Computer Vision Coursework Submission (IN3060/INM460)

Student name, ID and cohort: Rajani Mohan Janipalli (210049506) - PG

Training and testing script for baseline CNN and VGG16 CNN models.

Google Colab Setup

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')

## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module.
```

```
Mounted at /content/drive
```

Updating Open CV

In [ ]:

```
!pip install opencv-python==4.5.5.64

## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module.
```

```
Requirement already satisfied: opencv-python==4.5.5.64 in /usr/local/lib/python3.7/dist-packages (4.5.5.64)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python==4.5.5.64) (1.21.6)
```

Check the version open CV

In [ ]:

```
!pip show opencv-python

## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module.
```

```
Name: opencv-python
Version: 4.5.5.64
Summary: Wrapper package for OpenCV python bindings.
Home-page: https://github.com/skvark/opencv-python
Author: None
Author-email: None
License: MIT
Location: /usr/local/lib/python3.7/dist-packages
Requires: numpy
Required-by: imgaug, dopamine-rl, albumentations
```

Assign path to link the folder containing the colab notebook.

In [ ]:

```
import os


GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Computer Vision Coursework/CW_Folder_PG/Code'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))


## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module.
```

```
['Copy of CV_CW_CNN3.ipynb', 'CV_CW_CNN1.ipynb', 'CV_CW_draft1.ipynb', 'dummy_CNN3.ipynb', 'CV_CW_CNN3.ipynb', 'CV_CW_SVM_MLP_1.ip
ynb']
```

Assign path to link the folder containing the train dataset.

In [ ]:

```
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE_DS = 'Colab Notebooks/Computer Vision Coursework/CW_Folder_PG/CW_Dataset'
GOOGLE_DRIVE_PATH_DS = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE_DS)
print(os.listdir(GOOGLE_DRIVE_PATH_DS))


# Code from above cell was modified in the this cell, as the dataset resides in a different folder.
```

```
['CW_Dataset.zip']
```

Copy and unzip the dataset directly in colab server.

In [ ]:

```
# Identify path to zipped dataset
CW_zip_path = os.path.join(GOOGLE_DRIVE_PATH_DS, 'CW_Dataset.zip')

# Copy it to Colab
!cp '{CW_zip_path}' .

# Unzip it
!yes|unzip -q CW_Dataset.zip

# Delete zipped version from Colab (not from Drive)
!rm CW_Dataset.zip


## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module.
```

Import necessary libraries.

In [ ]:

```
import cv2
from sklearn.model_selection import train_test_split
from skimage import img_as_ubyte, io, color
from sklearn.cluster import MiniBatchKMeans
from sklearn import svm, metrics
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

%matplotlib inline

## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module.
```

Create a fucntio to import data image names and labels as encoded numbers into a pandas data frame.

In [ ]:

```
import pandas as pd


def create_imagename_dataframe(path):
    """Create a dataframe of images and labels from selected directories"""
    data_df = pd.DataFrame(columns=["imgname", "label"])
    data_df["imgname"] = [file for file in sorted(os.listdir(os.path.join(path))) if file.endswith('.jpg')]
    label_set = np.loadtxt(os.path.join('labels', 'list_label_{}.txt'.format(path)), dtype='str')
    label_nums = [] # create an empty list to append label numbers.
    for i in range(len(label_set)): # execute a for loop to extract the exact labels from data and append them to a list.
      label_nums.append(label_set[i][1])

    # label_names = label_names = ['Suprise' if p == '1' else 'Fear' if p == '2' else 'Disgust' if p == '3' else 'Happiness' if p

    data_df["label"] = label_nums

    data_df.to_csv (r'{}_df_csv.csv'.format(path), index = False, header=True)

    print('Create datafame with file name {}_df_csv.csv'.format(path))

## CODING REFERENCE: Code was taken from Lab Tutorial 07 of Computer Vision - IN3060/INM460 module
## and modified as per the requirement in this task.
## https://tutorial.eyehunts.com/python/python-elif-in-list-comprehension-conditionals-example-code/
## https://medium.com/analytics-vidhya/implementing-cnn-in-pytorch-with-custom-dataset-and-transfer-learning-1864daac14cc
```

In [ ]:

```
create_imagename_dataframe('train')
```

```
Create datafame with file name train_df_csv.csv
```

In [ ]:

```
create_imagename_dataframe('test')
```

```
Create datafame with file name test_df_csv.csv
```

Create a class to convert pandas to pytorch dataset.

In [ ]:

```
from torch.utils.data import Dataset
import pandas as pd
import os
from PIL import Image
import torch

class CreateDataset(Dataset):
    def __init__(self, root_dir, annotation_file, transform=None):
        self.root_dir = root_dir
        self.annotations = pd.read_csv(annotation_file)
        self.transform = transform

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_id = self.annotations.iloc[idx, 0]
        img = Image.open(os.path.join(self.root_dir, img_id))
        y_label = self.annotations.iloc[idx, 1]

        if self.transform is not None:
            img = self.transform(img)

        return (img, y_label)


## CODING REFERENCE
## https://medium.com/analytics-vidhya/implementing-cnn-in-pytorch-with-custom-dataset-and-transfer-learning-1864daac14cc
## https://pytorch.org/tutorials/recipes/recipes/custom_dataset_transforms_loader.html
```

Import additional libraries.

In [ ]:

```
import torch
import torch.nn as nn
```

```
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
# from Model import CNN
# from Dataset import CatsAndDogsDataset
from tqdm import tqdm
device = ("cuda" if torch.cuda.is_available() else "cpu")


## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

Create transformation object.

In [ ]:

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

## CODING REFERENCE
## https://medium.com/analytics-vidhya/implementing-cnn-in-pytorch-with-custom-dataset-and-transfer-learning-1864daac14cc
```

Create datasets and dataloaders for training and test data.

In [ ]:

```
train_dataset = CreateDataset("train","train_df_csv.csv",transform=transform)
test_dataset = CreateDataset("test","test_df_csv.csv",transform=transform)
train_loader = DataLoader(dataset=train_dataset, shuffle=True, batch_size=16,num_workers=1)
test_loader = DataLoader(dataset=test_dataset, shuffle=False, batch_size=16,num_workers=1)

## CODING REFERENCE
## https://medium.com/analytics-vidhya/implementing-cnn-in-pytorch-with-custom-dataset-and-transfer-learning-1864daac14cc
```

In [ ]:

```
print('Training set size (num images)', len(train_dataset))
print('Testing set size (num images)', len(test_dataset))

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

```
Training set size (num images) 12271
Testing set size (num images) 3068
```

In [ ]:

```
dataiter_tr1 = iter(train_loader)
images_tr1, labels_tr1 = dataiter_tr1.next()
print(type(images_tr1))
print(images_tr1.shape)
print(labels_tr1.shape)

## CODING REFERENCE: https://discuss.pytorch.org/t/how-to-find-shape-and-columns-for-dataloader/34901
```

```
<class 'torch.Tensor'>
torch.Size([16, 3, 100, 100])
torch.Size([16])
```

In [ ]:

```
dataiter_te1 = iter(test_loader)
images_te1, labels_te1 = dataiter_te1.next()
print(type(images_te1))
print(images_te1.shape)
print(labels_te1.shape)

## CODING REFERENCE: https://discuss.pytorch.org/t/how-to-find-shape-and-columns-for-dataloader/34901
```

```
<class 'torch.Tensor'>
torch.Size([16, 3, 100, 100])
torch.Size([16])
```

In [ ]:

```
print('Training set size (num mini-batches)', len(train_loader))
print('Testing set size (num mini-batches)', len(test_loader))

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

```
Training set size (num mini-batches) 767
Testing set size (num mini-batches) 192
```

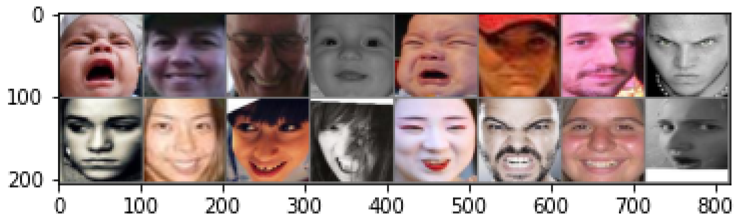
In [ ]:

```
import torchvision
import torchvision.transforms as transforms
```

Create and use function to display images from dataloaders.

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

# function to show an image
def imagedisplay(img):
    img = img / 2 + 0.5     # Unnormalize: back to range [0, 1] just for showing the images
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))     # Reshape: C, H, W -> H, W, C
    plt.show()


# get some random training images
dataiter_tr2 = iter(train_loader)
images_tr2, labels_tr2 = dataiter_tr2.next()

# show images and print labels
imagedisplay(torchvision.utils.make_grid(images_tr2))
first_labels = [labels_tr2[label].item() for label in labels_tr2]
print('Ground-truth:', first_labels)
print('Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral')

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```



```
Ground-truth: [4, 5, 5, 5, 4, 5, 5, 4, 4, 5, 5, 4, 5, 4, 5, 4]
Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral
```

Create network architechture for baseline CNN model.

In [ ]:

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 22 * 22, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 8)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), 16 * 22 * 22)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()


## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
## https://github.com/PacktPublishing/Convolutional-Neural-Network-with-PyTorch/blob/master/cnn.py
```

Note: The above architecture in terms of number of nodes was obtained after mutiple trial and errors attemps and was the most suitable for the given data.

Define loss calculation method and optimization configuration for baseline CNN.

In [ ]:

```
import torch.optim as optim

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

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

Train baseline CNN.

In [ ]:

```
import time

t0 = time.time()

for epoch in range(20):  # loop over the training set two times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels_train = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels_train)
        loss.backward()
        optimizer.step()

        # print statistics (loss.item() returns the mean loss in the mini-batch)
        running_loss += loss.item()
        if i % 500 == 499:    # print every 200 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training: total time in seconds =', time.time() - t0)


## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
## https://github.com/PacktPublishing/Convolutional-Neural-Network-with-PyTorch/blob/master/cnn.py
```

```
[1,   500] loss: 0.436
[2,   500] loss: 0.330
[3,   500] loss: 0.261
[4,   500] loss: 0.230
[5,   500] loss: 0.203
[6,   500] loss: 0.184
[7,   500] loss: 0.163
[8,   500] loss: 0.145
[9,   500] loss: 0.126
[10,   500] loss: 0.109
[11,   500] loss: 0.087
[12,   500] loss: 0.069
[13,   500] loss: 0.052
[14,   500] loss: 0.040
[15,   500] loss: 0.026
[16,   500] loss: 0.017
[17,   500] loss: 0.020
[18,   500] loss: 0.014
[19,   500] loss: 0.014
[20,   500] loss: 0.011
Finished Training: total time in seconds = 924.2781212329865
```

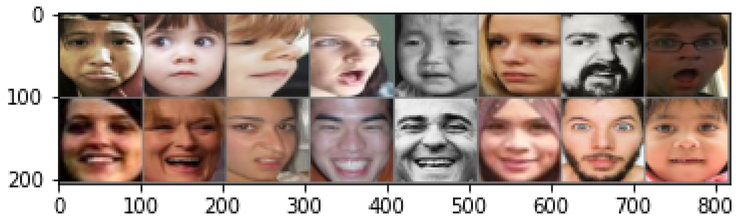
Save the baseline CNN model.

In [ ]:

```
MODELPATH = 'drive/My Drive/Colab Notebooks/Computer Vision Coursework/CW_Folder_PG/Models/CNN_FER.pt'

torch.save(net.state_dict(), MODELPATH)
```

In [ ]:

```
dataiter_te2 = iter(test_loader)
images_te2, labels_te2 = dataiter_te2.next()

# show images and print labels
imagedisplay(torchvision.utils.make_grid(images_te2))
testfirst_labels = [labels_te2[label].item() for label in labels_te2]
print('Ground-truth:', testfirst_labels)
print('Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral')

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```



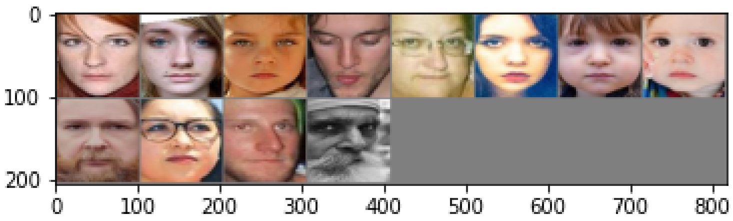
```
Ground-truth: [5, 1, 5, 1, 5, 5, 1, 1, 5, 5, 1, 5, 5, 5, 1, 5]
Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral
```

Test baseline CNN on test data.

In [ ]:

```
# Estimate average accuracy
correct = 0
total = 0
with torch.no_grad():              # Avoid backprop at test
    for data in test_loader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()


print(f"Accuracy of the network on the 3068 test images: {100 * correct / total}%")
imagedisplay(torchvision.utils.make_grid(images))
print('Ground-truth:', labels)
print('Ground-truth:', predicted)
print('Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral')

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

```
Accuracy of the network on the 3068 test images: 73.53324641460235%
```



```
Ground-truth: tensor([7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
Ground-truth: tensor([7, 7, 7, 7, 4, 7, 5, 7, 3, 4, 4, 7])
Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral
```

Label explanation: (taken from Readme file of dataset)

1: Surprise

2: Fear

3: Disgust

4: Happiness

5: Sadness

6: Anger

7: Neutral

Load VGG16 pretrained model and define its loss calculation method and optimization configuration.

In [ ]:

```
from torchvision import models
import torch.optim as optim

vgg16cnn = models.vgg16(pretrained = True)
criterionvgg = nn.CrossEntropyLoss().cuda()
optimizervgg = optim.SGD(vgg16cnn.parameters(), lr=0.001, momentum=0.9)

## CODING REFERENCE:
## Code was taken from Lab Tutorial 09 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
```

Train VGG16 CNN model.

In [ ]:

```
import time

t0 = time.time()

for epoch in range(15):  # loop over the training set two times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        vgg16cnn = vgg16cnn.cuda()
        inputs, labels_train = data

        # zero the parameter gradients
        optimizervgg.zero_grad()

        # forward + backward + optimize
        outputs = vgg16cnn(inputs.cuda())
        loss = criterionvgg(outputs.cuda(), labels_train.cuda())
        loss.backward()
        optimizervgg.step()

        # print statistics (loss.item() returns the mean loss in the mini-batch)
        running_loss += loss.item()
        if i % 500 == 499:    # print every 200 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 500))
            running_loss = 0.0

print('Finished Training: total time in seconds =', time.time() - t0)

## CODING REFERENCE:
## Code was taken from Lab Tutorial 08 of Computer Vision - IN3060/INM460 module and modified as per the requirement in this task.
```

```
[1,   500] loss: 1.432
[2,   500] loss: 0.825
[3,   500] loss: 0.645
[4,   500] loss: 0.484
[5,   500] loss: 0.349
[6,   500] loss: 0.245
```

```
[7,   500] loss: 0.196
[8,   500] loss: 0.119
[9,   500] loss: 0.097
[10,   500] loss: 0.071
[11,   500] loss: 0.075
[12,   500] loss: 0.044
[13,   500] loss: 0.069
[14,   500] loss: 0.037
[15,   500] loss: 0.037
Finished Training: total time in seconds = 633.7178964614868
```

Save VGG16 CNN model.

In [ ]:

```
VGG16CNN1MODELPATH = 'drive/My Drive/Colab Notebooks/Computer Vision Coursework/CW_Folder_PG/Models/VGG16CNN1_FER.pt'

torch.save(vgg16cnn.state_dict(), VGG16CNN1MODELPATH)
```

Test VGG16 CNN model on test data.

In [ ]:

```
# Estimate average accuracy
correct = 0
total = 0
with torch.no_grad():             # Avoid backprop at test
    for data in test_loader:
        vgg16cnn = vgg16cnn.cuda()
        images, labels = data
        outputs = vgg16cnn(images.cuda())
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels.cuda()).sum().item()

print(f"Accuracy of the network on the 3068 test images: {100 * correct / total}%")
```

```
Accuracy of the network on the 3068 test images: 79.98696219035202%
```