

DL EXAM

Sentiment Analysis from text and Images

1st Step:

The First step is to load the data which is straightforward with the help of Kaggle API

```
First thing to do is to download kaggle dataset using kaggle API ,So I need to authenticate my self by selecting the json file kaggle.json that contains my credential data .
```

```
[ ] from google.colab import files
    files.upload()
```

kaggle.json(application/json) - 66 bytes, last modified: n/a - 100% done
Saving kaggle.json to kaggle (1).json
{'kaggle.json': b'{"username":"bahriaymen","key":"a13346c4c42326125bedf0153e4a7d20"}'}

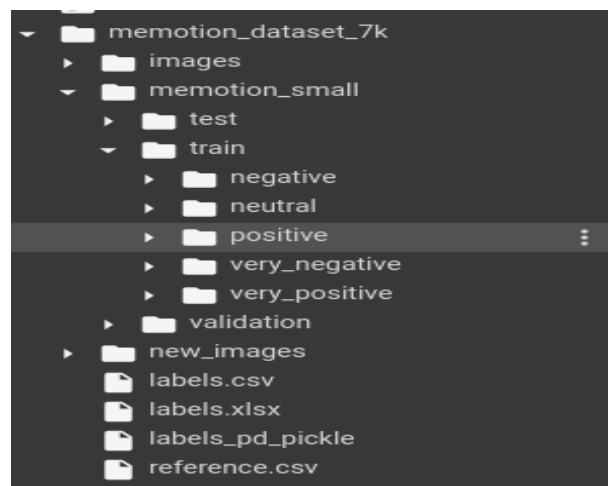
next step I need to copy the json file to the kaggle directory so I need to create a root directory using the command **mkdir ~/.kaggle** and I can also add a command to make the file readable only by me by changing the permission using **chmod ... 600** is me (the current user)

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

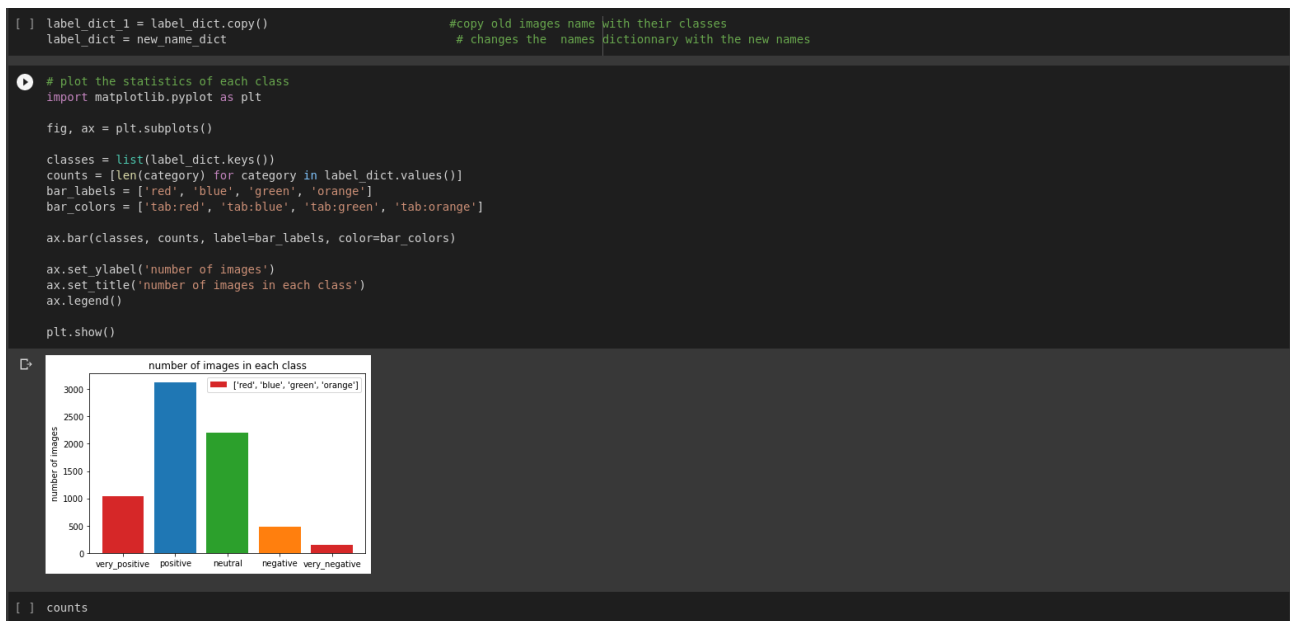
mkdir: cannot create directory '/root/.kaggle': File exists

2nd Step :

After Unzipping the data an important step (that I see it important is to split the images of each class rename them according to their class and move them to subdirectories of train ,validation and test ,this step is the most important step because it helps us use the dataset object later by calling `image_dataset_from_directory` ,which in turns will generate batches of data from each class ,reshaped to our desired shape ,i.e (300,300) and their corresponding labels



after running the code cell we end up with those directories ,the dataset object used later will list all subdirectories lets say for training and returns an iterable object that I will feed it into the model



the data distribution is very unbalanced ,classes negative and very negative contains a small number of samples ,so the model will be not able to generalize on those tow classes and will certainly overfits on the others .I could use data augmentation to increase the number of samples of those tow classes but I won't use it since visually speaking ,images of the same class doesn't have sheared patterns between them so this step could be useless ,it could be useful but I won't include it here

3rd Step :

I can now prepare a basic model architecture ,since I am dealing with images ,CNN models will be the best choice because as opposed to dense layers ,they learn features from their position in images and can recognize them anywhere later when testing the model

```
Double click (or enter) to edit
# now we will use keras functional API to create a CNN model at first,using functional API will allow us to alter the model later by adding layers and enables me to use multi_inp

from tensorflow import keras
from keras import layers
inputs =keras.Input(shape = (300,300,3))
x = layers.Rescaling(1./255)(inputs) # input shape (300,300) was chosen at random
x = layers.Conv2D(filters = 32,kernel_size=3,activation="relu")(x) # rescaling inputs by deviding each pixel by 255
x = layers.MaxPooling2D(pool_size=2)(x) #downsampling the input ,deviding the input shape by 2
x = layers.Conv2D(filters = 64,kernel_size = 3,activation = "relu")(x)
x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.Conv2D(filters=128,kernel_size = 3,activation ="relu" )(x)
x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.Conv2D(filters=256,kernel_size = 3,activation ="relu" )(x)
x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.Conv2D(filters=256,kernel_size = 3,activation ="relu" )(x)
x = layers.Flatten()(x)

outputs = layers.Dense(5,activation = "softmax")(x)
model = keras.Model(inputs = inputs,outputs = outputs )

model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 300, 300, 3)]	0
rescaling (Rescaling)	(None, 300, 300, 3)	0
conv2d (Conv2D)	(None, 298, 298, 32)	896
max_pooling2d (MaxPooling2D)	(None, 149, 149, 32)	0
conv2d_1 (Conv2D)	(None, 147, 147, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 73, 73, 64)	0

The MaxPooling layers used will downsample the feature maps by halving them(deviding by 2 the dimension of their outputs it consists of extracting windows from the input feature maps(output of convolution layers) and outputting the max value of each channel.

4th Step :

I need to split the data between training ,test,and validation
the number of samples chosen for each call is much random because the negative and very negative classes contains very few examples ,we tried to split them by using most of them on training (200 negative ,100 very negative) and the rest to test and validation

```
!rm -r /content/memotio dataset 7k/memotio small #Just a terminal command to remove recursively my small dataset if its already created
import os,shutil,pathlib

original_dir = pathlib.Path("/content/memotio dataset 7k/new images")
new_base_dir = pathlib.Path("/content/memotio dataset 7k/memotio small")

label_dict = new_name_dict

#since data is not split equivalently between classes we will split them manually not following any rule between validation , train and test
l_train = {}
train_positive = label_dict["positive"][:2000]
train_very_positive = label_dict["very_positive"][:600]
train_neutral = label_dict["neutral"][:1000]
train_negative = label_dict["negative"][:200]
train_very_negative = label_dict["very_negative"][:100]
l_train["positive"],l_train["very_positive"],l_train["neutral"],l_train["negative"],l_train["very_negative"] = train_positive,train_very_positive, train_neutral, train_negative, t

l_test ={}
test_positive = label_dict["positive"][2000:2500]
test_very_positive = label_dict["very_positive"][600:800]
test_neutral = label_dict["neutral"][1000:1800]
test_negative = label_dict["negative"][200:400]
test_very_negative = label_dict["very_negative"][100:125]

l_test["positive"],l_test["very_positive"],l_test["neutral"],l_test["negative"],l_test["very_negative"] = test_positive,test_very_positive, test_neutral, test_negative, test_very_

l_validation = {}
validation_positive = label_dict["positive"][2500:]
validation_very_positive = label_dict["very_positive"][800:]
validation_neutral = label_dict["neutral"][1800:]
validation_negative = label_dict["negative"][400:]
validation_very_negative = label_dict["very_negative"][125:]

l_validation["positive"],l_validation["very_positive"],l_validation["neutral"],l_validation["negative"],l_validation["very_negative"] = validation_positive,validation_very_positiv
```

Now that train, validation and test contain the chosen number of images for each class we need to create subdirectories and save data for each set accordingly, this is what the function `make_subset` does :

```
def make_subset(subset_name, image_names):
    """ This function will be used to create train validation and test subdirectories ,each one of them of the 5 classes
        each class contains the number of images that was specified above """
    for category in label_dict.keys():
        dir = new_base_dir / subset_name / category          #path to our memotion_small/ (train,test,or validation) /(psotive ,negative ,very_positive ...)
        os.makedirs(dir)

        for fname in image_names[category]:
            shutil.copyfile(src=original_dir / fname, dst=dir / fname)    #loop over the images that belongs to each class ,and move them to the appropriate directory
```

we call the function and give it as parameters the name of our 3 directories (train, validation, test) and the dictionary that contains the data for each set, for the train set for example, it will create a subdirectory inside `memotion_small` called train, and 5 sub_directories with the name of the five classes, for each class it copies the images according to their names drawn from the dictionary to the specified directory holding that class name.

5th Step :

Since our directories are ready and each one contains the correct data, we can now use the dataset object by using the **`image_dataset_from_directory`** class as discussed before it will list all the subdirectories assuming that each (train, validation, test) contains correct subdirectories with correct data, it will then assign classes to those subdirectories, ordered list of integers as the subdirectories order.

```
#image dataset from directory will list subdirectories and assume that each directory belongs to a class ,so as directroies are listed as
#(negative,neutral,positive,very_negative,very_positive) classes will be (0,1,2,3,4) accordingly

from tensorflow.keras.utils import image_dataset_from_directory

shape = (300,300)

train_dataset = image_dataset_from_directory(new_base_dir / "train",
                                             image_size = shape, batch_size = 32)    # train_dataset will be an iterable that holds the training data as 32 batches with (300,300) shape ,and their corresponding lab

validation_dataset = image_dataset_from_directory(new_base_dir / "validation",
                                                  image_size = shape, batch_size = 32)

test_dataset = image_dataset_from_directory(new_base_dir / "test",
                                            image_size = shape, batch_size = 32)
```

We can check for instance the shape of tensors that the `image_dataset_from_directory` returns, it returns an iterable that we can loop on its elements.

```
[15] #to check the data shape in our train dataset object
for data_batch, labels_batch in train_dataset:
    print(data_batch.shape, labels_batch, labels_batch.shape)
    image = data_batch[0]
    image_label = labels_batch[0]
    break

(32, 300, 300, 3) tf.Tensor([1 4 2 2 2 0 2 4 4 2 2 1 1 4 1 2 2 2 1 2 2 2 1 2 4 2 0 2 1 4], shape=(32,), dtype=int32) (32,)
```

```
#lets see a reshaped image ,we converted the image to an unsigned int array so it can be visualised properly with plt.imshow()
import matplotlib.pyplot as plt
plt.imshow(image.numpy().astype("uint8"))
plt.axis("off")
```



The shape of the data is (32,300,300,3) which is a batch of 32 samples of (300,300) 3 channel images

6th Step :

Now we are ready to fit our model ,at first sight we can only fit the data ,see if the model fit and try different number of epochs and batch sizes ,to remove the need of waiting till the end of fit each time to check the accuracy and loss on train and validation we can simply use keras callbacks ,EarlyStopping ,specify which metric to monitor and the number of epochs where to interrupt the fit if there are no change or improvements .

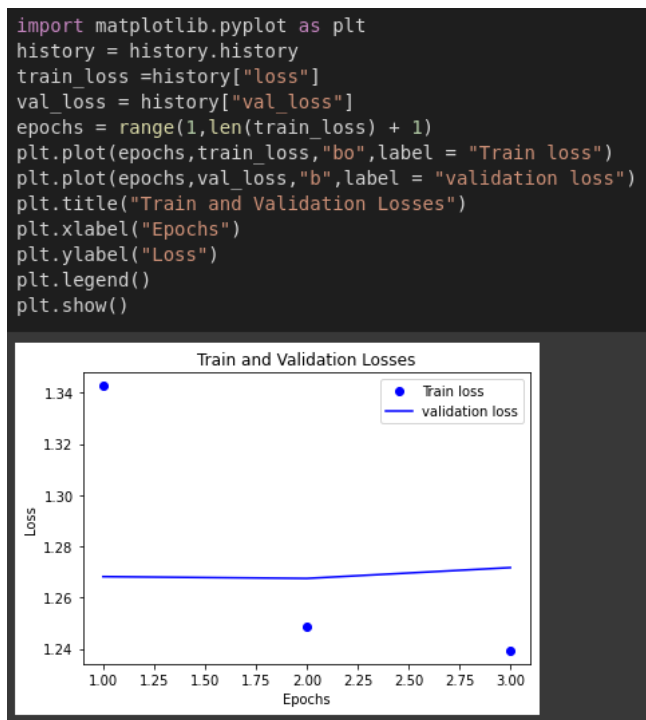
We can also use ModelCheckpoint callback to save the best version of the model obtained .

```
#we will set a base line of 50% accuracy on training ,and then try to make the model overfit
callback_list = [keras.callbacks.EarlyStopping(monitor="val_accuracy",patience=2),
keras.callbacks.ModelCheckpoint(filepath="checkpoint.keras",monitor="val_accuracy",save_best_only=True)] #EarlyStopping callback will allow the model to stop fitting as soon as the validation acc
history = model.fit(train_dataset,epochs=30,validation_data=validation_dataset,callbacks=callback_list) #history is a dictionary that will hold the values of accuracy and loss for each epoch

Epoch 1/30
122/122 [=====] - 38s 225ms/step - loss: 1.3426 - accuracy: 0.4960 - val_loss: 1.2682 - val_accuracy: 0.4587
Epoch 2/30
122/122 [=====] - 26s 208ms/step - loss: 1.2488 - accuracy: 0.5127 - val_loss: 1.2675 - val_accuracy: 0.4587
Epoch 3/30
122/122 [=====] - 26s 208ms/step - loss: 1.2393 - accuracy: 0.5127 - val_loss: 1.2717 - val_accuracy: 0.4587
```

7th Step :

To be able to plot metrics versus epochs the fit method returns an object that has an attribute called history that records metrics during each epoch ,we can use it to plot the accuracy and loss for both train and validation .



Now that the accuracy of train is very low ,we can think of increasing the number of layers ,which means giving the model more degrees of freedom ,adding layers will allow the model to learn more complex representation on the way and can help it (a big theoretical maybe) learn a better representation of the data ,this will never happen mainly because looking at the images can say that data is very noisy meaning that each class related to an image doesn't really hold any clear visual representation with respect to other images ,so likely the model won't be able to learn any good representations of those data ,and since images contains text ,where text is more related to the classes ,maybe we can use a multi input model and make it learn to link images to text ,this what we tried but didn't work due to the labels of both classes ,I try to make it work nevertheless .

We can list the layers of our model

```
model.layers
[<keras.engine.input_layer.InputLayer at 0x7f4faab316a0>,
 <keras.layers.preprocessing.image_preprocessing.Rescaling at 0x7f4faab31640>,
 <keras.layers.convolutional.conv2d.Conv2D at 0x7f4f483d5340>,
 <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x7f4f483d55b0>,
 <keras.layers.convolutional.conv2d.Conv2D at 0x7f4f483d51f0>,
 <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x7f4f483d5610>,
 <keras.layers.convolutional.conv2d.Conv2D at 0x7f4f483d5ca0>,
 <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x7f4f47affc70>,
 <keras.layers.convolutional.conv2d.Conv2D at 0x7f4f47affeb0>,
 <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x7f4f47aff760>,
 <keras.layers.convolutional.conv2d.Conv2D at 0x7f4f47afff10>,
 <keras.layers.resizing.flatten.Flatten at 0x7f4f47affb20>,
 <keras.layers.core.dense.Dense at 0x7f4f47aded60>]
```

We can see that the index of the layer just before flatten ,is 10
so we can use the output of this layer and use it as the input for the new layers that we will add to the model ,here 2 conv2D layers with a maxpooling in between ,since maxpooling halves the shape by 2 ,we just used one layer ,so we don't need to change the input shape .

```
x = model.layers[10].output #store the output of that 10th layer

# add 2 conv2D and a MAXpooling layer

x = layers.Conv2D(filters=256,kernel_size = 3,activation = "relu" )(x)
x = layers.MaxPooling2D(pool_size = 2)(x)
x = layers.Conv2D(filters=256,kernel_size = 3,activation = "relu" )(x)
x = layers.Flatten()(x)
outputs = layers.Dense(5,activation = "softmax")(x)

model_2 =keras.Model(inputs = inputs,outputs = outputs ) # make a new model

model_2.summary()
```

8th Step :

we compile and fit the model just as before

```
[22] model_2.compile(optimizer="rmsprop",loss="sparse_categorical_crossentropy",metrics = ["accuracy"])

shape = (300,300)

train_dataset = image_dataset_from_directory(new_base_dir / "train",
                                             image_size = shape,batch_size = 32)

validation_dataset = image_dataset_from_directory(new_base_dir / "validation",
                                                  image_size = shape,batch_size = 32)
test_dataset = image_dataset_from_directory(new_base_dir / "test",
                                             image_size = shape,batch_size = 32)

Found 3899 files belonging to 5 classes.
Found 1367 files belonging to 5 classes.
Found 1725 files belonging to 5 classes.

[24] callback_list = [keras.callbacks.EarlyStopping(monitor="val_accuracy",patience=2),
                    keras.callbacks.ModelCheckpoint(filepath="checkpoint.keras",monitor="val_accuracy",save_best_only=True)]
history = model_2.fit(train_dataset,epochs=30,validation_data=validation_dataset,callbacks=callback_list)

Epoch 1/30
122/122 [=====] - 28s 215ms/step - loss: 1.2681 - accuracy: 0.5106 - val_loss: 1.2722 - val_accuracy: 0.4587
Epoch 2/30
122/122 [=====] - 27s 212ms/step - loss: 1.2460 - accuracy: 0.5127 - val_loss: 1.2750 - val_accuracy: 0.4587
Epoch 3/30
122/122 [=====] - 28s 222ms/step - loss: 1.2386 - accuracy: 0.5127 - val_loss: 1.2674 - val_accuracy: 0.4587

there are no improvements ,the model barely fits the data,there are no clear patterns between the data and the
labels

• I will make a multi input model where I will feed to it images and the text associated to them
```

we can see that the accuracy of training was improved for the first epoch but the values are the same as the old model so there are no improvements

```
Epoch 1/30
122/122 [=====] - 38s 225ms/step - loss: 1.3426 - accuracy: 0.4960 - val_loss: 1.2682 - val_accuracy: 0.4587
Epoch 2/30
122/122 [=====] - 26s 208ms/step - loss: 1.2488 - accuracy: 0.5127 - val_loss: 1.2675 - val_accuracy: 0.4587
Epoch 3/30
122/122 [=====] - 26s 208ms/step - loss: 1.2393 - accuracy: 0.5127 - val_loss: 1.2717 - val_accuracy: 0.4587
```

Figure 1: old model

```
Epoch 1/30
122/122 [=====] - 28s 215ms/step - loss: 1.2681 - accuracy: 0.5106 - val_loss: 1.2722 - val_accuracy: 0.4587
Epoch 2/30
122/122 [=====] - 27s 212ms/step - loss: 1.2460 - accuracy: 0.5127 - val_loss: 1.2750 - val_accuracy: 0.4587
Epoch 3/30
122/122 [=====] - 28s 222ms/step - loss: 1.2386 - accuracy: 0.5127 - val_loss: 1.2674 - val_accuracy: 0.4587
```

Figure 2: model with added layers

9th Step :

Now we will make a new model to process text instead of a CNN model ,the model will be a simple sequential model ,

as what we did with images ,instead of recording the names of images belonging to each class ,I will record the text belonging to each class ,letting in mind that I need to maintain the same index for the text and images so I keep the same labels (having in mind that I will use the multi input model in future)

```

• and then I will use a pretrained model

[25] text_dict = {}
    with open("/content/memotio dataset 7k/labels.csv","r") as file:
        labels = csv.reader(file)
        next(labels)
        for row in labels:
            text_dict.setdefault(row[-1], []).append(row[3])
            #this will be a dictionary where I will store the text discription of each image belonging to each sentiment class
            #csv.reader returns an iterable ,next will increment the iterrable so I pass the header and I only store image names and their classes
            # for each key ,I append to an empty list the corresponding image text so I get the list of texts for each class

• #checking if each text discription will be linked to the correct image
text_dict["positive"][0],label_dict_1["positive"][0],label_dict["positive"][0]

["Sam Thorne @Strippin ( Follow Follow Saw everyone posting these 2009 vs 2019 pics so here's mine 6:23 PM · 12 Jan 2019 0 636 Retweets 3 224 LIKES 65 636 3.2K",
'image 3.JPG',
'positive 0.JPG')

```

the text descriptions are in the 4th column of the csv file ,classes are in the final column (index -1, and 3) so for each class as a key I append to an empty list the corresponding class ,that empty list is the value of the text_dict dictionary .

now we need to decode the text into integers with the help of a dictionary ,We though that we could use a dictionary from the imdb example that is prebuilt in keras . So for each word we store the index of the corresponding word if its exists if it doesn't I replace it with 0 instead ,this could not be practical because 0 could alter the text meaning and bias the model,but it what I though about using .

Just as before,for each class ,I add to an empty list a list that contains the integers for

```

positive 0.JPG

• # I need a dictionary of words to be able to numerically encode the text discriptions so I will use the imdb get_word_index
from tensorflow.keras.datasets import imdb
import tensorflow as tf
word_index = imdb.get_word_index() #Returns the index of a word inside a dictionary

vecto_words = {}
for category in list(text_dict.keys()):
    for i,name in enumerate(text_dict[category]):
        vecto_words.setdefault(category, []).append([word_index.get(word,0) for word in text_dict[category][i] ]) #get the list of word indexes and replace a missing word index with 0
        #vecto_words[category][i] = [1 for i in vecto_words[category][i] if i != 0 ] # remove 0 and its occurrences 0

```

each text ,the role of get word method is will return the index of the word if it exist ,or it will return 0 if not

10th Step :

Now that we have numbers instead of words ,we prepare the train ,test and validation the same way we did with images

```

• # split text data into train,test and validation

t_train = {}
train_positive = vecto_words["positive"][:2000]
train_very_positive = vecto_words["very_positive"][:600]
train_neutral = vecto_words["neutral"][:1000]
train_negative = vecto_words["negative"][:200]
train_very_negative = vecto_words["very_negative"][:100]
t_train["positive"],t_train["very_positive"],t_train["neutral"],t_train["negative"],t_train["very_negative"] = train_positive,train_very_positive, train_neutral, train_negative, train_very_negative

t_test = {}
test_positive = vecto_words["positive"][2000:2500]
test_very_positive = vecto_words["very_positive"][600:800]
test_neutral = vecto_words["neutral"][1000:1800]
test_negative = vecto_words["negative"][200:400]
test_very_negative = vecto_words["very_negative"][100:125]
t_test["positive"],t_test["very_positive"],t_test["neutral"],t_test["negative"],t_test["very_negative"] = test_positive,test_very_positive, test_neutral, test_negative, test_very_negative

t_validation = {}
validation_positive = vecto_words["positive"][2500:]
validation_very_positive = vecto_words["very_positive"][800:]
validation_neutral = vecto_words["neutral"][1800:]
validation_negative = vecto_words["negative"][400:]
validation_very_negative = vecto_words["very_negative"][125:]
t_validation["positive"],t_validation["very_positive"],t_validation["neutral"],t_validation["negative"],t_validation["very_negative"] = validation_positive,validation_very_positive, validation_neutral, validation_very_negative

```


11th Step :

the data is stored in a dictionary and to be able to use it to train a model it should be in the form of tensors ,so we defined a function that transforms the data ,for each vector belonging to each class ,store the vector on a list and its corresponding class in separate list ,we used the same labels as in the CNN section ,hoping that later I can

```
import numpy as np
def load_data(data_name):
    labels = ["negative","neutral","positive","very_negative","very_positive"] # will help me change class names with their corresponding numbers
    data = []
    data_labels = []

    for label in labels:
        for elt in data_name[label]:
            if(len(elt) != 0):
                data.append(elt)
                data_labels.append(labels.index(label))

    return np.asarray(data),np.asarray(data_labels)

#loop over labels
# loop over samples of vectorized data ,each text was replaced with a vector of indexes before we loop over each vector in each class
# if the elt is not 0 ,that is the 0 we use to replace the missing data before
# append the value to the list called label
#append its corresponding label in a different list so we end up with a vector of samples ,and its corresponding vector of labels
# the tow lists need to be converted to numpy array so we can see their shapes

train_data,train_labels = load_data(t_train)
val_data,val_labels = load_data(t_validation)

# get train data and its labels using our created function
#get the validation data and their labels

return np.asarray(data),np.asarray(data_labels)
```

<ipython-input-30-6a78c44f0de7>:13: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated

use them both to train 11th Step :in a multi input model

12th Step:

we can now define a sequential model ,before feeding the data to this model we should convert it to a tensor that contains only 0s and 1s ,so we used a function that takes each data vector and replace each index of that vector by 1 in that position index ,for making such a tensor we need to search for the max index available in the train and validation data because that will be used as the second axis dimension for our tensor ,we finally get a tensor with (number of samples ,dimension) shape ,for each sample we have a vector that contains 0 and 1s ,and its label ,we could also hot encode the labels ,for this we used to_categorical

```
[31] # Build a simple 3 layers sequential model

model_t = keras.Sequential([
    layers.Dense(128,activation="relu"),
    layers.Dense(128,activation="relu"),
    layers.Dense(5,activation="softmax")
])
model_t.compile(optimizer="rmsprop",loss="categorical_crossentropy",metrics=["accuracy"])

[32] dimension = max(max([max(sequence) for sequence in train_data ]),max([max(sequence) for sequence in val_data ])) + 1 #search for the largest word index
dimension

63165

import numpy as np
def vectorize_data(sequences,dimension=63165):
    """ this function will one hot encode each data sample by replacing each value with 1 corresponding to the index value """
    result = np.zeros((len(sequences),dimension))
    for i,sequence in enumerate(sequences):
        for j in sequence:
            result[i,j] = 1.
    return result
x_train = vectorize_data((train_data))
x_val = vectorize_data(val_data)
x_train.shape

(3899, 63165)

[34] y_train=np.asarray(train_labels).astype("float32") # transforming labels to array of float32
y_val = np.asarray(val_labels).astype("float32")

[35] # one hot encoding the labels
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_train.shape
```

we fit then our model

```
(3899, 5)

history = model_t.fit(x_train,y_train,epochs=30,batch_size=64,validation_data=(x_val,y_val))

Epoch 1/30
61/61 [=====] - 2s 23ms/step - loss: 1.2779 - accuracy: 0.5076 - val_loss: 1.2968 - val_accuracy: 0.4575
Epoch 2/30
61/61 [=====] - 1s 16ms/step - loss: 1.2310 - accuracy: 0.5127 - val_loss: 1.2813 - val_accuracy: 0.4575
Epoch 3/30
61/61 [=====] - 1s 16ms/step - loss: 1.2279 - accuracy: 0.5127 - val_loss: 1.2730 - val_accuracy: 0.4575
Epoch 4/30
61/61 [=====] - 1s 16ms/step - loss: 1.2253 - accuracy: 0.5127 - val_loss: 1.2785 - val_accuracy: 0.4575
Epoch 5/30
```

add more layers to see if we can make him over fits and use callbacks

it seems that a sequential model is performing a littel better than the CNN on images which means that the textual data has more clear bonds and patterns than images

1. I will Add more layers to the model to improve its power
2. I will use the EarlyStop callback to stop training the model as soon the accuracy stalls, and modelcheckpoint callbacks to save the best version of the model as I did in the CNN training section .

```
#definig a new model with more layers
model_t_1 = keras.Sequential([
    layers.Dense(128,activation="relu"),
    layers.Dense(128,activation="relu"),
    layers.Dense(256,activation="relu"),
    layers.Dense(512,activation="relu"),
    layers.Dense(5,activation="softmax")
])
model_t_1.compile(optimizer="rmsprop",loss="categorical_crossentropy",metrics=["accuracy"])

[38] callback_list = [keras.callbacks.EarlyStopping(monitor="val_accuracy",patience=2),
                    keras.callbacks.ModelCheckpoint(filepath="checkpoint_text_model.keras",monitor="val_accuracy",save_best_only=True)]
history = model_t_1.fit(x_train,y_train,epochs=30,batch_size=64,validation_data=(x_val,y_val),callbacks=callback_list)

Epoch 1/30
61/61 [=====] - 2s 26ms/step - loss: 1.2655 - accuracy: 0.5050 - val_loss: 1.2725 - val_accuracy: 0.4575
Epoch 2/30
61/61 [=====] - 1s 16ms/step - loss: 1.2365 - accuracy: 0.5127 - val_loss: 1.2680 - val_accuracy: 0.4575
Epoch 3/30
61/61 [=====] - 1s 16ms/step - loss: 1.2322 - accuracy: 0.5127 - val_loss: 1.2718 - val_accuracy: 0.4575
```

3th Step :

since the text and images seems to hold the same results I hough of using a multiinput model with the help of the functional api but it didn't work due to an error because of data labeling ,one coming from dataset objects and the others from a tensor ,nonetheless Ill try to make it work and see what it gives

it seems that text alone wont pass the 50% accuracy ,I will now use a model with multi inputs by combining the CNN and the sequential models with the help of the functional API

```
input_text = keras.Input(shape=(dimension,))
input_images = keras.Input(shape=(300,300,3))

#defining the model that will process text
x = layers.Dense(64,activation="relu")(input_text)
x = layers.Dense(128,activation="relu")(x)
x = layers.Dense(256,activation="relu")(x)
outputs = layers.Dense(28,activation="relu")(x) #intermediate layer to use it for concatenation
model_text = keras.Model(inputs = input_text,outputs = outputs)

#defining the model that will process images I will use the same model as before but with changing the layer that becomes after Flatten
y = layers.Rescaling(1./255)(input_images) # rescaling inputs by deviding each pixel by 255
y = layers.Conv2D(filters = 32, kernel_size=3,activation="relu")(y)
y = layers.MaxPooling2D(pool_size=2)(y) #downsampling the input ,deviding the input shape by 2 and pooling the max value by a 2*2 window
y = layers.Conv2D(filters = 64, kernel_size = 3,activation = "relu")(y)
y = layers.MaxPooling2D(pool_size = 2)(y)
y = layers.Conv2D(filters=128, kernel_size = 3,activation = "relu" )(y)
y = layers.MaxPooling2D(pool_size = 2)(y)
y = layers.Conv2D(filters=256, kernel_size = 3,activation = "relu" )(y)
y = layers.MaxPooling2D(pool_size = 2)(y)
y = layers.Conv2D(filters=256, kernel_size = 3,activation = "relu" )(y)
y = layers.Flatten()(y)

outputs_2 = layers.Dense(28,activation = "relu")(y)
model_images = keras.Model(inputs = input_images,outputs = outputs_2 )

# concatenation both model outputs
features = layers.concatenate([model_images.output,model_text.output])

# we finish by adding the classification layer
outputs_3 = layers.Dense(5,activation="softmax")(features)

final_model = keras.Model(inputs = [input_images,input_text],outputs =outputs_3 )
```

Final Step:

the final step was to use a pretrained model ,we could use Xception ,ResNet ,Densnet ,VGG19 or others ,we tried to use VGG16 a pretrained model on the ImageNet dataset with 1000 classes ,the first thing after importing the model base ,without the classification layer is to ,use its base to predict on the train and validation data,and record that output in list ,that will be used later as an input to a classifier that we create for our needs ,5 class classification ,,this is exactly as freezing the weights and training the classification layer

That didn't work because the train_dataset object contains the labels and the labels for the text model are not coherent with images

we need more time to deal with it but I will skip it because there is no time left for the report and to push the notebook

```
# there is no improvements at all I will try to use a pretrained model
#I will load the VGG16 which is a model trained on the Imagenet dataset that contains 1000 classes so what I need to do is remove the classification layer
#Freeze the model base weights and retrain the model on my images ,luckilly I can only load the Base by setting te include_top to False
new_model_base = keras.applications.vgg16.VGG16(weights='imagenet',include_top=False,input_shape=(300, 300, 3))
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [=====] - 3s 0us/step

[43] new_model_base.summary()

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_4 (InputLayer)	[(None, 300, 300, 3)]	0
block1_conv1 (Conv2D)	(None, 300, 300, 64)	1792
block1_conv2 (Conv2D)	(None, 300, 300, 64)	36928
block1_pool (MaxPooling2D)	(None, 150, 150, 64)	0
block2_conv1 (Conv2D)	(None, 150, 150, 128)	73856
block2_conv2 (Conv2D)	(None, 150, 150, 128)	147584
block2_pool (MaxPooling2D)	(None, 75, 75, 128)	0
block3_conv1 (Conv2D)	(None, 75, 75, 256)	295168

After importing the base model ,we can use its weights and runs it through our dataset ,record the outputs of the process and then use them as an input to another classifier this exactly equivalent to freezing model weights ,adding a densely classification layer to it and fitting it to our data what will happend is the base will output a prediction and the training will be held only on the classification layer

```
def get_features_and_labels(dataset):
    """ This functions runs a prediction over our dataset and record the predicted values in numpy arrays"""
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images) #Preprocessing images which include normalization
        features = new_model_base.predict(preprocessed_images) #predict using the pretrained model base
        all_features.append(features) #store the predicted values and there labels
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)
```

```
train_features, train_labels = get_features_and_labels(train_dataset)
val_features, val_labels = get_features_and_labels(validation_dataset)
```

```
1/1 [=====] - 6s 6s/step
1/1 [=====] - 0s 37ms/step
1/1 [=====] - 0s 38ms/step
1/1 [=====] - 0s 52ms/step
1/1 [=====] - 0s 50ms/step
1/1 [=====] - 0s 54ms/step
1/1 [=====] - 0s 60ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 44ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 138ms/step
1/1 [=====] - 0s 44ms/step
1/1 [=====] - 0s 73ms/step
1/1 [=====] - 0s 54ms/step
1/1 [=====] - 0s 40ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 44ms/step
```