Exercise on dog breed classification.

In this exercise you will perform dog breed classification on images from https://www.kaggle.com/datasets/abhinavkrjha/dog-breed-classification

You goal is to use pytorch to read the dataset and train a CNN of your choice. Since the data is small, you should use transfer learning to get the best results.

The kaggle also comes with a Keras version of the task (https://www.kaggle.com/code/stpeteishii/dog-breed-classify-densenet201). You can use this as a reference for data loading. However, you should do this excercise using pytorch.

You might also look into data augmentation https://pytorch.org/docs/stable/torchvision/transforms.html to get the best results.

To easily download data from kaggle you should create a kaggle account. Generate an API key and upload it to colab.

```
! pip install -q kaggle
```

```
# Upload kaggle API key file
from google.colab import files
uploaded = files.upload()

!mkdir '/root/.kaggle'
!cp 'kaggle.json' '/root/.kaggle/.'
!chmod 600 '/root/.kaggle/kaggle.json'
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
```

```
!kaggle datasets download -d abhinavkrjha/dog-breed-classification
```

```
Downloading dog-breed-classification.zip to /content
 99% 111M/112M [00:03<00:00, 45.1MB/s]
100% 112M/112M [00:03<00:00, 33.7MB/s]
```

```
!unzip dog-breed-classification.zip
```

```
Archive:  dog-breed-classification.zip
  inflating: Affenhuahua dog/Image_1.jpg
  inflating: Affenhuahua dog/Image_10.jpg
  inflating: Affenhuahua dog/Image_11.jpg
  inflating: Affenhuahua dog/Image_12.jpg
  inflating: Affenhuahua dog/Image_13.jpg
  inflating: Affenhuahua dog/Image_14.jpg
  inflating: Affenhuahua dog/Image_15.jpg
  inflating: Affenhuahua dog/Image_16.jpg
  inflating: Affenhuahua dog/Image_17.jpg
  inflating: Affenhuahua dog/Image_18.jpg
  inflating: Affenhuahua dog/Image_19.jpg
  inflating: Affenhuahua dog/Image_2.jpg
  inflating: Affenhuahua dog/Image_20.jpg
  inflating: Affenhuahua dog/Image_21.png
  inflating: Affenhuahua dog/Image_22.jpg
  inflating: Affenhuahua dog/Image_23.jpg
  inflating: Affenhuahua dog/Image_24.jpg
  inflating: Affenhuahua dog/Image_25.jpg
  inflating: Affenhuahua dog/Image_26.png
  inflating: Affenhuahua dog/Image_27.jpg
  inflating: Affenhuahua dog/Image_28.jpg
  inflating: Affenhuahua dog/Image_29.jpg
  inflating: Affenhuahua dog/Image_3.jpg
  inflating: Affenhuahua dog/Image_30.jpg
  inflating: Affenhuahua dog/Image_31.jpg
  inflating: Affenhuahua dog/Image_32.jpg
  inflating: Affenhuahua dog/Image_33.jpg
  inflating: Affenhuahua dog/Image_34.jpg
  inflating: Affenhuahua dog/Image_35.jpg
  inflating: Affenhuahua dog/Image_36.jpg
  inflating: Affenhuahua dog/Image_37.jpg
  inflating: Affenhuahua dog/Image_38.png
  inflating: Affenhuahua dog/Image_39.jpg
  inflating: Affenhuahua dog/Image_4.png
  inflating: Affenhuahua dog/Image_40.jpg
  inflating: Affenhuahua dog/Image_41.jpg
  inflating: Affenhuahua dog/Image_42.jpg
  inflating: Affenhuahua dog/Image_43.jpg
  inflating: Affenhuahua dog/Image_44.jpg
  inflating: Affenhuahua dog/Image_45.jpg
```

```
  inflating: Affenhuahua dog/Image_46.png
  inflating: Affenhuahua dog/Image_47.jpg
  inflating: Affenhuahua dog/Image_48.jpg
  inflating: Affenhuahua dog/Image_49.jpg
  inflating: Affenhuahua dog/Image_5.jpg
  inflating: Affenhuahua dog/Image_50.jpg
  inflating: Affenhuahua dog/Image_6.jpg
  inflating: Affenhuahua dog/Image_7.jpg
  inflating: Affenhuahua dog/Image_8.jpg
  inflating: Affenhuahua dog/Image_9.jpg
  inflating: Afgan Hound dog/Image_1.jpg
  inflating: Afgan Hound dog/Image_10.jpg
  inflating: Afgan Hound dog/Image_11.jpg
  inflating: Afgan Hound dog/Image_12.jpg
  inflating: Afgan Hound dog/Image_13.jpg
  inflating: Afgan Hound dog/Image_14.jpg
  inflating: Afgan Hound dog/Image_15.jpg
```

```
import random
import numpy as np
import cv2
import os
from matplotlib import pyplot as plt
from tqdm.notebook import tqdm

import torch
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
from torchvision import transforms
from torchvision import models as models
from sklearn.metrics import confusion_matrix

# To guarantee reproducible results
torch.manual_seed(2)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(2)
```

```
# importing the zipfile module
from zipfile import ZipFile

# loading the temp.zip and creating a zip object
with ZipFile("/content/dog-breed-classification.zip", 'r') as zObject:

    # Extracting all the members of the zip
    # into a specific location.
    zObject.extractall(
        path="/content/dog-breed-classification/")
zObject.close()
```

```
directory = '/content/dog-breed-classification'
```

```
dir = '/content/Bulldog dog'
print(len(os.listdir(dir)))
```

```
50
```

```
Name=[]
for file in os.listdir(directory):
    Name+=[file]
print(Name)
print(len(Name))
```

```
['Bulldog dog', 'Akita dog', 'Beagle dog', 'Bocker dog', 'Boxer dog', 'Affenhuahua dog', 'Belgian Tervuren dog', 'Bugg d
14
```

```
for i in Name:
    print(i)
```

```
Bulldog dog
Akita dog
Beagle dog
Bocker dog
Boxer dog
Affenhuahua dog
Belgian Tervuren dog
Bugg dog
Alaskan Malamute dog
Bichon Frise dog
Auggie dog
Borzoi dog
Afgan Hound dog
American Bulldog dog
```

Preparing Dataset

Start: 14 folder of 14 dog breeds -> 2 folders of train and test set -> each folder have 14 dog breeds

```
newpath1 = '/content/dog-breed-classification/train/'
newpath2 = '/content/dog-breed-classification/test/'
os.makedirs(newpath1)
os.makedirs(newpath2)
```

```
import os
import shutil

len_all = 0
for i in Name:

    source_folder = '/content/dog-breed-classification/{}/'.format(i)

    destination_train = '/content/dog-breed-classification/train/{}/'.format(i)
    if not os.path.exists(destination_train):
        os.makedirs(destination_train)

    destination_test = '/content/dog-breed-classification/test/{}/'.format(i)
    if not os.path.exists(destination_test):
        os.makedirs(destination_test)

    len_data = len(os.listdir(source_folder))
    print(len_data)
    len_all += len_data
    count = 0
    # fetch all files
    for file_name in os.listdir(source_folder):
        # construct full file path
        source = source_folder + file_name
        des_train = destination_train + file_name
        des_test = destination_test + file_name

        # copy only files
        if os.path.isfile(source):
            if count <= len_data*0.8:
                shutil.copy(source, des_train)
                # print('copied', file_name)
            else:
                shutil.copy(source, des_test)
                # print('copied', file_name)
        count += 1
print(len_all)
```

```
50
49
50
50
50
50
50
```

```
50
50
50
50
50
50
48
697
```

```
### Helper function to display image from dataset ###
def getImageFromDataset(dataset, idx):
  sampleImage, sampleLabel = dataset.__getitem__(idx)
  ### Revert transformation ###
  sampleImage = ((sampleImage.permute(1,2,0).numpy() * np.array([0.229, 0.224, 0.225])) + np.
  sampleImage = sampleImage.astype(np.uint8)
  sampleClassName = dataset.classes[sampleLabel]
  return sampleImage, sampleClassName
```

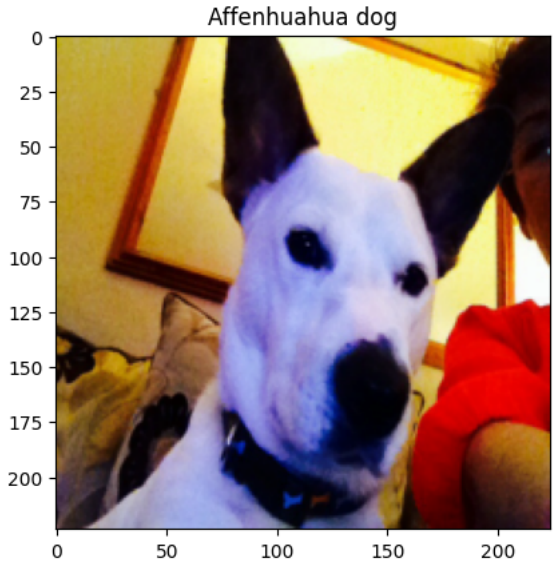
```
### Dataloader for our dataset ###
transform = transforms.Compose([transforms.Resize((224,224)), transforms.ToTensor(), transfor
ddTrainDataset = ImageFolder('/content/dog-breed-classification/train/', transform=transform)
ddTestDataset = ImageFolder('/content/dog-breed-classification/test/', transform=transform)

print('Train images :', len(ddTrainDataset))
print('Test images :', len(ddTestDataset))

```

```
Train images : 570
Test images : 124
```

```
image,name = getImageFromDataset(ddTrainDataset, 7)
plt.imshow(image)
plt.title(name)
plt.show()
```



Transfer learning from pretrained model

```
### Train and test helper function ###
def testModel(testDatasetLoader, net):
  net.eval()
  correctImages = 0
  totalImages = 0
  allLabels = []
  allPredicted = []
  testingProgressbar = tqdm(enumerate(testDatasetLoader), total=len(testDatasetLoader))
  with torch.no_grad():
    for batchIdx, batchData in testingProgressbar:
      images, labels = batchData

    #   images, labels = images.cuda(), labels.cuda()

      outputs = net(images)
      _, predicted = torch.max(outputs, 1)

      correctImages += (predicted == labels).sum().item()
      totalImages += labels.size(0)

      accumulateAccuracy = round((correctImages/totalImages)*100,4)
      testingProgressbar.set_description("Testing accuracy: {}".format(accumulateAccuracy ) )

      allLabels.append(labels)
      allPredicted.append(predicted)
  allLabels = torch.cat(allLabels).cpu().numpy()
  allPredicted = torch.cat(allPredicted).cpu().numpy()
  return correctImages, totalImages, allLabels, allPredicted

def trainAndTestModel(trainDatasetLoader, testDatasetLoader, net, optimizer,scheduler, criter

  bestAccuracy = 0
  correctImages = 0
  totalImages = 0
  trainacc = []
  testacc = []
  lossall = []
  for currentEpoch in tqdm(range(trainEpoch), desc='Overall Training Progress:'):
    trainingLoss = 0.0
    net.train()
    print('Epoch',str(currentEpoch+1),'/',str(trainEpoch))
    trainingProgressbar = tqdm(enumerate(trainDatasetLoader), total=len(trainDatasetLoader))
    for batchIdx, batchData in trainingProgressbar:
      images, labels = batchData
    #   images, labels = images.cuda(), labels.cuda()

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = net(images)
      loss = criterion(outputs, labels)

      _, predicted = torch.max(outputs, 1)
      correctImages += (predicted == labels).sum().item()
      totalImages += labels.size(0)

      loss.backward()
      optimizer.step()


      trainingLoss += loss.item()
      accumulateAccuracy = round((correctImages/totalImages)*100,4)
      trainingProgressbar.set_description("Training accuracy: {} loss: {}".format(accumulateA
      trainacc.append(accumulateAccuracy)
    lossall.append(round(loss.item(),4))
    scheduler.step(trainingLoss)
    correctImages, totalImages, allLabels, allPredicted = testModel(testDatasetLoader, net)
```

```
    testAccuracy = round((correctImages/totalImages)*100,2)
    testacc.append(testAccuracy)

    print('='*10)

    if testAccuracy > bestAccuracy:
      bestAccuracy = testAccuracy
      bestPredicted = allPredicted
      bestNet = net

  return bestAccuracy, bestPredicted, allLabels, bestNet, trainacc, testacc, lossall
```

```
pretrainNet = models.resnet34(pretrained=True)
# pretrainNet = models.densenet201()
# or you can use this
num_ftrs = pretrainNet.fc.in_features
pretrainNet.fc = nn.Linear(num_ftrs, 14)
# instead of figuring out the dimension from previous layer
# pretrainNet.fc = nn.Linear(512, 14)
# pretrainNet.cuda()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(pretrainNet.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, verbose=True)

ddTrainDatasetLoader = DataLoader(ddTrainDataset, batch_size=16, shuffle=True, num_workers=4,
ddTestDatasetLoader = DataLoader(ddTestDataset, batch_size=16, shuffle=False, num_workers=4,
```

```
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depr
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-
100%|██████████| 83.3M/83.3M [00:00<00:00, 128MB/s]
/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:561: UserWarning: This DataLoader will create 4 wo
  warnings.warn(_create_warning_msg(
```

Train Model

```
bestAccuracy, bestPredicted, allLabels, bestNet, trainacc, testacc, loss  = trainAndTestModel
                                                                    pretrainNet,
                                                                    optimizer, scheduler, cri
                                                                    trainEpoch=15)
```

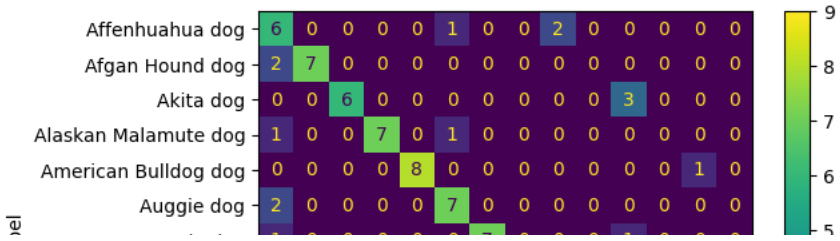
```
Overall Training Progress:: 15/15 [43:04<00:00, 171.44s/it]
Epoch 1 / 15
Training accuracy: 18.0702 loss: 2.6912: 100% 36/36 [02:48<00:00, 3.87s/it]
Testing accuracy: 8.0645: 100% 8/8 [00:13<00:00, 1.42s/it]
==========
Epoch 2 / 15
Training accuracy: 27.0893 loss: 3.0881: 100% 36/36 [02:38<00:00, 3.86s/it]
Testing accuracy: 30.6452: 100% 8/8 [00:13<00:00, 1.47s/it]
==========
Epoch 3 / 15
Training accuracy: 43.3718 loss: 1.6845: 100% 36/36 [02:37<00:00, 3.87s/it]
Testing accuracy: 24.1935: 100% 8/8 [00:13<00:00, 1.50s/it]
==========
Epoch 4 / 15
Training accuracy: 53.4582 loss: 0.6395: 100% 36/36 [02:43<00:00, 3.79s/it]
Testing accuracy: 27.4194: 100% 8/8 [00:12<00:00, 1.24s/it]
==========
Epoch 5 / 15
Training accuracy: 57.9251 loss: 2.0475: 100% 36/36 [02:37<00:00, 3.78s/it]
Testing accuracy: 50.0: 100% 8/8 [00:13<00:00, 1.31s/it]
==========
Epoch 6 / 15
Training accuracy: 71.0375 loss: 1.6774: 100% 36/36 [02:37<00:00, 3.78s/it]
Testing accuracy: 48.3871: 100% 8/8 [00:13<00:00, 1.41s/it]
==========
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

cm = confusion_matrix(allLabels, bestPredicted)
print(classification_report(allLabels, bestPredicted))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=ddTrainDataset.classes)
disp.plot()
plt.show()
```

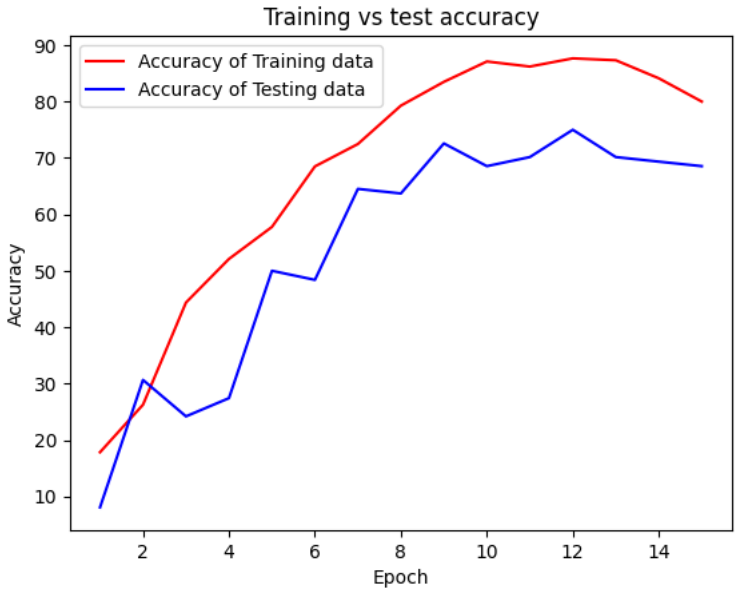
	precision	recall	f1-score	support
0	0.35	0.67	0.46	9
1	0.88	0.78	0.82	9
2	0.86	0.67	0.75	9
3	1.00	0.78	0.88	9
4	1.00	0.89	0.94	9
5	0.70	0.78	0.74	9
6	0.88	0.78	0.82	9
7	1.00	0.44	0.62	9
8	0.64	1.00	0.78	9
9	0.90	1.00	0.95	9
10	0.54	0.78	0.64	9
11	1.00	0.56	0.71	9
12	0.78	0.78	0.78	9
13	1.00	0.57	0.73	7
accuracy			0.75	124
macro avg	0.82	0.75	0.76	124
weighted avg	0.82	0.75	0.76	124



Plot accuracy and loss

```
1 trainacc = trainacc[33::34]
```

```
1 epochs = range(1,16)
2 plt.plot(epochs, trainacc, 'r', label='Accuracy of Training data')
3 plt.plot(epochs, testacc, 'b', label='Accuracy of Testing data')
4 plt.title('Training vs test accuracy')
5 plt.xlabel('Epoch')
6 plt.ylabel('Accuracy')
7 plt.legend(loc=0)
8 plt.figure()
9 plt.show()
```




```
<Figure size 640x480 with 0 Axes>
```

```
1 plt.plot(epochs, loss, 'r')
2 plt.title('Loss of Training data')
3 plt.xlabel('Epoch')
4 plt.ylabel('Loss')
5 plt.figure
6 plt.show()
```


Loss of Training data
Loss
Epoch
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
8
10
12
14