facial-expression

April 2, 2024

[1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[ ]:
```
# to be ran only once
# check the folders on the left, if aligned exists ==> do not run
!unzip "/content/gdrive/My Drive/DL_project_2022/data/data.zip"
```

[ ]:
```
# create labels
# NOTE: the images and labels are assumed to be in the same order
path_to_labels = 'gdrive/MyDrive/DL_project_2022/data/list_patition_label.txt'
label =[]
with open(path_to_labels, "r") as file1:
    for line in file1:
        label.append(int(line.split()[1]))
```

[ ]:
```
# sort the images according to their labels
# create 7 lists containing the name of the images corresponding to the labels

# we start with label 1
path_to_labels = 'gdrive/MyDrive/DL_project_2022/data/list_patition_label.txt'
label_1 = []
label_2 = []
label_3 = []
label_4 = []
label_5 = []
label_6 = []
label_7 = []
with open(path_to_labels, "r") as file1:
    for line in file1:
        linesplit = line.split()
        if int(linesplit[1]) == 1:
          splitted = linesplit[0].split('.')
          new = splitted[0]+'_aligned.'+splitted[1]
          label_1.append(new)
        elif int(linesplit[1]) == 2:
          splitted = linesplit[0].split('.')
```

```
          new = splitted[0]+'_aligned.'+splitted[1]
          label_2.append(new)
        elif int(linesplit[1]) == 3:
          splitted = linesplit[0].split('.')
          new = splitted[0]+'_aligned.'+splitted[1]
          label_3.append(new)
        elif int(linesplit[1]) == 4:
          splitted = linesplit[0].split('.')
          new = splitted[0]+'_aligned.'+splitted[1]
          label_4.append(new)
        elif int(linesplit[1]) == 5:
          splitted = linesplit[0].split('.')
          new = splitted[0]+'_aligned.'+splitted[1]
          label_5.append(new)
        elif int(linesplit[1]) == 6:
          splitted = linesplit[0].split('.')
          new = splitted[0]+'_aligned.'+splitted[1]
          label_6.append(new)
        elif int(linesplit[1]) == 7:
          splitted = linesplit[0].split('.')
          new = splitted[0]+'_aligned.'+splitted[1]
          label_7.append(new)
        else:
          print('error')
```

[ ]:
```
len(label_1)+len(label_2)+len(label_3)+len(label_4)+len(label_5)+len(label_6)+len(label_7)␣
↪# = 15339 ==> HURRAY
```

[ ]:
```
15339
```

[ ]:
```
# install google api
# status: DO NOT RUN IT
!pip install --upgrade google-api-python-client google-auth-httplib2␣
↪google-auth-oauthlib
```

[ ]:
```
!pip install pydrive
```

[ ]:
```
from google.colab import auth
from oauth2client.client import GoogleCredentials
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

[ ]:
```
# get the file ID of all the pictures
imgID_dict = dict()
fileList = drive.ListFile({'q': "'14tbCY4oaR462OX3C_HLv0HGdkv2dgP1M' in parents␣
 ↪and trashed=false"}).GetList()
for file in fileList:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title = file['title']
  id = file['id']
  imgID_dict[title]=id

# now we have the tuple name of image/ID
```

[ ]:
```
len(imgID_dict) # good good
```

[ ]:
```
15339
```

[ ]:
```
imgID_dict[label_1[0]]
```

[ ]:
```
'13hNEfGzMDvXXzyzEZT4cwZmXpLz6or6C'
```

[ ]:
```
# move images from file to file based on the label
# start with label 1
f1_ID = '1qkaGd2wMcySsxchX3dM4BHbetQDY7EjT'
for img in label_1:
  f = drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f1_ID}]})
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
```

[ ]:
```
# then label 2
f2_ID = '1jFP8HoidkHM823EHer1D9r_JHKKf1eCe'
for img in label_2:
  f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f2_ID}]})
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
```

[ ]:
```
# then label 3
f3_ID = '1DsUh9po9EfeIPg_7w_4t5dBc8_KFAhcF'
for img in label_3:
  f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f3_ID}]})
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
```

[ ]:
```
# then label 4
i = 5380
f4_ID = '12n_gJeoKZwJCU0ngpyrkbZ7kYN5QROb1'
for img in label_4[5381:len(label_4)]:
  f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f4_ID}]})
```

```
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
  i=i+1
  print(i)
```

[ ]:
```
# then label 5
f5_ID = '1subZLaT0W0AerzKS4DKp0tFVVY7YezPH'
j=0
for img in label_5:
  f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f5_ID}]})
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
  j=j+1
  print(j)
```

[ ]:
```
# then label 6
f6_ID = '1xC72gLWwgh1D_BOsWb85a-5DlvUQDjd2'
k=0
for img in label_6:
  f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f6_ID}]})
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
  k=k+1
```

[ ]:
```
# then label 7
#p=0
f7_ID = '1v3VhNaP5VB7fGfun0tdl-eLaQgfmIJX6'
for img in label_7[2793:len(label_7)]:
  f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f7_ID}]})
  f.SetContentFile('/content/aligned/'+img)
  f.Upload()
  p=p+1
  print(p)
```

[ ]:
```
# what picture is missing in folder 7?
f7List = drive.ListFile({'q': "'1v3VhNaP5VB7fGfun0tdl-eLaQgfmIJX6' in parents␣
 ↪and trashed=false"}).GetList()
title7=[]
for file in f7List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title7.append(file['title'].split('/')[3])
```

[ ]:
```
print("Additional values in first list:", (set(label_7).difference(title7)))
```

```
Additional values in first list: {'test_2657_aligned.jpg'}
```

[ ]:
```
# add test_2657_aligned.jpg to the file 7
img = 'test_2657_aligned.jpg'
f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": f7_ID}]})
f.SetContentFile('/content/aligned/'+img)
f.Upload()
```

[ ]:
```
# verify
f7List = drive.ListFile({'q': "'1v3VhNaP5VB7fGfun0tdl-eLaQgfmIJX6' in parents␣
 ↪and trashed=false"}).GetList()
title7=[]
for file in f7List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title7.append(file['title'].split('/')[3])

print("Additional values in first list:", (set(label_7).difference(title7)))␣
 ↪#GOOD GOOD
```

```
Additional values in first list: set()
```

[ ]:
```
print(f'folder 1 should have: {len(label_1)} images')
f1List = drive.ListFile({'q': "'1qkaGd2wMcySsxchX3dM4BHbetQDY7EjT' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 1 has: {len(f1List)} images')
print('====================================')

print(f'folder 2 should have: {len(label_2)} images')
f2List = drive.ListFile({'q': "'1jFP8HoidkHM823EHer1D9r_JHKKf1eCe' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 2 has: {len(f2List)} images')
print('====================================')

print(f'folder 3 should have: {len(label_3)} images')
f3List = drive.ListFile({'q': "'1DsUh9po9EfeIPg_7w_4t5dBc8_KFAhcF' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 3 has: {len(f3List)} images')
print('====================================')

print(f'folder 4 should have: {len(label_4)} images')
f4List = drive.ListFile({'q': "'12n_gJeoKZwJCU0ngpyrkbZ7kYN5QROb1' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 4 has: {len(f4List)} images')
print('====================================')

print(f'folder 5 should have: {len(label_5)} images')
f5List = drive.ListFile({'q': "'1subZLaT0W0AerzKS4DKp0tFVVY7YezPH' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 5 has: {len(f5List)} images')
```

```
print('==================================')

print(f'folder 6 should have: {len(label_6)} images')
f6List = drive.ListFile({'q': "'1xC72gLWwgh1D_BOsWb85a-5DlvUQDjd2' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 6 has: {len(f6List)} images')
print('==================================')

print(f'folder 7 should have: {len(label_7)} images')
f7List = drive.ListFile({'q': "'1v3VhNaP5VB7fGfun0tdl-eLaQgfmIJX6' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 7 has: {len(f7List)} images')
print('==================================')
print(f'Total images:␣
 ↪{len(f1List)+len(f2List)+len(f3List)+len(f4List)+len(f5List)+len(f6List)+len(f7List)}')
print(f'There should be:␣
 ↪{len(label_1)+len(label_2)+len(label_3)+len(label_4)+len(label_5)+len(label_6)+len(label_7)
```

```
folder 1 should have: 1619 images
folder 1 has: 1619 images
==================================
folder 2 should have: 355 images
folder 2 has: 355 images
==================================
folder 3 should have: 877 images
folder 3 has: 877 images
==================================
folder 4 should have: 5957 images
folder 4 has: 5957 images
==================================
folder 5 should have: 2460 images
folder 5 has: 2460 images
==================================
folder 6 should have: 867 images
folder 6 has: 867 images
==================================
folder 7 should have: 3204 images
folder 7 has: 3204 images
==================================
Total images: 15339
There should be: 15339
```

[ ]:
```
f7List = drive.ListFile({'q': "'1v3VhNaP5VB7fGfun0tdl-eLaQgfmIJX6' in parents␣
 ↪and trashed=false"}).GetList()
print(f'folder 7 has: {len(f7List)} images')
```

```
folder 7 has: 3204 images
```

[ ]:
```
import os

import pandas as pd



#dataSize = pd.Dataframe()

for i in range(1,8):

  path, dirs, files = next(os.walk("/content/gdrive/MyDrive/DL_project_2022/
 ↪data/labelled_folder/" + str(i)))

  #print(len(files))

  file_count = len(files)

  #s1 = pd.Series.append(i)

  #s2 = pd.Series.append(file_count)

  #dataSize = pd.concat(s1,s2)

  #dataSize = pd.

  print('No. of images are: ' + str(file_count) + ' in label ' + str(i))
```

```
No. of images are: 1619 in label 1
No. of images are: 355 in label 2
No. of images are: 877 in label 3
No. of images are: 5957 in label 4
No. of images are: 2460 in label 5
No. of images are: 867 in label 6
No. of images are: 3204 in label 7
```

[ ]:
```
# verify
title1=[]
for file in f1List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title1.append(file['title'].split('/')[3])
print("Missing values in first list:", (set(title1).difference(label_1)))
print("Additional values in first list, folder 1:", (set(label_1).
 ↪difference(title1))) #GOOD GOOD
print('==============================================')
# verify
title2=[]
for file in f2List:
```

```
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title2.append(file['title'].split('/')[3])
print("Missing values in first list:", (set(title2).difference(label_2)))
print("Additional values in first list, folder 2:", (set(label_2).
 ↪difference(title2))) #GOOD GOOD
print('================================================')
# verify
title3=[]
for file in f3List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title3.append(file['title'].split('/')[3])
print("Missing values in first list:", (set(title3).difference(label_3)))
print("Additional values in first list, folder 3:", (set(label_3).
 ↪difference(title3))) #GOOD GOOD
print('================================================')
# verify
title4=[]
for file in f4List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title4.append(file['title'].split('/')[3])
print("Missing values in first list:", (set(title4).difference(label_4)))
print("Additional values in first list, folder 4:", (set(label_4).
 ↪difference(title4))) #NOT GOOD
print('================================================')
# verify
title5=[]
for file in f5List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title5.append(file['title'].split('/')[3])
print("Missing values in first list:", (set(title5).difference(label_5)))
print("Additional values in first list, folder 5:", (set(label_5).
 ↪difference(title5))) #GOOD GOOD
print('================================================')
# verify
title6=[]
for file in f6List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title6.append(file['title'].split('/')[3])
print("Missing values in first list:", (set(title6).difference(label_6)))
print("Additional values in first list, folder 6:", (set(label_6).
 ↪difference(title6))) #GOOD GOOD
print('================================================')
# verify
title7=[]
for file in f7List:
  #print('Title: %s, ID: %s' % (file['title'], file['id']))
  title7.append(file['title'].split('/')[3])
```

```
print("Missing values in first list:", (set(title7).difference(label_7)))
print("Additional values in first list, folder 7:", (set(label_7).
 ↪difference(title7))) #GOOD GOOD
print('=============================================')
```

```
Missing values in first list: set()
Additional values in first list, folder 1: set()
=============================================
Missing values in first list: set()
Additional values in first list, folder 2: set()
=============================================
Missing values in first list: set()
Additional values in first list, folder 3: set()
=============================================
Missing values in first list: set()
Additional values in first list, folder 4: set()
=============================================
Missing values in first list: set()
Additional values in first list, folder 5: set()
=============================================
Missing values in first list: set()
Additional values in first list, folder 6: set()
=============================================
Missing values in first list: set()
Additional values in first list, folder 7: set()
=============================================
```

[ ]:
```
for elm in title4:
  count = title4.count(elm)
  if count>1:
    print(f"the element {elm} is found {count} times") # train_04687_aligned.
 ↪jpg is duplicated it needs to be replaced with train_04649_aligned.jpg
```

```
the element train_04687_aligned.jpg is found 3 times
the element train_04687_aligned.jpg is found 3 times
the element train_04687_aligned.jpg is found 3 times
```

[ ]:
```
file_dict = {'f1':'1qkaGd2wMcySsxchX3dM4BHbetQDY7EjT', 'f2':
 ↪'1jFP8HoidkHM823EHer1D9r_JHKKf1eCe', 'f3':
 ↪'1DsUh9po9EfeIPg_7w_4t5dBc8_KFAhcF',
             'f4':'12n_gJeoKZwJCUOngpyrkbZ7kYN5QROb1', 'f5':
 ↪'1subZLaTOWOAerzKS4DKpOtFVVY7YezPH', 'f6':
 ↪'1xC72gLWwgh1D_BOsWb85a-5DlvUQDjd2',
             'f7':'1v3VhNaP5VB7fGfunOtdl-eLaQgfmIJX6'}
```

[ ]:
```
# replace one train_04687_aligned.jpg with train_04649_aligned.jpg
```

```
# first add train_04649_aligned.jpg
img = 'train_04649_aligned.jpg'
f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id":␣
 ↪file_dict['f4']}]})
f.SetContentFile('/content/aligned/'+img)
f.Upload()

# then delete manually
```

[ ]:
```
# then add train_04687_aligned.jpg again because it was deleted

img = 'train_04687_aligned.jpg'
f= drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id":␣
 ↪file_dict['f4']}]})
f.SetContentFile('/content/aligned/'+img)
f.Upload()
```

[ ]:
```
# move the aligned folder to another folder on my drive
# to be ran ONLY ONCE
# STATUS: RAN ==> do not run it again the folder is created in mydrive ==>
# ==> /content/gdrive/MyDrive/DL_project_2022/data/complete_data
%cd /aligned
%cp -av aligned gdrive/MyDrive/DL_project_2022/data/complete_data/
```

[ ]:
```
!pip install -q torch torchvision
```

1 Building the Neural Networks

[2]:
```
#import torch
#import torch.nn as nn
#import torch.nn.functional as F
#import torch.optim as optim
#import torchvision
#import sklearn
#import numpy as np
#import matplotlib.pyplot as plt
#import copy
#from torchvision import transforms
import tensorflow as tf
from keras.models import Sequential, save
from keras.layers import Dense, Conv2D, Flatten, BatchNormalization, MaxPool2D,␣
 ↪Dropout, Softmax, ReLU
#import math
```

[3]:
```
# get the data and split it to train and validation datasets
dir = '/content/gdrive/MyDrive/DL_project_2022/data/labelled_folder'
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory = dir,
    batch_size = batch_size,
    image_size=(100, 100),
    shuffle = True,
    validation_split = 0.3,
    subset = 'training',
    seed = 123,
    label_mode='categorical')

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    directory = dir,
    batch_size = batch_size,
    image_size=(100, 100),
    shuffle = True,
    validation_split = 0.3,
    subset = 'validation',
    seed = 123,
    label_mode='categorical')
```

```
Found 15339 files belonging to 7 classes.
Using 10738 files for training.
Found 15339 files belonging to 7 classes.
Using 4601 files for validation.
```

[ ]:
```
# or load the saved data
path = '/content/gdrive/MyDrive/DL_project_2022/data/DATASETS/'
train_ds = tf.data.experimental.load(path+'training_dataset/')
val_ds = tf.data.experimental.load(path+'validation_dataset/')
```

[ ]:
```
print(f'the training dataset has {len(train_ds)} batches')
print(f'the validation dataset has {len(val_ds)} batches')
```

```
the training dataset has 336 batches
the validation dataset has 144 batches
```

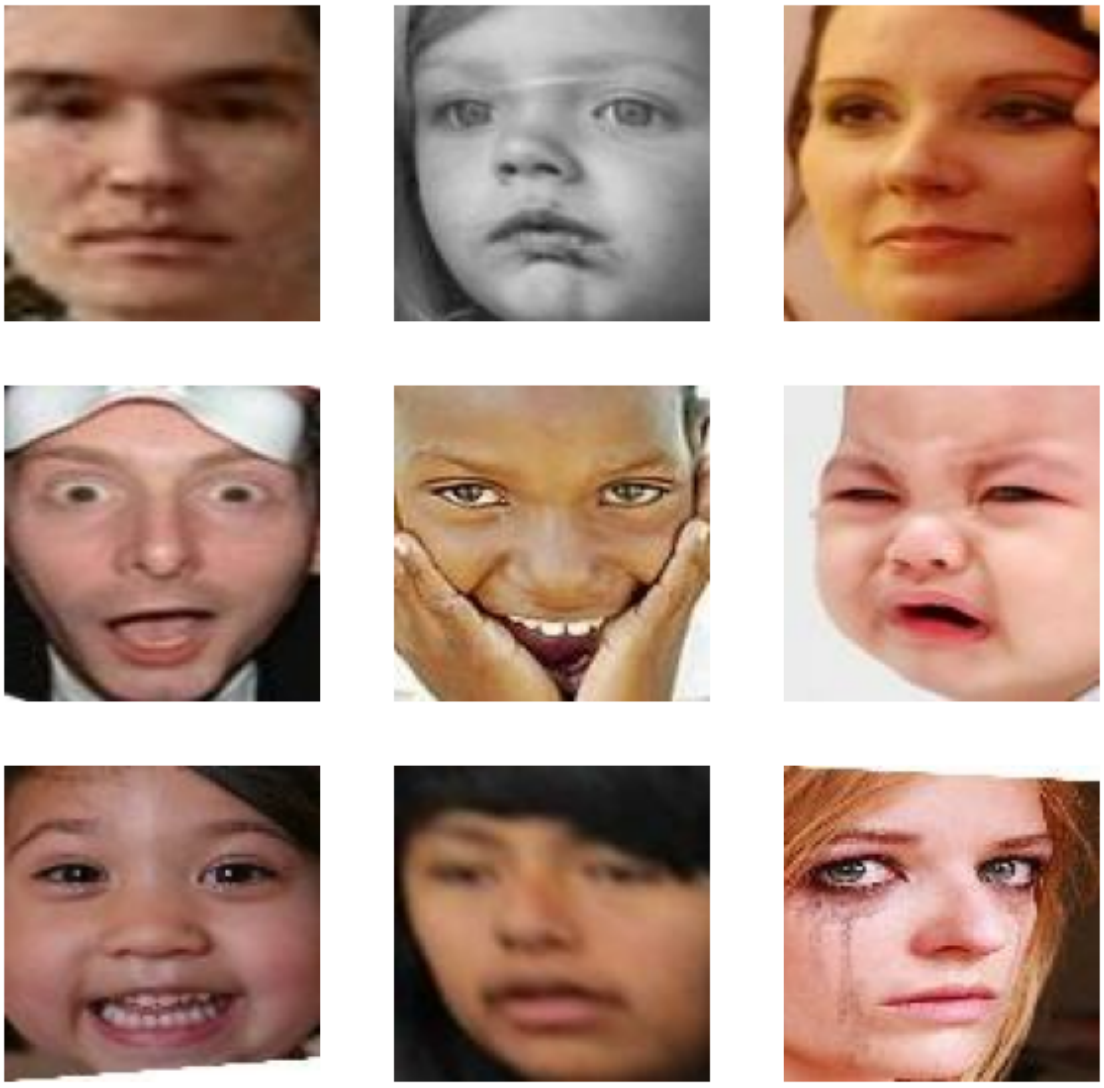
[ ]:
```
# do not run
path = '/content/gdrive/MyDrive/DL_project_2022/data/DATASETS/training_dataset/'
tf.data.experimental.save(
    train_ds, path, compression=None, shard_func=None, checkpoint_args=None
)
```

[ ]:
```
# do not run
path = '/content/gdrive/MyDrive/DL_project_2022/data/DATASETS/
 ↪validation_dataset/'
tf.data.experimental.save(
    val_ds, path, compression=None, shard_func=None, checkpoint_args=None
)
```

[ ]:
```
class_names = train_ds.class_names
print(class_names)
class_names = val_ds.class_names
print(class_names)
```

```
['1', '2', '3', '4', '5', '6', '7']
['1', '2', '3', '4', '5', '6', '7']
```

[ ]:
```
# visualize the data
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.axis("off")
```



[ ]:
```
# visualize the data
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

[ ]:
```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 100, 100, 3)
(32,)
```

http://cs231n.stanford.edu/reports/2016/pdfs/023_Report.pdf

1.1 Simple classifier with one convolution layer

[ ]:
```
num_classes = 7
#create base_model
base_model = Sequential()
#add model layers
base_model.add(Conv2D(64, kernel_size=3, activation='relu',␣
 ↪input_shape=(100,100,3)))
base_model.add(Flatten())
base_model.add(Dense(num_classes, activation='softmax'))
```

[ ]:
```
#compile model using accuracy to measure model performance
base_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
base_model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

```
Epoch 1/10
336/336 [==============================] - 2247s 7s/step - loss: 165.2318 -
accuracy: 0.3429 - val_loss: 2.5734 - val_accuracy: 0.4003
Epoch 2/10
336/336 [==============================] - 18s 52ms/step - loss: 1.3060 -
accuracy: 0.6096 - val_loss: 2.5484 - val_accuracy: 0.4523
Epoch 3/10
336/336 [==============================] - 18s 53ms/step - loss: 0.8310 -
accuracy: 0.7371 - val_loss: 3.0773 - val_accuracy: 0.4638
Epoch 4/10
336/336 [==============================] - 18s 52ms/step - loss: 0.6716 -
accuracy: 0.7969 - val_loss: 3.1328 - val_accuracy: 0.4453
Epoch 5/10
336/336 [==============================] - 18s 52ms/step - loss: 0.5778 -
accuracy: 0.8191 - val_loss: 3.8365 - val_accuracy: 0.4549
Epoch 6/10
336/336 [==============================] - 18s 52ms/step - loss: 0.5131 -
accuracy: 0.8429 - val_loss: 4.4919 - val_accuracy: 0.4516
Epoch 7/10
336/336 [==============================] - 18s 52ms/step - loss: 0.5420 -
accuracy: 0.8374 - val_loss: 4.1297 - val_accuracy: 0.4571
Epoch 8/10
336/336 [==============================] - 18s 52ms/step - loss: 0.5181 -
accuracy: 0.8498 - val_loss: 5.0929 - val_accuracy: 0.4586
Epoch 9/10
336/336 [==============================] - 18s 52ms/step - loss: 0.4467 -
accuracy: 0.8709 - val_loss: 5.7596 - val_accuracy: 0.4595
```

```
Epoch 10/10
336/336 [==============================] - 18s 52ms/step - loss: 0.3966 -
accuracy: 0.8816 - val_loss: 6.4423 - val_accuracy: 0.4525
```

[ ]:
```
<keras.callbacks.History at 0x7f2c89a07ed0>
```

[ ]:
```
# save the model
base_model.save('/content/gdrive/MyDrive/DL_project_2022/data/models/
 ↪base_model')
```

```
INFO:tensorflow:Assets written to:
/content/gdrive/MyDrive/DL_project_2022/data/models/base_model/assets
```

1.2 Five layer CNN

[ ]:
```
num_classes = 7
#create five layer model
f_layer_model1 = Sequential()
#add model layers
f_layer_model1.add(Conv2D(64, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
f_layer_model1.add(BatchNormalization())
f_layer_model1.add(ReLU())

f_layer_model1.add(Conv2D(64, kernel_size=3,strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
f_layer_model1.add(BatchNormalization())
f_layer_model1.add(ReLU())

f_layer_model1.add(Conv2D(64, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
f_layer_model1.add(BatchNormalization())
f_layer_model1.add(ReLU())

f_layer_model1.add(MaxPool2D(strides=2))
f_layer_model1.add(Dropout(rate=0.3))

f_layer_model1.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
f_layer_model1.add(BatchNormalization())
f_layer_model1.add(ReLU())

f_layer_model1.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
f_layer_model1.add(BatchNormalization())
f_layer_model1.add(ReLU())
```

```
f_layer_model1.add(MaxPool2D(strides=2))
f_layer_model1.add(Dropout(rate=0.3))

f_layer_model1.add(Flatten())
f_layer_model1.add(Dense(512, activation=None))
f_layer_model1.add(BatchNormalization())
f_layer_model1.add(ReLU())
f_layer_model1.add(Dropout(rate=0.3))
f_layer_model1.add(Dense(num_classes))
f_layer_model1.add(Softmax())
```

[ ]:
```
#compile model using accuracy to measure model performance
f_layer_model1.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

[ ]:
```
f_layer_model1.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

```
Epoch 1/10
336/336 [==============================] - 25s 69ms/step - loss: 1.2690 -
accuracy: 0.5692 - val_loss: 1.3325 - val_accuracy: 0.5575
Epoch 2/10
336/336 [==============================] - 23s 68ms/step - loss: 0.7991 -
accuracy: 0.7209 - val_loss: 0.9439 - val_accuracy: 0.6631
Epoch 3/10
336/336 [==============================] - 23s 68ms/step - loss: 0.5604 -
accuracy: 0.7994 - val_loss: 0.8770 - val_accuracy: 0.6905
Epoch 4/10
336/336 [==============================] - 23s 68ms/step - loss: 0.3793 -
accuracy: 0.8666 - val_loss: 0.9178 - val_accuracy: 0.6983
Epoch 5/10
336/336 [==============================] - 23s 68ms/step - loss: 0.2433 -
accuracy: 0.9194 - val_loss: 0.8648 - val_accuracy: 0.7279
Epoch 6/10
336/336 [==============================] - 23s 68ms/step - loss: 0.1866 -
accuracy: 0.9341 - val_loss: 0.8310 - val_accuracy: 0.7377
Epoch 7/10
336/336 [==============================] - 23s 69ms/step - loss: 0.1427 -
accuracy: 0.9527 - val_loss: 0.8241 - val_accuracy: 0.7492
Epoch 8/10
```

```
336/336 [==============================] - 23s 69ms/step - loss: 0.1214 -
accuracy: 0.9592 - val_loss: 1.0873 - val_accuracy: 0.7175
Epoch 9/10
336/336 [==============================] - 23s 69ms/step - loss: 0.0883 -
accuracy: 0.9717 - val_loss: 0.9801 - val_accuracy: 0.7474
Epoch 10/10
336/336 [==============================] - 23s 68ms/step - loss: 0.0806 -
accuracy: 0.9738 - val_loss: 1.0650 - val_accuracy: 0.7259
```

[ ]:
```
<keras.callbacks.History at 0x7f2c0e01de10>
```

[ ]:
```
f_layer_model1.save('/content/gdrive/MyDrive/DL_project_2022/data/models/
 ↪f_layer_model1')
```

```
INFO:tensorflow:Assets written to:
/content/gdrive/MyDrive/DL_project_2022/data/models/f_layer_model1/assets
```

[ ]:
```
# no batch normalization
num_classes = 7
#create five layer model
f_layer_model2 = Sequential()
#add model layers
f_layer_model2.add(Conv2D(64, kernel_size=3, strides=1, padding='same',␣
 ↪activation='relu', input_shape=(100,100,3)))

f_layer_model2.add(Conv2D(64, kernel_size=3,strides=1, padding='same',␣
 ↪activation='relu', input_shape=(100,100,3)))

f_layer_model2.add(Conv2D(64, kernel_size=3, strides=1, padding='same',␣
 ↪activation='relu', input_shape=(100,100,3)))

f_layer_model2.add(MaxPool2D(strides=2))
f_layer_model2.add(Dropout(rate=0.3))

f_layer_model2.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation='relu', input_shape=(100,100,3)))

f_layer_model2.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation='relu', input_shape=(100,100,3)))

f_layer_model2.add(MaxPool2D(strides=2))
f_layer_model2.add(Dropout(rate=0.3))

f_layer_model2.add(Flatten())
f_layer_model2.add(Dense(512, activation='relu'))
f_layer_model2.add(Dropout(rate=0.3))
f_layer_model2.add(Dense(num_classes, activation='softmax'))
```

[ ]:
```
#compile model using accuracy to measure model performance
f_layer_model2.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
f_layer_model2.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

```
Epoch 1/10
336/336 [==============================] - 41s 120ms/step - loss: 4.8820 -
accuracy: 0.4533 - val_loss: 1.1972 - val_accuracy: 0.5653
Epoch 2/10
336/336 [==============================] - 32s 95ms/step - loss: 1.1662 -
accuracy: 0.5844 - val_loss: 1.0339 - val_accuracy: 0.6296
Epoch 3/10
336/336 [==============================] - 32s 94ms/step - loss: 1.0692 -
accuracy: 0.6231 - val_loss: 1.0419 - val_accuracy: 0.6231
Epoch 4/10
336/336 [==============================] - 32s 94ms/step - loss: 0.9891 -
accuracy: 0.6500 - val_loss: 1.0015 - val_accuracy: 0.6464
Epoch 5/10
336/336 [==============================] - 32s 94ms/step - loss: 0.9241 -
accuracy: 0.6687 - val_loss: 0.9690 - val_accuracy: 0.6627
Epoch 6/10
336/336 [==============================] - 32s 94ms/step - loss: 0.8363 -
accuracy: 0.6985 - val_loss: 0.9647 - val_accuracy: 0.6564
Epoch 7/10
336/336 [==============================] - 32s 95ms/step - loss: 0.7988 -
accuracy: 0.7129 - val_loss: 0.9029 - val_accuracy: 0.6914
Epoch 8/10
336/336 [==============================] - 32s 95ms/step - loss: 0.6918 -
accuracy: 0.7550 - val_loss: 0.9126 - val_accuracy: 0.6805
Epoch 9/10
336/336 [==============================] - 32s 94ms/step - loss: 0.6294 -
accuracy: 0.7774 - val_loss: 0.9677 - val_accuracy: 0.6766
Epoch 10/10
336/336 [==============================] - 32s 95ms/step - loss: 0.5619 -
accuracy: 0.7981 - val_loss: 0.8915 - val_accuracy: 0.7062
```

[ ]:
```
<keras.callbacks.History at 0x7fd564070590>
```

[ ]:
```
f_layer_model2.save('/content/gdrive/MyDrive/DL_project_2022/data/models/
↪f_layer_model2')
```

```
INFO:tensorflow:Assets written to:
/content/gdrive/MyDrive/DL_project_2022/data/models/f_layer_model2/assets
```

```
INFO:tensorflow:Assets written to:
/content/gdrive/MyDrive/DL_project_2022/data/models/f_layer_model2/assets
```

Kernel size = 9 with batch normalization

[ ]:
```
num_classes = 7
kernel_s = 9
#create five layer model
f_layer_model1_9 = Sequential()
#add model layers
f_layer_model1_9.add(Conv2D(64, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_9.add(BatchNormalization())
f_layer_model1_9.add(ReLU())

f_layer_model1_9.add(Conv2D(64, kernel_size=kernel_s,strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
f_layer_model1_9.add(BatchNormalization())
f_layer_model1_9.add(ReLU())

f_layer_model1_9.add(Conv2D(64, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_9.add(BatchNormalization())
f_layer_model1_9.add(ReLU())

f_layer_model1_9.add(MaxPool2D(strides=2))
f_layer_model1_9.add(Dropout(rate=0.3))

f_layer_model1_9.add(Conv2D(128, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_9.add(BatchNormalization())
f_layer_model1_9.add(ReLU())

f_layer_model1_9.add(Conv2D(128, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_9.add(BatchNormalization())
f_layer_model1_9.add(ReLU())

f_layer_model1_9.add(MaxPool2D(strides=2))
f_layer_model1_9.add(Dropout(rate=0.3))

f_layer_model1_9.add(Flatten())
f_layer_model1_9.add(Dense(512, activation=None))
f_layer_model1_9.add(BatchNormalization())
f_layer_model1_9.add(ReLU())
f_layer_model1_9.add(Dropout(rate=0.3))
f_layer_model1_9.add(Dense(num_classes))
f_layer_model1_9.add(Softmax())
```

[ ]:
```
#compile model using accuracy to measure model performance
f_layer_model1_9.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
f_layer_model1_9.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

```
Epoch 1/10
336/336 [==============================] - 73s 201ms/step - loss: 1.5855 -
accuracy: 0.4516 - val_loss: 1.3128 - val_accuracy: 0.5371
Epoch 2/10
336/336 [==============================] - 65s 192ms/step - loss: 1.1302 -
accuracy: 0.6037 - val_loss: 1.1467 - val_accuracy: 0.5712
Epoch 3/10
336/336 [==============================] - 63s 187ms/step - loss: 0.8882 -
accuracy: 0.6825 - val_loss: 0.9613 - val_accuracy: 0.6459
Epoch 4/10
336/336 [==============================] - 64s 190ms/step - loss: 0.6877 -
accuracy: 0.7566 - val_loss: 0.8218 - val_accuracy: 0.7122
Epoch 5/10
336/336 [==============================] - 64s 189ms/step - loss: 0.5494 -
accuracy: 0.8031 - val_loss: 0.8090 - val_accuracy: 0.7216
Epoch 6/10
336/336 [==============================] - 64s 190ms/step - loss: 0.4329 -
accuracy: 0.8466 - val_loss: 0.8135 - val_accuracy: 0.7257
Epoch 7/10
336/336 [==============================] - 64s 190ms/step - loss: 0.3190 -
accuracy: 0.8877 - val_loss: 0.8300 - val_accuracy: 0.7342
Epoch 8/10
336/336 [==============================] - 65s 193ms/step - loss: 0.2352 -
accuracy: 0.9208 - val_loss: 0.9601 - val_accuracy: 0.7340
Epoch 9/10
336/336 [==============================] - 66s 195ms/step - loss: 0.1990 -
accuracy: 0.9316 - val_loss: 0.9024 - val_accuracy: 0.7398
Epoch 10/10
336/336 [==============================] - 65s 194ms/step - loss: 0.1550 -
accuracy: 0.9492 - val_loss: 0.9681 - val_accuracy: 0.7272
```

[ ]:
```
<keras.callbacks.History at 0x7f02ce057d50>
```

[ ]:
```
f_layer_model1_9.save('/content/gdrive/MyDrive/DL_project_2022/data/models/
↪f_layer_model1_9')
```

```
INFO:tensorflow:Assets written to:
/content/gdrive/MyDrive/DL_project_2022/data/models/f_layer_model1_9/assets
```

kernel size = 20

```
[ ]: num_classes = 7
kernel_s = 20
#create five layer model
f_layer_model1_20 = Sequential()
#add model layers
f_layer_model1_20.add(Conv2D(64, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_20.add(BatchNormalization())
f_layer_model1_20.add(ReLU())

f_layer_model1_20.add(Conv2D(64, kernel_size=kernel_s,strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_20.add(BatchNormalization())
f_layer_model1_20.add(ReLU())

f_layer_model1_20.add(Conv2D(64, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_20.add(BatchNormalization())
f_layer_model1_20.add(ReLU())

f_layer_model1_20.add(MaxPool2D(strides=2))
f_layer_model1_20.add(Dropout(rate=0.3))

f_layer_model1_20.add(Conv2D(128, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_20.add(BatchNormalization())
f_layer_model1_20.add(ReLU())

f_layer_model1_20.add(Conv2D(128, kernel_size=kernel_s, strides=1,␣
 ↪padding='same', activation=None, input_shape=(100,100,3)))
f_layer_model1_20.add(BatchNormalization())
f_layer_model1_20.add(ReLU())

f_layer_model1_20.add(MaxPool2D(strides=2))
f_layer_model1_20.add(Dropout(rate=0.3))

f_layer_model1_20.add(Flatten())
f_layer_model1_20.add(Dense(512, activation=None))
f_layer_model1_20.add(BatchNormalization())
f_layer_model1_20.add(ReLU())
f_layer_model1_20.add(Dropout(rate=0.3))
f_layer_model1_20.add(Dense(num_classes))
f_layer_model1_20.add(Softmax())
```

[ ]:
```
#compile model using accuracy to measure model performance
f_layer_model1_20.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
f_layer_model1_20.fit(
  train_ds,
  validation_data=val_ds,
  epochs=15
)
```

```
Epoch 1/15
336/336 [==============================] - 2388s 7s/step - loss: 1.6902 -
accuracy: 0.3970 - val_loss: 2.5928 - val_accuracy: 0.1715
Epoch 2/15
336/336 [==============================] - 58s 172ms/step - loss: 1.3910 -
accuracy: 0.5080 - val_loss: 2.5633 - val_accuracy: 0.3165
Epoch 3/15
336/336 [==============================] - 58s 172ms/step - loss: 1.1739 -
accuracy: 0.5824 - val_loss: 1.1084 - val_accuracy: 0.6036
Epoch 4/15
336/336 [==============================] - 58s 171ms/step - loss: 1.0395 -
accuracy: 0.6282 - val_loss: 1.0249 - val_accuracy: 0.6323
Epoch 5/15
336/336 [==============================] - 58s 171ms/step - loss: 0.9100 -
accuracy: 0.6759 - val_loss: 0.9412 - val_accuracy: 0.6664
Epoch 6/15
336/336 [==============================] - 58s 171ms/step - loss: 0.8086 -
accuracy: 0.7089 - val_loss: 1.0022 - val_accuracy: 0.6512
Epoch 7/15
336/336 [==============================] - 58s 173ms/step - loss: 0.7131 -
accuracy: 0.7445 - val_loss: 0.8863 - val_accuracy: 0.6877
Epoch 8/15
336/336 [==============================] - 58s 171ms/step - loss: 0.6196 -
accuracy: 0.7799 - val_loss: 0.9224 - val_accuracy: 0.6888
Epoch 9/15
336/336 [==============================] - 58s 171ms/step - loss: 0.5343 -
accuracy: 0.8044 - val_loss: 1.0782 - val_accuracy: 0.6505
Epoch 10/15
336/336 [==============================] - 58s 171ms/step - loss: 0.4473 -
accuracy: 0.8416 - val_loss: 1.0731 - val_accuracy: 0.6696
Epoch 11/15
336/336 [==============================] - 59s 173ms/step - loss: 0.3705 -
accuracy: 0.8670 - val_loss: 0.9604 - val_accuracy: 0.6909
Epoch 12/15
336/336 [==============================] - 58s 172ms/step - loss: 0.3036 -
accuracy: 0.8937 - val_loss: 1.0191 - val_accuracy: 0.6918
Epoch 13/15
```

```
336/336 [==============================] - 58s 173ms/step - loss: 0.2449 -
accuracy: 0.9126 - val_loss: 1.0451 - val_accuracy: 0.7003
Epoch 14/15
336/336 [==============================] - 58s 172ms/step - loss: 0.1926 -
accuracy: 0.9331 - val_loss: 1.1049 - val_accuracy: 0.7146
Epoch 15/15
336/336 [==============================] - 58s 172ms/step - loss: 0.1775 -
accuracy: 0.9383 - val_loss: 1.1564 - val_accuracy: 0.7083
```

[ ]:
```
<keras.callbacks.History at 0x7fa610854950>
```

[ ]:
```
sess = tf.compat.v1.Session(config=tf.compat.v1.
 ↪ConfigProto(log_device_placement=True))
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla
P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
```

[ ]:
```
tf.config.list_physical_devices('GPU')
```

[ ]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

1.3 Deeper neural networks

1.3.1 Model 1

[4]:
```
num_classes = 7

deep_model1 = Sequential()

# stage 1 - convolution with filter 1, batch normalization, ReLu
# At this level the kernel size is 5 (we want to take more information)
deep_model1.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())

# stage 2 back to back convolution layers with normalization and ReLu followed␣
 ↪by maxpooling (stride = 1)
#                 (again, taking max amount of information) and dropout layer.
  # first convolution
deep_model1.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
  # second convolution
```

```
deep_model1.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
  # max pooling and dropout
deep_model1.add(MaxPool2D(strides = 1))
deep_model1.add(Dropout(rate = 0.3))

# stage 3 - similar to stage 2 but with a different convolution kernel and␣
 ↪number of neurons in the convolution layer
  # first convolution ------ math.floor((deep_model1.input_shape - (2,2)) / 1)␣
 ↪+ 1
deep_model1.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(99,99,3)))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
  # second convolution
deep_model1.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(99,99,3)))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
  # max pooling and dropout
deep_model1.add(MaxPool2D(strides = 1))
deep_model1.add(Dropout(rate = 0.3))

# stage 4 - Fully connected layers, batch normalization, ReLu, dropout
deep_model1.add(Flatten())
  # first dense layer
deep_model1.add(Dense(512, activation=None))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
deep_model1.add(Dropout(rate=0.3))
  # second dense layer
deep_model1.add(Dense(256, activation=None))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
deep_model1.add(Dropout(rate=0.3))
  # Third dense layer
deep_model1.add(Dense(128, activation=None))
deep_model1.add(BatchNormalization())
deep_model1.add(ReLU())
deep_model1.add(Dropout(rate=0.3))

# stage 5 - output
deep_model1.add(Dense(num_classes, activation='softmax'))
```

[6]:
```
#compile model using accuracy to measure model performance
deep_model1.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
deep_model1.input_shape
```

[ ]:
```
(None, 100, 100, 3)
```

[ ]:
```
deep_model1.output_shape
```

[ ]:
```
(None, 7)
```

[ ]:
```
deep_model1.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

[7]:
```
deep_model1.fit(
  train_ds,
  validation_data=val_ds,
  epochs=20
)
```

```
Epoch 1/20
336/336 [==============================] - 1382s 4s/step - loss: 1.4713 -
accuracy: 0.4787 - val_loss: 1.1525 - val_accuracy: 0.5910
Epoch 2/20
336/336 [==============================] - 61s 181ms/step - loss: 0.9778 -
accuracy: 0.6597 - val_loss: 1.2912 - val_accuracy: 0.5477
Epoch 3/20
336/336 [==============================] - 61s 181ms/step - loss: 0.7739 -
accuracy: 0.7262 - val_loss: 1.0554 - val_accuracy: 0.6266
Epoch 4/20
336/336 [==============================] - 61s 181ms/step - loss: 0.6255 -
accuracy: 0.7796 - val_loss: 0.9787 - val_accuracy: 0.6618
Epoch 5/20
336/336 [==============================] - 61s 181ms/step - loss: 0.4740 -
accuracy: 0.8320 - val_loss: 1.0329 - val_accuracy: 0.6627
Epoch 6/20
336/336 [==============================] - 61s 181ms/step - loss: 0.3509 -
accuracy: 0.8766 - val_loss: 1.0784 - val_accuracy: 0.6766
Epoch 7/20
336/336 [==============================] - 61s 181ms/step - loss: 0.2662 -
accuracy: 0.9072 - val_loss: 1.2055 - val_accuracy: 0.6849
Epoch 8/20
336/336 [==============================] - 61s 181ms/step - loss: 0.2158 -
```

```
accuracy: 0.9251 - val_loss: 1.0013 - val_accuracy: 0.7246
Epoch 9/20
336/336 [==============================] - 61s 181ms/step - loss: 0.1852 -
accuracy: 0.9364 - val_loss: 1.0385 - val_accuracy: 0.7268
Epoch 10/20
336/336 [==============================] - 61s 181ms/step - loss: 0.1555 -
accuracy: 0.9468 - val_loss: 1.0912 - val_accuracy: 0.7105
Epoch 11/20
336/336 [==============================] - 61s 182ms/step - loss: 0.1245 -
accuracy: 0.9590 - val_loss: 1.1869 - val_accuracy: 0.7233
Epoch 12/20
336/336 [==============================] - 62s 182ms/step - loss: 0.1287 -
accuracy: 0.9547 - val_loss: 1.2341 - val_accuracy: 0.7205
Epoch 13/20
336/336 [==============================] - 61s 182ms/step - loss: 0.1069 -
accuracy: 0.9638 - val_loss: 1.1441 - val_accuracy: 0.7370
Epoch 14/20
336/336 [==============================] - 61s 181ms/step - loss: 0.0959 -
accuracy: 0.9685 - val_loss: 1.1472 - val_accuracy: 0.7351
Epoch 15/20
336/336 [==============================] - 61s 181ms/step - loss: 0.1031 -
accuracy: 0.9646 - val_loss: 1.1060 - val_accuracy: 0.7381
Epoch 16/20
336/336 [==============================] - 61s 181ms/step - loss: 0.0846 -
accuracy: 0.9709 - val_loss: 1.1954 - val_accuracy: 0.7116
Epoch 17/20
336/336 [==============================] - 61s 181ms/step - loss: 0.0914 -
accuracy: 0.9716 - val_loss: 1.1369 - val_accuracy: 0.7351
Epoch 18/20
336/336 [==============================] - 61s 181ms/step - loss: 0.0713 -
accuracy: 0.9764 - val_loss: 1.3006 - val_accuracy: 0.7218
Epoch 19/20
336/336 [==============================] - 61s 181ms/step - loss: 0.0845 -
accuracy: 0.9727 - val_loss: 1.1902 - val_accuracy: 0.7340
Epoch 20/20
336/336 [==============================] - 61s 181ms/step - loss: 0.0672 -
accuracy: 0.9778 - val_loss: 1.2516 - val_accuracy: 0.7353
```

[7]: <keras.callbacks.History at 0x7fadb90e4990>

[ ]:
```
deep_model1.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

```
Epoch 1/10
336/336 [==============================] - 2231s 7s/step - loss: 1.4563 -
```

```
accuracy: 0.4857 - val_loss: 1.3794 - val_accuracy: 0.4949
Epoch 2/10
336/336 [==============================] - 120s 355ms/step - loss: 0.9586 -
accuracy: 0.6624 - val_loss: 1.8998 - val_accuracy: 0.3188
Epoch 3/10
336/336 [==============================] - 123s 364ms/step - loss: 0.7644 -
accuracy: 0.7352 - val_loss: 0.9969 - val_accuracy: 0.6451
Epoch 4/10
336/336 [==============================] - 123s 364ms/step - loss: 0.6034 -
accuracy: 0.7875 - val_loss: 0.7907 - val_accuracy: 0.7235
Epoch 5/10
336/336 [==============================] - 122s 363ms/step - loss: 0.4582 -
accuracy: 0.8368 - val_loss: 0.8729 - val_accuracy: 0.7185
Epoch 6/10
336/336 [==============================] - 122s 363ms/step - loss: 0.3438 -
accuracy: 0.8802 - val_loss: 1.0313 - val_accuracy: 0.6962
Epoch 7/10
336/336 [==============================] - 122s 362ms/step - loss: 0.2650 -
accuracy: 0.9090 - val_loss: 1.0163 - val_accuracy: 0.7125
Epoch 8/10
336/336 [==============================] - 122s 363ms/step - loss: 0.2057 -
accuracy: 0.9306 - val_loss: 1.1917 - val_accuracy: 0.6942
Epoch 9/10
336/336 [==============================] - 123s 364ms/step - loss: 0.1779 -
accuracy: 0.9383 - val_loss: 1.2157 - val_accuracy: 0.7079
Epoch 10/10
336/336 [==============================] - 122s 363ms/step - loss: 0.1504 -
accuracy: 0.9493 - val_loss: 1.0768 - val_accuracy: 0.7424
```

[ ]:
```
<keras.callbacks.History at 0x7fea0f196a50>
```

[ ]:
```
# save the model to google drive
deep_model1.save('/content/gdrive/MyDrive/DL_project_2022/data/models/
 ↪deep_model1') # i am not sure that the model was saved correctly
```

```
INFO:tensorflow:Assets written to:
/content/gdrive/MyDrive/DL_project_2022/data/models/deep_model1/assets
```

1.3.2 Model 2

2 convolution layers with filter 1 (stage 2) and 2 convolution layers with filter 2 (stage 3)

[ ]:
```
num_classes = 7

deep_model2 = Sequential()

# stage 1 - convolution with filter 1, batch normalization, ReLu
# At this level the kernel size is 5 (we want to take more information)
```

```
deep_model2.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())

# stage 2.1 - 2 back to back convolution layers with normalization and ReLu␣
 ↪followed by maxpooling (stride = 1)
#                 (again, taking max amount of information) and dropout layer.
  # first convolution
deep_model2.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # second convolution
deep_model2.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(100,100,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # max pooling and dropout
deep_model2.add(MaxPool2D(strides = 1))
deep_model2.add(Dropout(rate = 0.3))

# stage 2.2 - 2 back to back convolution layers with normalization and ReLu␣
 ↪followed by maxpooling (stride = 1)
#                 (again, taking max amount of information) and dropout layer.
  # first convolution
deep_model2.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(99,99,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # second convolution
deep_model2.add(Conv2D(64, kernel_size=5, strides=1, padding='same',␣
 ↪activation=None, input_shape=(99,99,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # max pooling and dropout
deep_model2.add(MaxPool2D(strides = 1))
deep_model2.add(Dropout(rate = 0.3))

# stage 3.1 - similar to stage 2 but with a different convolution kernel and␣
 ↪number of neurons in the convolution layer
  # first convolution ------ math.floor((deep_model1.input_shape - (2,2)) / 1)␣
 ↪+ 1
deep_model2.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(98,98,3)))
deep_model2.add(BatchNormalization())
```

```
deep_model2.add(ReLU())
  # second convolution
deep_model2.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(98,98,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # max pooling and dropout
deep_model2.add(MaxPool2D(strides = 1))
deep_model2.add(Dropout(rate = 0.3))

# stage 3.2 - similar to stage 2 but with a different convolution kernel and␣
 ↪number of neurons in the convolution layer
  # first convolution ------ math.floor((deep_model1.input_shape - (2,2)) / 1)␣
 ↪+ 1
deep_model2.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(97,97,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # second convolution
deep_model2.add(Conv2D(128, kernel_size=3, strides=1, padding='same',␣
 ↪activation=None, input_shape=(97,97,3)))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
  # max pooling and dropout
deep_model2.add(MaxPool2D(strides = 1))
deep_model2.add(Dropout(rate = 0.3))

# stage 4 - Fully connected layers, batch normalization, ReLu, dropout
deep_model2.add(Flatten())
  # first dense layer
deep_model2.add(Dense(512, activation=None))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
deep_model2.add(Dropout(rate=0.3))
  # second dense layer
deep_model2.add(Dense(256, activation=None))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
deep_model2.add(Dropout(rate=0.3))
  # Third dense layer
deep_model2.add(Dense(128, activation=None))
deep_model2.add(BatchNormalization())
deep_model2.add(ReLU())
deep_model2.add(Dropout(rate=0.3))

# stage 5 - output
deep_model2.add(Dense(num_classes, activation='softmax'))
```

[ ]:
```
#compile model using accuracy to measure model performance
deep_model2.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
deep_model2.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

```
Epoch 1/10
336/336 [==============================] - 200s 561ms/step - loss: 1.6779 -
accuracy: 0.3921 - val_loss: 1.5724 - val_accuracy: 0.4588
Epoch 2/10
336/336 [==============================] - 189s 561ms/step - loss: 1.1680 -
accuracy: 0.5806 - val_loss: 2.2122 - val_accuracy: 0.3458
Epoch 3/10
336/336 [==============================] - 190s 565ms/step - loss: 0.9342 -
accuracy: 0.6687 - val_loss: 0.9318 - val_accuracy: 0.6507
Epoch 4/10
336/336 [==============================] - 190s 566ms/step - loss: 0.7903 -
accuracy: 0.7214 - val_loss: 1.0332 - val_accuracy: 0.6157
Epoch 5/10
336/336 [==============================] - 190s 564ms/step - loss: 0.6649 -
accuracy: 0.7644 - val_loss: 0.8957 - val_accuracy: 0.6883
Epoch 6/10
336/336 [==============================] - 190s 565ms/step - loss: 0.5249 -
accuracy: 0.8120 - val_loss: 0.8589 - val_accuracy: 0.7153
Epoch 7/10
336/336 [==============================] - 191s 567ms/step - loss: 0.4141 -
accuracy: 0.8515 - val_loss: 1.1491 - val_accuracy: 0.6562
Epoch 8/10
336/336 [==============================] - 190s 566ms/step - loss: 0.3147 -
accuracy: 0.8897 - val_loss: 1.0605 - val_accuracy: 0.6988
Epoch 9/10
336/336 [==============================] - 190s 565ms/step - loss: 0.2515 -
accuracy: 0.9127 - val_loss: 1.0877 - val_accuracy: 0.7033
Epoch 10/10
336/336 [==============================] - 191s 566ms/step - loss: 0.2023 -
accuracy: 0.9305 - val_loss: 1.2860 - val_accuracy: 0.6688
```

[ ]:
```
<keras.callbacks.History at 0x7f9d104d6750>
```

[ ]:
```
# save the model to google drive
deep_model2.save('/content/gdrive/MyDrive/DL_project_2022/data/models/
 ↪deep_model2')
```

```
INFO:tensorflow:Assets written to:
```

```
/content/gdrive/MyDrive/DL_project_2022/data/models/deep_model2/assets
```

we see that the model ovefits the training data and the validation accuracy decreased. So we will stick with the previous model

[ ]:
```
!free -h --si | awk  '/Mem:/{print $2}'
```

```
13G
```

[ ]:
```
#hard disk space that we can use
!df -h / | awk '{print $4}'
```

```
Avail
127G
```