step: 1000
Epoch:  [31/50]: Loss : 0.35624568958953023 Step: 1000
Epoch:  [32/50]: Loss : 0.3181510470137 Step: 1000
Epoch:  [33/50]: Loss : 0.2899043241515756 Step: 1000
Epoch:  [34/50]: Loss : 0.2681153809502721 Step: 1000
Epoch:  [35/50]: Loss : 0.2410107906125486 Step: 1000
Epoch:  [36/50]: Loss : 0.22955693044140935 Step: 1000
Epoch:  [37/50]: Loss : 0.21519428296759724 Step: 1000
Epoch:  [38/50]: Loss : 0.19838316901400685 Step: 1000
Epoch:  [39/50]: Loss : 0.18326347655802966 Step: 1000
Epoch:  [40/50]: Loss : 0.16742151408083736 Step: 1000
Epoch:  [41/50]: Loss : 0.15615761752426624 Step: 1000
Epoch:  [42/50]: Loss : 0.14360196989774704 Step: 1000
Epoch:  [43/50]: Loss : 0.13492356652766466 Step: 1000
Epoch:  [44/50]: Loss : 0.12666631453670563 Step: 1000
Epoch:  [45/50]: Loss : 0.11810735175386071 Step: 1000
Epoch:  [46/50]: Loss : 0.1104555981317535 Step: 1000
Epoch:  [47/50]: Loss : 0.10186342680361121 Step: 1000
Epoch:  [48/50]: Loss : 0.108775508842431 Step: 1000
Epoch:  [49/50]: Loss : 0.09753606062289327 Step: 1000
Epoch:  [50/50]: Loss : 0.09404279227368534 Step: 1000
The Training is Finished
```

[39]:
```
torch.save(cnn,'cnn_nn.pkl')
```

LeNet

[40]:
```
fit(lenet,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
```

```
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")

Epoch:  [1/50]: Loss : 4.61542963886261 Step: 1000
Epoch:  [2/50]: Loss : 4.607282531261444 Step: 1000
Epoch:  [3/50]: Loss : 4.6070120644569394 Step: 1000
Epoch:  [4/50]: Loss : 4.6071216058731075 Step: 1000
Epoch:  [5/50]: Loss : 4.607118339061737 Step: 1000
Epoch:  [6/50]: Loss : 4.607046228408813 Step: 1000
Epoch:  [7/50]: Loss : 4.607022094249725 Step: 1000
Epoch:  [8/50]: Loss : 4.6070923047065735 Step: 1000
Epoch:  [9/50]: Loss : 4.607124197006225 Step: 1000
Epoch:  [10/50]: Loss : 4.607030871391296 Step: 1000
Epoch:  [11/50]: Loss : 4.606976990222931 Step: 1000
Epoch:  [12/50]: Loss : 4.60688343334198 Step: 1000
Epoch:  [13/50]: Loss : 4.606955656051635 Step: 1000
Epoch:  [14/50]: Loss : 4.607049481868744 Step: 1000
Epoch:  [15/50]: Loss : 4.606971336364746 Step: 1000
Epoch:  [16/50]: Loss : 4.60686450433731 Step: 1000
Epoch:  [17/50]: Loss : 4.606996448993683 Step: 1000
Epoch:  [18/50]: Loss : 4.607018396377564 Step: 1000
Epoch:  [19/50]: Loss : 4.606984129428864 Step: 1000
Epoch:  [20/50]: Loss : 4.607054743766785 Step: 1000
Epoch:  [21/50]: Loss : 4.606983383655548 Step: 1000
Epoch:  [22/50]: Loss : 4.606953495025635 Step: 1000
Epoch:  [23/50]: Loss : 4.606945690631867 Step: 1000
Epoch:  [24/50]: Loss : 4.60699021577835 Step: 1000
Epoch:  [25/50]: Loss : 4.606950966835022 Step: 1000
Epoch:  [26/50]: Loss : 4.606820618629455 Step: 1000
Epoch:  [27/50]: Loss : 4.60697933959961 Step: 1000
Epoch:  [28/50]: Loss : 4.606869065284729 Step: 1000
Epoch:  [29/50]: Loss : 4.6067976222038265 Step: 1000
Epoch:  [30/50]: Loss : 4.606891972541809 Step: 1000
Epoch:  [31/50]: Loss : 4.606839633464813 Step: 1000
Epoch:  [32/50]: Loss : 4.606784490585327 Step: 1000
Epoch:  [33/50]: Loss : 4.6068675451278684 Step: 1000
Epoch:  [34/50]: Loss : 4.6067769589424135 Step: 1000
Epoch:  [35/50]: Loss : 4.60685941696167 Step: 1000
Epoch:  [36/50]: Loss : 4.606758150100708 Step: 1000
Epoch:  [37/50]: Loss : 4.606709567546845 Step: 1000
Epoch:  [38/50]: Loss : 4.6067784543037416 Step: 1000
Epoch:  [39/50]: Loss : 4.6068148827552795 Step: 1000
Epoch:  [40/50]: Loss : 4.606748072624207 Step: 1000
Epoch:  [41/50]: Loss : 4.606836824893952 Step: 1000
Epoch:  [42/50]: Loss : 4.606840590953827 Step: 1000
Epoch:  [43/50]: Loss : 4.606822072982788 Step: 1000
Epoch:  [44/50]: Loss : 4.606746115684509 Step: 1000
Epoch:  [45/50]: Loss : 4.606810522556305 Step: 1000
```

```
Epoch:  [46/50]: Loss : 4.606718451499939 Step: 1000
Epoch:  [47/50]: Loss : 4.606667377948761 Step: 1000
Epoch:  [48/50]: Loss : 4.60664480304718 Step: 1000
Epoch:  [49/50]: Loss : 4.606763448238373 Step: 1000
Epoch:  [50/50]: Loss : 4.60672319316864 Step: 1000
The Training is Finished
```

[41]:
```
torch.save(lenet,'lenet_nn.pkl')
```

AlexNet

[42]:
```
fit(alexnet,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/50]: Loss : 4.605416027069092 Step: 1000
Epoch:  [2/50]: Loss : 4.605262865066528 Step: 1000
Epoch:  [3/50]: Loss : 4.605209973812103 Step: 1000
Epoch:  [4/50]: Loss : 4.605112452983856 Step: 1000
Epoch:  [5/50]: Loss : 4.604927471637726 Step: 1000
Epoch:  [6/50]: Loss : 4.60448598241806 Step: 1000
Epoch:  [7/50]: Loss : 4.603274716377259 Step: 1000
Epoch:  [8/50]: Loss : 4.544906435012817 Step: 1000
Epoch:  [9/50]: Loss : 4.351325972557068 Step: 1000
Epoch:  [10/50]: Loss : 4.2912385396957395 Step: 1000
Epoch:  [11/50]: Loss : 4.211585951328278 Step: 1000
Epoch:  [12/50]: Loss : 4.129200117349624 Step: 1000
Epoch:  [13/50]: Loss : 4.069087975978851 Step: 1000
Epoch:  [14/50]: Loss : 3.9927594034671783 Step: 1000
Epoch:  [15/50]: Loss : 3.9072845816612243 Step: 1000
Epoch:  [16/50]: Loss : 3.8002532489299776 Step: 1000
Epoch:  [17/50]: Loss : 3.6923680322170256 Step: 1000
Epoch:  [18/50]: Loss : 3.5993553168773653 Step: 1000
Epoch:  [19/50]: Loss : 3.5146681532859803 Step: 1000
Epoch:  [20/50]: Loss : 3.435331071138382 Step: 1000
Epoch:  [21/50]: Loss : 3.3549393808841703 Step: 1000
Epoch:  [22/50]: Loss : 3.2761444272994997 Step: 1000
Epoch:  [23/50]: Loss : 3.2082489037513735 Step: 1000
Epoch:  [24/50]: Loss : 3.1446655459403994 Step: 1000
Epoch:  [25/50]: Loss : 3.085576654672623 Step: 1000
Epoch:  [26/50]: Loss : 3.022324504852295 Step: 1000
Epoch:  [27/50]: Loss : 2.9549592499732973 Step: 1000
Epoch:  [28/50]: Loss : 2.897660544157028 Step: 1000
Epoch:  [29/50]: Loss : 2.8393803108930586 Step: 1000
Epoch:  [30/50]: Loss : 2.7765077970027923 Step: 1000
Epoch:  [31/50]: Loss : 2.7041289836168287 Step: 1000
Epoch:  [32/50]: Loss : 2.6491429270505904 Step: 1000
Epoch:  [33/50]: Loss : 2.591402633070946 Step: 1000
Epoch:  [34/50]: Loss : 2.529949787378311 Step: 1000
Epoch:  [35/50]: Loss : 2.476334375023842 Step: 1000
```

```
Epoch:  [36/50]: Loss : 2.4210771373510362 Step: 1000
Epoch:  [37/50]: Loss : 2.358743262052536 Step: 1000
Epoch:  [38/50]: Loss : 2.3132674360275267 Step: 1000
Epoch:  [39/50]: Loss : 2.253588455557823 Step: 1000
Epoch:  [40/50]: Loss : 2.195942039132118 Step: 1000
Epoch:  [41/50]: Loss : 2.140341108083725 Step: 1000
Epoch:  [42/50]: Loss : 2.0803150091171263 Step: 1000
Epoch:  [43/50]: Loss : 2.0426924891471865 Step: 1000
Epoch:  [44/50]: Loss : 1.9817742170095445 Step: 1000
Epoch:  [45/50]: Loss : 1.9413048338890075 Step: 1000
Epoch:  [46/50]: Loss : 1.868305791079998 Step: 1000
Epoch:  [47/50]: Loss : 1.8351181132793426 Step: 1000
Epoch:  [48/50]: Loss : 1.7805578573942185 Step: 1000
Epoch:  [49/50]: Loss : 1.7216581008434295 Step: 1000
Epoch:  [50/50]: Loss : 1.6670366756916046 Step: 1000
The Training is Finished
```

[43]:
```
torch.save(alexnet,'alexnet_nn.pkl')
```

Salvando os Modelos

[44]:
```
torch.save(cnn,'cnn_nn.pkl')
torch.save(lenet,'lenet_nn.pkl')
torch.save(alexnet,'alexnet_nn.pkl')

#export_to_onnx(model,num_epochs = 1,steps = 500)
```

Test Carregando os Modelos

[45]:
```
cnn = torch.load('cnn_nn.pkl')
lenet = torch.load('lenet_nn.pkl')
alexnet = torch.load('alexnet_nn.pkl')
```

Nosso Modelo

[46]:
```
test(cnn,dl_test)
```

```
Accuracy =  4976 / 10000 49.76 %
```

LeNet

[47]:
```
test(lenet,dl_test)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

```
Accuracy =  67 / 10000 0.67 %
```

AlexNet

[48]:
```
test(alexnet,dl_test)
```

```
Accuracy =  4210 / 10000 42.1 %
```

Métricas de Desempenho Nosso Modelo

[49]:
```
y_pred = predict(cnn,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.75      0.71      0.73       100
aquarium_fish       0.54      0.62      0.58       100
        baby       0.32      0.39      0.35       100
        bear       0.44      0.25      0.32       100
      beaver       0.38      0.32      0.35       100
         bed       0.43      0.47      0.45       100
         bee       0.63      0.47      0.54       100
      beetle       0.59      0.45      0.51       100
     bicycle       0.65      0.60      0.62       100
      bottle       0.62      0.64      0.63       100
        bowl       0.50      0.28      0.36       100
         boy       0.32      0.43      0.37       100
      bridge       0.50      0.60      0.54       100
         bus       0.43      0.35      0.39       100
   butterfly       0.53      0.38      0.44       100
       camel       0.42      0.39      0.40       100
         can       0.59      0.49      0.54       100
      castle       0.56      0.71      0.63       100
  caterpillar       0.41      0.38      0.40       100
      cattle       0.42      0.45      0.44       100
       chair       0.68      0.82      0.75       100
   chimpanzee       0.64      0.58      0.61       100
       clock       0.41      0.50      0.45       100
       cloud       0.81      0.50      0.62       100
   cockroach       0.73      0.61      0.66       100
       couch       0.37      0.43      0.40       100
        crab       0.41      0.45      0.43       100
   crocodile       0.41      0.26      0.32       100
         cup       0.64      0.76      0.69       100
    dinosaur       0.45      0.47      0.46       100
     dolphin       0.45      0.54      0.49       100
    elephant       0.52      0.38      0.44       100
    flatfish       0.46      0.37      0.41       100
      forest       0.41      0.43      0.42       100
```

fox	0.48	0.45	0.46	100
girl	0.30	0.22	0.25	100
hamster	0.63	0.47	0.54	100
house	0.49	0.36	0.42	100
kangaroo	0.31	0.28	0.30	100
keyboard	0.59	0.59	0.59	100
lamp	0.44	0.46	0.45	100
lawn_mower	0.81	0.64	0.72	100
leopard	0.34	0.71	0.46	100
lion	0.57	0.58	0.57	100
lizard	0.21	0.24	0.22	100
lobster	0.35	0.29	0.32	100
man	0.24	0.32	0.28	100
maple_tree	0.52	0.69	0.59	100
motorcycle	0.72	0.83	0.77	100
mountain	0.67	0.59	0.63	100
mouse	0.32	0.27	0.29	100
mushroom	0.42	0.52	0.47	100
oak_tree	0.60	0.50	0.55	100
orange	0.68	0.78	0.73	100
orchid	0.64	0.56	0.60	100
otter	0.24	0.20	0.22	100
palm_tree	0.73	0.69	0.71	100
pear	0.64	0.49	0.56	100
pickup_truck	0.59	0.67	0.63	100
pine_tree	0.51	0.49	0.50	100
plain	0.88	0.76	0.82	100
plate	0.67	0.62	0.64	100
poppy	0.51	0.46	0.48	100
porcupine	0.49	0.50	0.49	100
possum	0.31	0.28	0.29	100
rabbit	0.25	0.29	0.27	100
raccoon	0.51	0.45	0.48	100
ray	0.47	0.36	0.41	100
road	0.77	0.75	0.76	100
rocket	0.75	0.61	0.67	100
rose	0.43	0.69	0.53	100
sea	0.62	0.75	0.68	100
seal	0.25	0.24	0.25	100
shark	0.46	0.53	0.50	100
shrew	0.30	0.36	0.33	100
skunk	0.66	0.79	0.72	100
skyscraper	0.62	0.71	0.66	100
snail	0.42	0.34	0.38	100
snake	0.28	0.26	0.27	100
spider	0.52	0.53	0.53	100
squirrel	0.23	0.28	0.25	100
streetcar	0.48	0.54	0.51	100

```
    sunflower       0.84      0.71      0.77       100
 sweet_pepper       0.38      0.45      0.41       100
        table       0.46      0.46      0.46       100
         tank       0.67      0.62      0.65       100
    telephone       0.45      0.58      0.50       100
   television       0.46      0.56      0.50       100
        tiger       0.66      0.49      0.56       100
      tractor       0.63      0.41      0.50       100
        train       0.53      0.55      0.54       100
        trout       0.65      0.61      0.63       100
        tulip       0.38      0.47      0.42       100
       turtle       0.45      0.27      0.34       100
     wardrobe       0.76      0.79      0.77       100
        whale       0.45      0.63      0.52       100
  willow_tree       0.54      0.38      0.44       100
         wolf       0.45      0.53      0.49       100
        woman       0.18      0.19      0.18       100
         worm       0.43      0.59      0.50       100

     accuracy                           0.50     10000
    macro avg       0.51      0.50      0.50     10000
 weighted avg       0.51      0.50      0.50     10000
```

LeNet

[50]:
```
y_pred = predict(lenet,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1806: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
              precision    recall  f1-score   support

       apple       0.00      0.00      0.00       100
aquarium_fish       0.00      0.11      0.01       100
        baby       0.00      0.00      0.00       100
        bear       0.00      0.00      0.00       100
      beaver       0.00      0.00      0.00       100
         bed       0.00      0.00      0.00       100
         bee       0.00      0.00      0.00       100
      beetle       0.00      0.00      0.00       100
     bicycle       0.00      0.00      0.00       100
      bottle       0.00      0.00      0.00       100
        bowl       0.00      0.00      0.00       100
         boy       0.00      0.00      0.00       100
```

bridge	0.00	0.00	0.00	100
bus	0.00	0.00	0.00	100
butterfly	0.00	0.00	0.00	100
camel	0.00	0.00	0.00	100
can	0.00	0.00	0.00	100
castle	0.00	0.00	0.00	100
caterpillar	0.00	0.00	0.00	100
cattle	0.00	0.00	0.00	100
chair	0.00	0.00	0.00	100
chimpanzee	0.00	0.00	0.00	100
clock	0.00	0.00	0.00	100
cloud	0.00	0.00	0.00	100
cockroach	0.00	0.00	0.00	100
couch	0.00	0.00	0.00	100
crab	0.00	0.00	0.00	100
crocodile	0.00	0.00	0.00	100
cup	0.00	0.00	0.00	100
dinosaur	0.00	0.00	0.00	100
dolphin	0.00	0.00	0.00	100
elephant	0.00	0.00	0.00	100
flatfish	0.00	0.00	0.00	100
forest	0.00	0.00	0.00	100
fox	0.00	0.00	0.00	100
girl	0.00	0.00	0.00	100
hamster	0.00	0.00	0.00	100
house	0.00	0.00	0.00	100
kangaroo	0.00	0.00	0.00	100
keyboard	0.01	0.56	0.02	100
lamp	0.00	0.00	0.00	100
lawn_mower	0.00	0.00	0.00	100
leopard	0.00	0.00	0.00	100
lion	0.00	0.00	0.00	100
lizard	0.00	0.00	0.00	100
lobster	0.00	0.00	0.00	100
man	0.00	0.00	0.00	100
maple_tree	0.00	0.00	0.00	100
motorcycle	0.00	0.00	0.00	100
mountain	0.00	0.00	0.00	100
mouse	0.00	0.00	0.00	100
mushroom	0.00	0.00	0.00	100
oak_tree	0.00	0.00	0.00	100
orange	0.00	0.00	0.00	100
orchid	0.00	0.00	0.00	100
otter	0.00	0.00	0.00	100
palm_tree	0.00	0.00	0.00	100
pear	0.00	0.00	0.00	100
pickup_truck	0.00	0.00	0.00	100
pine_tree	0.00	0.00	0.00	100

```
       plain       0.00      0.00      0.00       100
       plate       0.00      0.00      0.00       100
       poppy       0.00      0.00      0.00       100
    porcupine       0.00      0.00      0.00       100
      possum       0.00      0.00      0.00       100
      rabbit       0.00      0.00      0.00       100
     raccoon       0.00      0.00      0.00       100
         ray       0.00      0.00      0.00       100
        road       0.00      0.00      0.00       100
      rocket       0.00      0.00      0.00       100
        rose       0.00      0.00      0.00       100
         sea       0.00      0.00      0.00       100
        seal       0.00      0.00      0.00       100
       shark       0.00      0.00      0.00       100
       shrew       0.00      0.00      0.00       100
       skunk       0.00      0.00      0.00       100
  skyscraper       0.00      0.00      0.00       100
       snail       0.00      0.00      0.00       100
       snake       0.00      0.00      0.00       100
      spider       0.00      0.00      0.00       100
    squirrel       0.00      0.00      0.00       100
   streetcar       0.00      0.00      0.00       100
   sunflower       0.00      0.00      0.00       100
 sweet_pepper       0.00      0.00      0.00       100
       table       0.00      0.00      0.00       100
        tank       0.00      0.00      0.00       100
   telephone       0.00      0.00      0.00       100
  television       0.00      0.00      0.00       100
       tiger       0.00      0.00      0.00       100
     tractor       0.00      0.00      0.00       100
       train       0.00      0.00      0.00       100
       trout       0.00      0.00      0.00       100
       tulip       0.00      0.00      0.00       100
      turtle       0.00      0.00      0.00       100
    wardrobe       0.00      0.00      0.00       100
       whale       0.00      0.00      0.00       100
  willow_tree       0.00      0.00      0.00       100
        wolf       0.00      0.00      0.00       100
       woman       0.00      0.00      0.00       100
        worm       0.00      0.00      0.00       100

    accuracy                           0.01     10000
   macro avg       0.00      0.01      0.00     10000
weighted avg       0.00      0.01      0.00     10000

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
```

```
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

AlexNet

[51]:
```
y_pred = predict(alexnet,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.66      0.65      0.65       100
aquarium_fish       0.50      0.50      0.50       100
        baby       0.31      0.38      0.34       100
        bear       0.24      0.23      0.23       100
      beaver       0.23      0.25      0.24       100
         bed       0.26      0.47      0.34       100
         bee       0.41      0.44      0.42       100
      beetle       0.61      0.37      0.46       100
     bicycle       0.39      0.42      0.41       100
      bottle       0.63      0.44      0.52       100
        bowl       0.31      0.16      0.21       100
         boy       0.50      0.20      0.29       100
      bridge       0.42      0.44      0.43       100
         bus       0.27      0.50      0.35       100
   butterfly       0.37      0.29      0.32       100
       camel       0.35      0.30      0.32       100
         can       0.60      0.29      0.39       100
      castle       0.45      0.69      0.54       100
  caterpillar       0.41      0.39      0.40       100
      cattle       0.37      0.41      0.39       100
       chair       0.74      0.75      0.74       100
   chimpanzee       0.60      0.58      0.59       100
       clock       0.24      0.41      0.30       100
       cloud       0.48      0.68      0.56       100
   cockroach       0.53      0.72      0.61       100
       couch       0.33      0.17      0.22       100
```

crab	0.46	0.21	0.29	100
crocodile	0.27	0.38	0.31	100
cup	0.55	0.49	0.52	100
dinosaur	0.52	0.40	0.45	100
dolphin	0.42	0.42	0.42	100
elephant	0.35	0.37	0.36	100
flatfish	0.45	0.35	0.40	100
forest	0.51	0.36	0.42	100
fox	0.49	0.43	0.46	100
girl	0.33	0.19	0.24	100
hamster	0.45	0.36	0.40	100
house	0.24	0.35	0.28	100
kangaroo	0.20	0.21	0.20	100
keyboard	0.70	0.57	0.63	100
lamp	0.39	0.29	0.33	100
lawn_mower	0.65	0.61	0.63	100
leopard	0.47	0.20	0.28	100
lion	0.45	0.57	0.50	100
lizard	0.27	0.19	0.22	100
lobster	0.37	0.22	0.27	100
man	0.36	0.29	0.32	100
maple_tree	0.52	0.49	0.50	100
motorcycle	0.48	0.64	0.55	100
mountain	0.60	0.33	0.43	100
mouse	0.27	0.21	0.24	100
mushroom	0.39	0.38	0.38	100
oak_tree	0.52	0.63	0.57	100
orange	0.50	0.82	0.62	100
orchid	0.47	0.61	0.53	100
otter	0.17	0.14	0.15	100
palm_tree	0.56	0.59	0.58	100
pear	0.38	0.46	0.42	100
pickup_truck	0.48	0.47	0.48	100
pine_tree	0.38	0.31	0.34	100
plain	0.74	0.78	0.76	100
plate	0.45	0.37	0.41	100
poppy	0.59	0.54	0.56	100
porcupine	0.43	0.35	0.39	100
possum	0.22	0.26	0.24	100
rabbit	0.34	0.22	0.27	100
raccoon	0.32	0.39	0.35	100
ray	0.50	0.35	0.41	100
road	0.74	0.68	0.71	100
rocket	0.43	0.65	0.52	100
rose	0.49	0.50	0.49	100
sea	0.60	0.56	0.58	100
seal	0.20	0.17	0.18	100
shark	0.42	0.41	0.42	100

```
           shrew       0.34      0.17      0.23       100
            skunk       0.66      0.61      0.64       100
      skyscraper       0.60      0.65      0.62       100
           snail       0.28      0.29      0.28       100
           snake       0.30      0.22      0.25       100
          spider       0.39      0.29      0.33       100
        squirrel       0.17      0.22      0.19       100
       streetcar       0.32      0.49      0.39       100
       sunflower       0.64      0.82      0.72       100
    sweet_pepper       0.36      0.50      0.42       100
           table       0.28      0.38      0.32       100
            tank       0.48      0.66      0.56       100
       telephone       0.54      0.49      0.51       100
      television       0.50      0.45      0.47       100
           tiger       0.36      0.37      0.36       100
         tractor       0.32      0.55      0.40       100
           train       0.40      0.40      0.40       100
           trout       0.50      0.60      0.55       100
           tulip       0.32      0.27      0.29       100
          turtle       0.42      0.15      0.22       100
       wardrobe       0.67      0.82      0.74       100
           whale       0.48      0.47      0.47       100
      willow_tree       0.37      0.29      0.32       100
            wolf       0.26      0.47      0.34       100
           woman       0.23      0.22      0.23       100
            worm       0.58      0.35      0.44       100

        accuracy                           0.42     10000
       macro avg       0.43      0.42      0.41     10000
    weighted avg       0.43      0.42      0.41     10000
```

LeNet Com ReLu

Dataset

[52]:
```
transform_train = transforms.Compose([#transforms.Resize((227,227)),
                                       transforms.RandomHorizontalFlip(p=0.7),
                                       transforms.ToTensor(),
                                       transforms.Normalize(mean=[0.5071, 0.4867,␣
 ↪0.4408], std=[0.2675, 0.2565, 0.2761])
                                       ])
transform_test = transforms.Compose([#transforms.Resize((227,227)),
                                     transforms.ToTensor(),
                                     transforms.Normalize(mean=[0.5071, 0.4867,␣
 ↪0.4408], std=[0.2675, 0.2565, 0.2761])
                                     ])
```

[53]:
```
%cd data
!rm -rf *
%cd ..
!ls
```

```
/data
/
alexnet0.pkl    content  lenet0.pkl    mnt          sbin                usr
alexnet_nn.pkl  data     lenet_nn.pkl  opt          srv                 var
bin             datalab  lib           proc         sys
boot            dev      lib32         python-apt   tensorflow-1.15.2
cnn0.pkl        etc      lib64         root         tmp
cnn_nn.pkl      home     media         run          tools
```

[54]:
```
ds_train = CIFAR100('data/', train = True, download = True,transform =␣
 ↪transform_train)
ds_test =  CIFAR100('data/', train = False, download = True, transform =␣
 ↪transform_test)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to
data/cifar-100-python.tar.gz

  0%|          | 0/169001437 [00:00<?, ?it/s]

Extracting data/cifar-100-python.tar.gz to data/
Files already downloaded and verified
```

[55]:
```
classes = ds_train.classes
```

[56]:
```
ds_train.data.shape
```

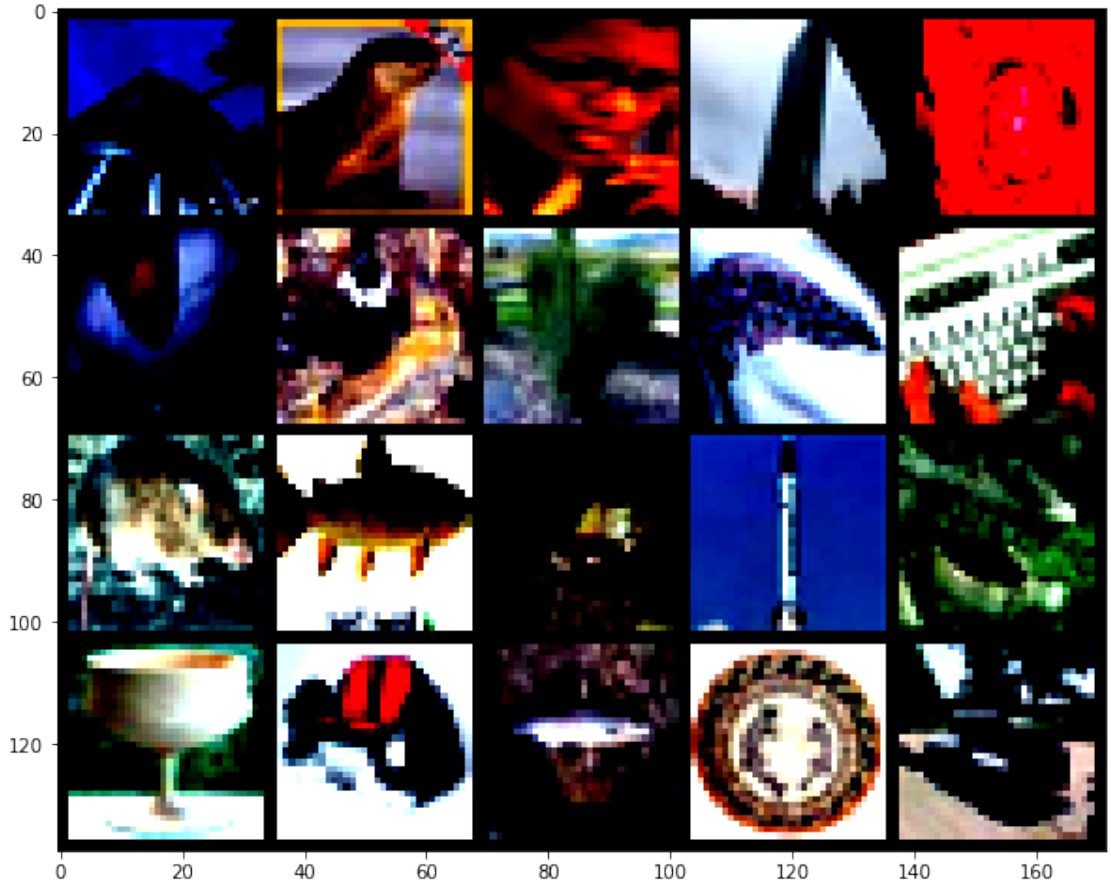
[56]:
```
(50000, 32, 32, 3)
```

Dataloaders

[57]:
```
batch_size = 40
dl_train = DataLoader(ds_train, batch_size = batch_size, shuffle = True)
dl_test =  DataLoader(ds_test, batch_size = batch_size, shuffle = True)
```

[58]:
```
batch = get_batch(dl_train)
plot_data(batch,classes)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).

<generator object plot_data.<locals>.<genexpr> at 0x7f377b056ad0>
```



Modelo

[59]:
```
class LeNet(nn.Module):
  def __init__(self):
    super(LeNet,self).__init__()
    self.conv1 = nn.Conv2d(3, 6, kernel_size=5, padding=2)
    self.avgPool = nn.AvgPool2d(kernel_size=2, stride=2)
    self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
    self.flatten = nn.Flatten()
    self.fc1 = nn.Linear(576, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 100)

  def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.avgPool(x)
    x = F.relu(self.conv2(x))
    x = self.avgPool(x)
    x = self.flatten(x)
```

```
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    #print(x.shape)
    return x
```

[60]:
```
lenet_relu = LeNet()
lenet_relu = lenet_relu.to(get_device())
```

Treino

[61]:
```
num_epochs = 50
```

[62]:
```
fit(lenet_relu,dl_train,num_epochs = num_epochs,steps = 1000)
```

```
Epoch:  [1/50]: Loss : 4.606516783237457 Step: 1000
Epoch:  [2/50]: Loss : 4.5618973054885865 Step: 1000
Epoch:  [3/50]: Loss : 4.180815997838974 Step: 1000
Epoch:  [4/50]: Loss : 3.866441859960556 Step: 1000
Epoch:  [5/50]: Loss : 3.7234234161376953 Step: 1000
Epoch:  [6/50]: Loss : 3.6055504398345946 Step: 1000
Epoch:  [7/50]: Loss : 3.5088953626155854 Step: 1000
Epoch:  [8/50]: Loss : 3.4231711633205415 Step: 1000
Epoch:  [9/50]: Loss : 3.3417946221828463 Step: 1000
Epoch:  [10/50]: Loss : 3.2589597444534304 Step: 1000
Epoch:  [11/50]: Loss : 3.194863090753555 Step: 1000
Epoch:  [12/50]: Loss : 3.134349441051483 Step: 1000
Epoch:  [13/50]: Loss : 3.064538962364197 Step: 1000
Epoch:  [14/50]: Loss : 3.0117904968261717 Step: 1000
Epoch:  [15/50]: Loss : 2.958566612958908 Step: 1000
Epoch:  [16/50]: Loss : 2.904628486633301 Step: 1000
Epoch:  [17/50]: Loss : 2.863016271352768 Step: 1000
Epoch:  [18/50]: Loss : 2.8129176807403566 Step: 1000
Epoch:  [19/50]: Loss : 2.767835429191589 Step: 1000
Epoch:  [20/50]: Loss : 2.731439539194107 Step: 1000
Epoch:  [21/50]: Loss : 2.697309470772743 Step: 1000
Epoch:  [22/50]: Loss : 2.66292913544178 Step: 1000
Epoch:  [23/50]: Loss : 2.6340649600028994 Step: 1000
Epoch:  [24/50]: Loss : 2.5861403868198396 Step: 1000
Epoch:  [25/50]: Loss : 2.5629611630439757 Step: 1000
Epoch:  [26/50]: Loss : 2.5305612366199495 Step: 1000
Epoch:  [27/50]: Loss : 2.5003522374629976 Step: 1000
Epoch:  [28/50]: Loss : 2.46563694357872 Step: 1000
Epoch:  [29/50]: Loss : 2.4448762459754945 Step: 1000
Epoch:  [30/50]: Loss : 2.4131632412672044 Step: 1000
Epoch:  [31/50]: Loss : 2.401885582923889 Step: 1000
Epoch:  [32/50]: Loss : 2.3762717964649203 Step: 1000
Epoch:  [33/50]: Loss : 2.3579055578708648 Step: 1000
```

```
Epoch:  [34/50]: Loss : 2.340741939306259 Step: 1000
Epoch:  [35/50]: Loss : 2.3156644077301025 Step: 1000
Epoch:  [36/50]: Loss : 2.295909655928612 Step: 1000
Epoch:  [37/50]: Loss : 2.279310473918915 Step: 1000
Epoch:  [38/50]: Loss : 2.2588508118391037 Step: 1000
Epoch:  [39/50]: Loss : 2.241517085790634 Step: 1000
Epoch:  [40/50]: Loss : 2.220813388824463 Step: 1000
Epoch:  [41/50]: Loss : 2.202756679773331 Step: 1000
Epoch:  [42/50]: Loss : 2.181867129445076 Step: 1000
Epoch:  [43/50]: Loss : 2.1814608387947083 Step: 1000
Epoch:  [44/50]: Loss : 2.167248051524162 Step: 1000
Epoch:  [45/50]: Loss : 2.147615460038185 Step: 1000
Epoch:  [46/50]: Loss : 2.1263825623989105 Step: 1000
Epoch:  [47/50]: Loss : 2.114527777552605 Step: 1000
Epoch:  [48/50]: Loss : 2.1056746519804 Step: 1000
Epoch:  [49/50]: Loss : 2.091613826751709 Step: 1000
Epoch:  [50/50]: Loss : 2.0808096026182175 Step: 1000
The Training is Finished
```

[65]:
```
torch.save(lenet_relu,'lenet_relu.pkl')
```

Teste

[66]:
```
lenet_relu = torch.load('lenet_relu.pkl')
```

[67]:
```
test(lenet_relu,dl_test)
```

```
Accuracy =  3375 / 10000 33.75 %
```

Métricas

[68]:
```
y_pred = predict(lenet_relu,ds_test)
y_test = [label for image,label in ds_test]
print(classification_report(y_test,y_pred,target_names = classes))
```

```
              precision    recall  f1-score   support

       apple       0.50      0.59      0.54       100
aquarium_fish       0.53      0.40      0.45       100
        baby       0.24      0.11      0.15       100
        bear       0.20      0.18      0.19       100
      beaver       0.18      0.19      0.19       100
         bed       0.38      0.22      0.28       100
         bee       0.39      0.35      0.37       100
      beetle       0.39      0.32      0.35       100
     bicycle       0.31      0.51      0.39       100
      bottle       0.54      0.39      0.45       100
        bowl       0.19      0.28      0.22       100
         boy       0.22      0.15      0.18       100
```

bridge	0.26	0.35	0.30	100
bus	0.28	0.39	0.32	100
butterfly	0.30	0.28	0.29	100
camel	0.27	0.15	0.19	100
can	0.35	0.28	0.31	100
castle	0.61	0.39	0.48	100
caterpillar	0.31	0.42	0.35	100
cattle	0.35	0.21	0.26	100
chair	0.63	0.66	0.64	100
chimpanzee	0.48	0.53	0.50	100
clock	0.41	0.20	0.27	100
cloud	0.47	0.55	0.51	100
cockroach	0.51	0.65	0.57	100
couch	0.29	0.24	0.26	100
crab	0.25	0.30	0.27	100
crocodile	0.18	0.23	0.20	100
cup	0.52	0.50	0.51	100
dinosaur	0.36	0.32	0.34	100
dolphin	0.34	0.45	0.38	100
elephant	0.28	0.23	0.25	100
flatfish	0.39	0.22	0.28	100
forest	0.41	0.21	0.28	100
fox	0.30	0.31	0.30	100
girl	0.17	0.10	0.13	100
hamster	0.35	0.33	0.34	100
house	0.29	0.16	0.21	100
kangaroo	0.16	0.17	0.16	100
keyboard	0.31	0.40	0.35	100
lamp	0.42	0.21	0.28	100
lawn_mower	0.67	0.56	0.61	100
leopard	0.27	0.35	0.30	100
lion	0.29	0.37	0.33	100
lizard	0.17	0.25	0.20	100
lobster	0.17	0.15	0.16	100
man	0.24	0.27	0.25	100
maple_tree	0.48	0.42	0.45	100
motorcycle	0.40	0.65	0.49	100
mountain	0.47	0.30	0.37	100
mouse	0.15	0.18	0.17	100
mushroom	0.24	0.31	0.27	100
oak_tree	0.48	0.55	0.51	100
orange	0.45	0.41	0.43	100
orchid	0.46	0.49	0.48	100
otter	0.09	0.06	0.07	100
palm_tree	0.49	0.42	0.45	100
pear	0.49	0.23	0.31	100
pickup_truck	0.54	0.37	0.44	100
pine_tree	0.32	0.21	0.25	100

plain	0.65	0.72	0.69	100
plate	0.60	0.28	0.38	100
poppy	0.49	0.39	0.43	100
porcupine	0.34	0.47	0.39	100
possum	0.14	0.20	0.16	100
rabbit	0.23	0.22	0.22	100
raccoon	0.19	0.19	0.19	100
ray	0.28	0.25	0.27	100
road	0.56	0.74	0.64	100
rocket	0.45	0.49	0.47	100
rose	0.28	0.60	0.38	100
sea	0.56	0.70	0.62	100
seal	0.17	0.10	0.13	100
shark	0.28	0.20	0.23	100
shrew	0.15	0.15	0.15	100
skunk	0.50	0.63	0.56	100
skyscraper	0.59	0.55	0.57	100
snail	0.10	0.16	0.12	100
snake	0.20	0.17	0.18	100
spider	0.31	0.29	0.30	100
squirrel	0.18	0.06	0.09	100
streetcar	0.41	0.24	0.30	100
sunflower	0.56	0.79	0.66	100
sweet_pepper	0.29	0.22	0.25	100
table	0.32	0.15	0.20	100
tank	0.46	0.42	0.44	100
telephone	0.25	0.34	0.29	100
television	0.49	0.33	0.40	100
tiger	0.34	0.13	0.19	100
tractor	0.25	0.37	0.30	100
train	0.18	0.41	0.25	100
trout	0.36	0.43	0.39	100
tulip	0.27	0.15	0.19	100
turtle	0.29	0.19	0.23	100
wardrobe	0.63	0.73	0.68	100
whale	0.36	0.57	0.44	100
willow_tree	0.29	0.24	0.26	100
wolf	0.38	0.30	0.34	100
woman	0.14	0.17	0.15	100
worm	0.15	0.33	0.20	100
accuracy			0.34	10000
macro avg	0.35	0.34	0.33	10000
weighted avg	0.35	0.34	0.33	10000