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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

```
import os
import random

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import cv2

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications import VGG19
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers
```

```
data_path = '/content/drive/MyDrive/Shuffeled_BT_Dataset'
```

```
def main(train_data_path):

    """
    return training and testing dataframes including 2 columns image path and its label

    Args:
      train_data_path (string): string includes your train data path
      test_data_path (string): string includes your test data path
      note: data_path argument must contains your classes dirs

    Returns:
      train_df (dataframe):  It consists of two columns (the path of the image and its label) and contains records of the path of the training images and the label of each image
      test_df (dataframe):  It consists of two columns (the path of the image and its label) and contains records of the path of the testing images and the label of each image

    """

    for dirs in [train_data_path]:

        filepaths = []
        labels = []
        classes_list = sorted(os.listdir(dirs))

        filepaths,labels = get_fileList(classes_list, dirs, filepaths, labels)
```

```
            train_df = obtain_train_df(filepaths, labels)


        return train_df



    def get_fileList(classes_list, dirs, filepaths, labels):


        """
        return file paths list including all full image paths and label for each one

        Args:
          classes_list (list): list includes name of classes dirs
          dirs (string): the data path
          filepaths (list): empty list to fill with images paths later
          labels (list): empty list to fill with each image label later

        Returns:
          count (scalar):  number of images in your dataset
        """


        for s_class in classes_list:

            label = s_class
            classpath = os.path.join(dirs, s_class)
            flist = sorted(os.listdir(classpath))

            store_image_names_and_its_labels(flist, classpath, filepaths, labels, label)

        return filepaths, labels


    def store_image_names_and_its_labels(flist, classpath, filepaths, labels, label):

        """
        return nothing, it fills the filepaths list with all the images paths and its labels

        Args:
          flist (list) : contains all the image names in certain class
          classes_list (list): list includes name of classes dirs
          filepaths (list): stores all the images paths
          labels (list): stores the label of each image in the dataset
          label (string): label for a specific image

        Returns:
          nothing
        """


        for file in flist:
            file_path = os.path.join(classpath, file)
            filepaths.append(file_path)
            labels.append(label)
```

```
def obtain_train_df(filepaths, labels):

    """
    Return Dataframe with 2 columns contains each image path with its label

    Args:
      filepaths (list): stores all the images paths
      labels (list): stores the label of each image in the dataset

    Returns:
      train_df (DataFrame):  Dataframe with 2 columns contains each image path with its label
    """

    train_df =pd.DataFrame({
        'image_path' : filepaths,
        'label' : labels
    })
    return train_df


data_df = main(data_path)


data_df.head()
```

	image_path	label
0	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1
1	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1
2	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1
3	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1
4	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1

```
num_classes = len(data_df['label'].unique())
print(f"We have {num_classes} classes")
print(f"We have {data_df.shape[0]} images")
```

```
We have 44 classes
We have 4497 images
```

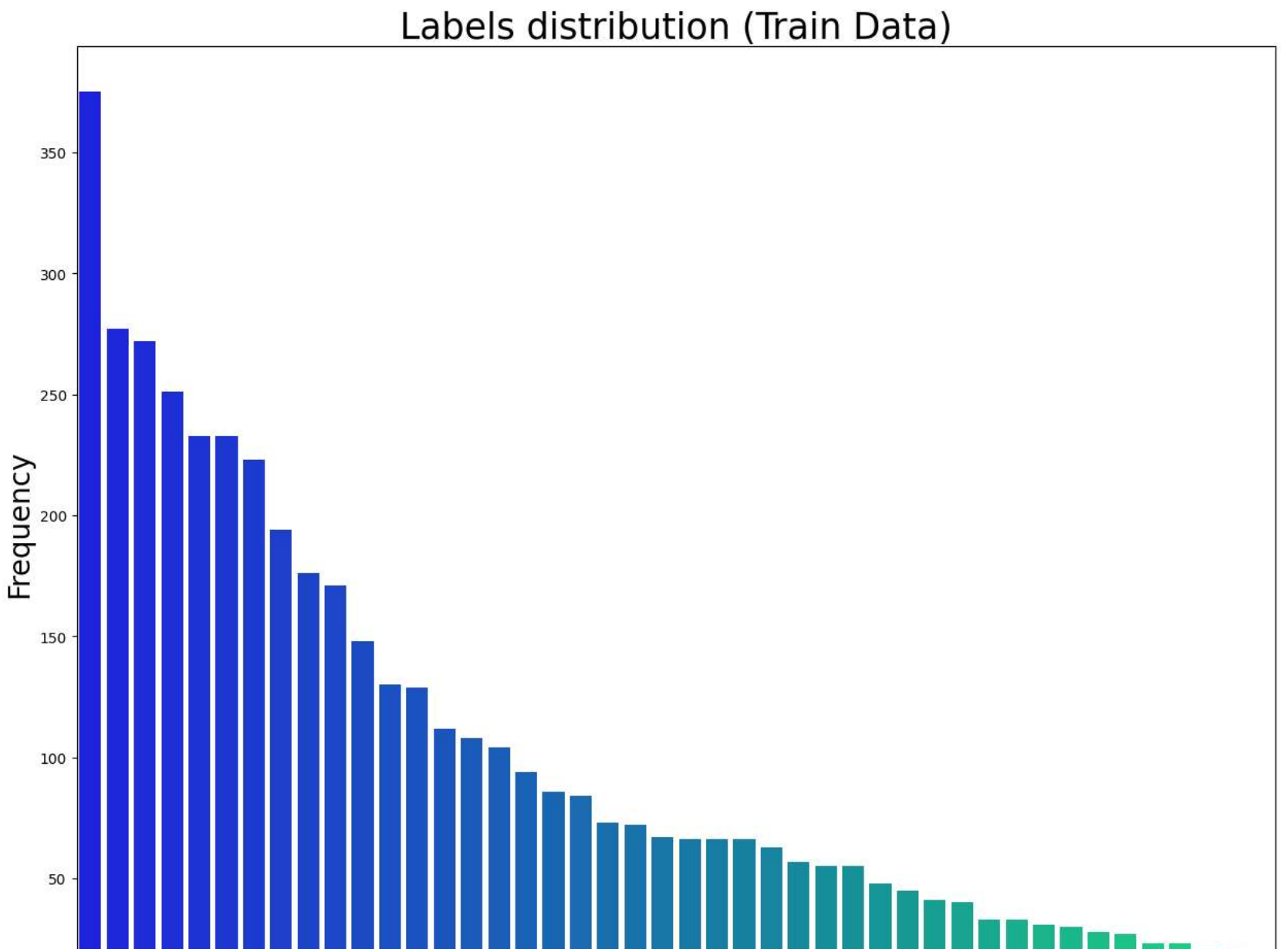
```
data_df['label'].value_counts()
```

```
Meningioma T1C+           375
_NORMAL T2                277
Meningioma T1             272
_NORMAL T1                251
Astrocitoma T1C+          233
Meningioma T2             233
Neurocitoma T1C+          223
Schwannoma T1C+           194
Astrocitoma T1            176
```

```
Astrocitoma T2            171
Schwannoma T1             148
Neurocitoma T1            130
Schwannoma T2             129
Carcinoma T1C+            112
Papiloma T1C+             108
Neurocitoma T2            104
Glioblastoma T1C+          94
Oligodendroglioma T1       86
Tuberculoma T1C+           84
Carcinoma T2               73
Oligodendroglioma T1C+     72
Meduloblastoma T1C+        67
Papiloma T1                66
Oligodendroglioma T2       66
Carcinoma T1               66
Papiloma T2                63
Ependimoma T2              57
Glioblastoma T2            55
Glioblastoma T1            55
Ependimoma T1C+            48
Ependimoma T1              45
Meduloblastoma T2          41
Germinoma T1C+             40
Tuberculoma T2             33
Germinoma T2               33
Granuloma T1C+             31
Granuloma T1               30
Tuberculoma T1             28
Germinoma T1               27
Ganglioglioma T2           23
Meduloblastoma T1          23
Ganglioglioma T1           20
Ganglioglioma T1C+         18
Granuloma T2               17
Name: label, dtype: int64
```

```
def count_plot(x, title, xlabel, ylabel, width, length, order = None, rotation=False, palette='winter'):
    plt.figure(figsize=(width,length))
    sns.countplot(x = x, palette=palette, order = order)
    plt.title(title, fontsize=25)
    if rotation:
        plt.xticks(rotation = 'vertical')
    plt.xlabel(xlabel, fontsize=20)
    plt.ylabel(ylabel, fontsize=20)
    plt.show()
```

```
x = data_df['label']
order = x.value_counts().index
count_plot(x, "Labels distribution (Train Data)", "Label", 'Frequency', 15,12, rotation = True, order=order)
```




```
test_split_size = 0.3
train_df, test_df = train_test_split(data_df, test_size=test_split_size, shuffle=True, random_state=123, stratify=data_df['label'])
test_df, valid_df = train_test_split(test_df, test_size=0.5, shuffle=True, random_state=123, stratify=test_df['label'])
```

```
x = train_df['label']
order = x.value_counts().index
```

```
count_plot(x, "Labels distribution (Train Data)", "Label", 'Frequency', 15,12, rotation = True, order=order)
```

Labels distribution (Train Data)

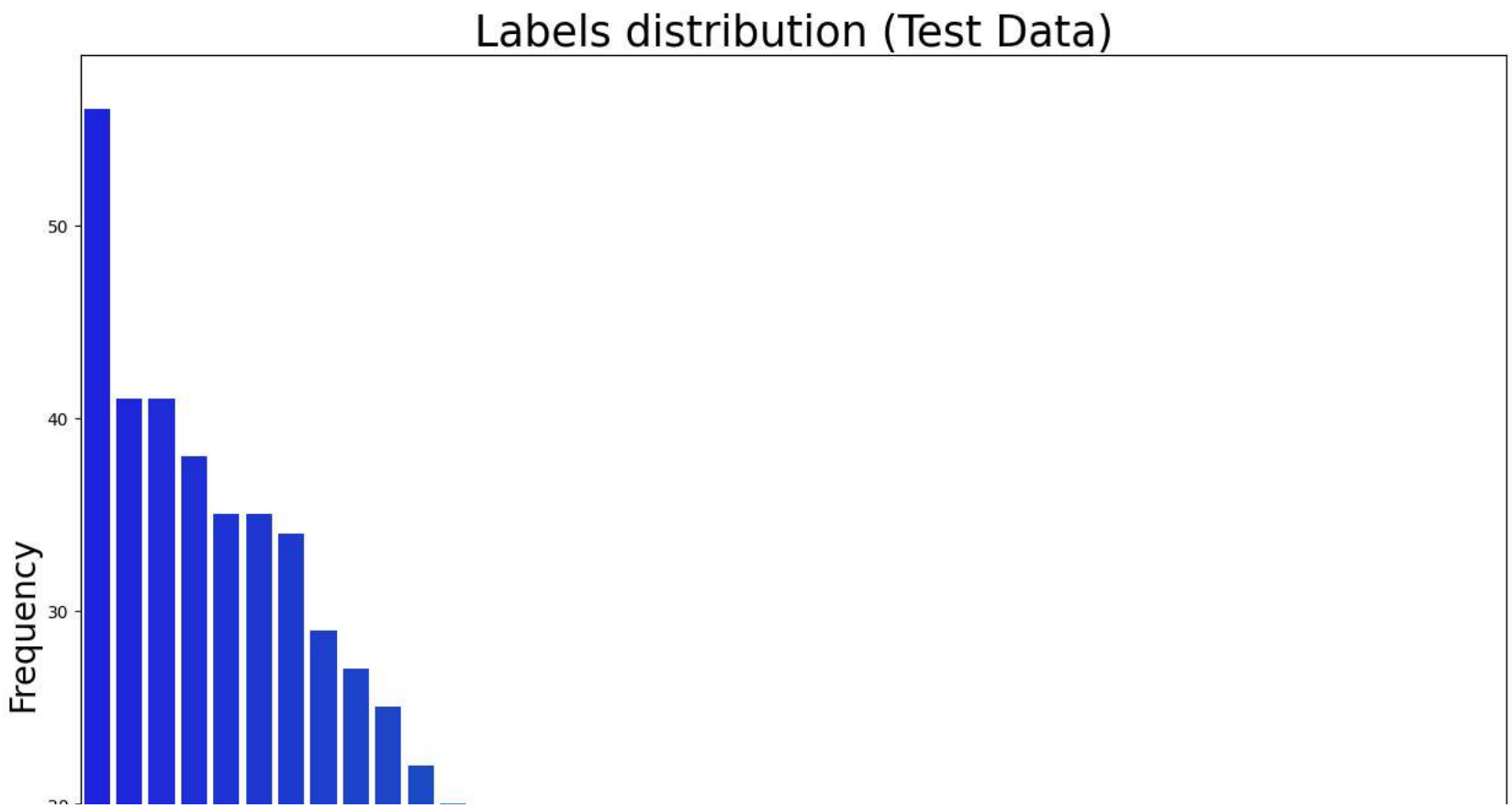
```
train_df.head()
```

	image_path	label
3972	/content/drive/MyDrive/Shuffeled_BT_Dataset/_N...	_NORMAL T1
3334	/content/drive/MyDrive/Shuffeled_BT_Dataset/Pa...	Papiloma T2
2885	/content/drive/MyDrive/Shuffeled_BT_Dataset/Ne...	Neurocitoma T2
1687	/content/drive/MyDrive/Shuffeled_BT_Dataset/Me...	Meningioma T1
1381	/content/drive/MyDrive/Shuffeled_BT_Dataset/Gr...	Granuloma T1C+

```
test_df.head()
```

	image_path	label
2323	/content/drive/MyDrive/Shuffeled_BT_Dataset/Me...	Meningioma T2
288	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1C+
3229	/content/drive/MyDrive/Shuffeled_BT_Dataset/Pa...	Papiloma T1C+
1496	/content/drive/MyDrive/Shuffeled_BT_Dataset/Me...	Medulloblastoma T1C+
3327	/content/drive/MyDrive/Shuffeled_BT_Dataset/Pa...	Papiloma T2

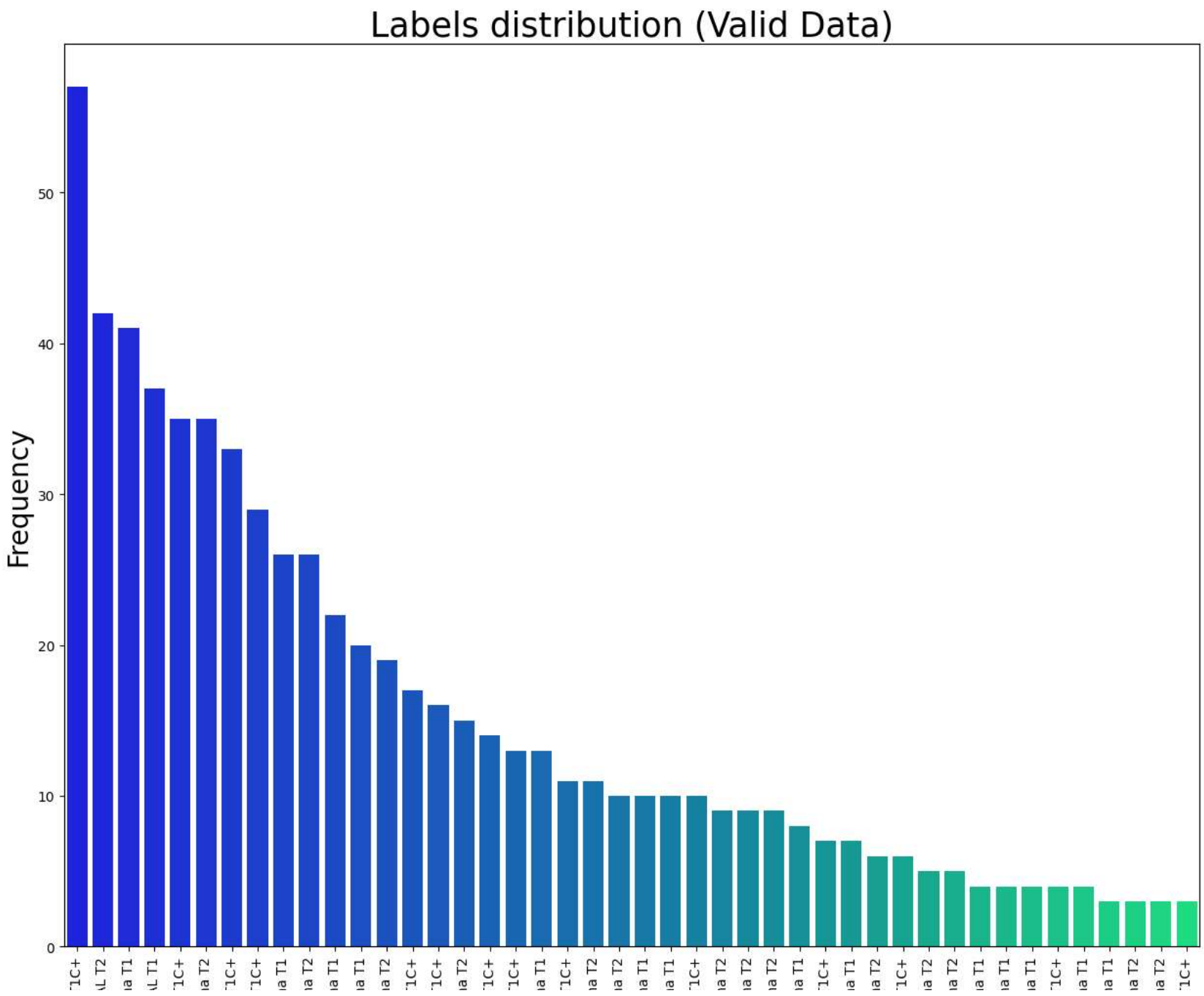
```
x = test_df['label']
order = x.value_counts().index
count_plot(x, "Labels distribution (Test Data)", "Label", 'Frequency', 15,12, rotation = True, order=order)
```




```
valid_df.head()
```

	image_path	label
2730	/content/drive/MyDrive/Shuffeled_BT_Dataset/Ne...	Neurocitoma T1C+
300	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T1C+
553	/content/drive/MyDrive/Shuffeled_BT_Dataset/As...	Astrocitoma T2
4000	/content/drive/MyDrive/Shuffeled_BT_Dataset/_N...	_NORMAL T1
892	/content/drive/MyDrive/Shuffeled_BT_Dataset/Ep...	Ependimoma T1C+

```
x = valid_df['label']
order = x.value_counts().index
count_plot(x, "Labels distribution (Valid Data)", "Label", 'Frequency', 15,12, rotation = True, order=order)
```




```
print(f"We have {len(train_df)} images in the training dataset\nWe have {len(valid_df)} images in the validating dataset\nWe have {len(test_df)} images in the testing dataset")
```

```
We have 3147 images in the training dataset
We have 675 images in the validating dataset
We have 675 images in the testing dataset
```

```
len(data_df)
```

```
4497
```

```
img_size = (224, 224)
channels = 3
color = 'rgb'
img_shape = (img_size[0], img_size[1], channels)
batch_size = 32
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size
def scalar(img):
    return img


tr_gen = ImageDataGenerator(preprocessing_function= scalar,
                            horizontal_flip= True)

ts_gen = ImageDataGenerator(preprocessing_function= scalar)

train_gen = tr_gen.flow_from_dataframe( train_df,
                                       x_col= 'image_path',
                                       y_col= 'label',
                                       target_size= img_size,
                                       class_mode= 'categorical',
                                       color_mode= color,
                                       shuffle= True,
                                       batch_size= batch_size)

valid_gen = ts_gen.flow_from_dataframe( valid_df,
                                       x_col= 'image_path',
                                       y_col= 'label',
                                       target_size= img_size,
                                       class_mode= 'categorical',
                                       color_mode= color,
                                       shuffle= True,
                                       batch_size= batch_size)

test_gen = ts_gen.flow_from_dataframe( test_df,
                                      x_col= 'image_path',
                                      y_col= 'label',
                                      target_size= img_size,
                                      class_mode= 'categorical',
                                      color_mode= color,
                                      shuffle= False,
                                      batch_size= test_batch_size)
```

```
Found 3146 validated image filenames belonging to 44 classes.
Found 675 validated image filenames belonging to 44 classes.
/usr/local/lib/python3.10/dist-packages/keras/src/preprocessing/image.py:1137: UserWarning: Found 1 invalid image filename(s) in x_col="image_path". These filename(s) will be ignored.
```

```
  warnings.warn(
Found 675 validated image filenames belonging to 44 classes.
```

```
class_count = len(list(train_gen.class_indices.keys()))
```

```
base_model = tf.keras.applications.efficientnet.EfficientNetB5(include_top= False,
                                                               weights= "imagenet",
                                                               input_shape= img_shape,
                                                               pooling= 'max')

model = Sequential([
    base_model,
    BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
    Dense(256,
          kernel_regularizer= regularizers.l2(l= 0.016),
          activity_regularizer= regularizers.l1(0.006),
          bias_regularizer= regularizers.l1(0.006),
          activation= 'relu'),

    Dropout(rate= 0.45,
            seed= 123),

    Dense(class_count, activation= 'softmax')
])

model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy', metrics= ['accuracy'])

model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb5_notop.h5
115263384/115263384 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb5 (Functional  (None, 2048)              28513527
 )

 batch_normalization (Batch  (None, 2048)              8192
 Normalization)

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 44)                11308

=================================================================
Total params: 29057571 (110.85 MB)
Trainable params: 28880732 (110.17 MB)
Non-trainable params: 176839 (690.78 KB)
_________________________________________________________________
```

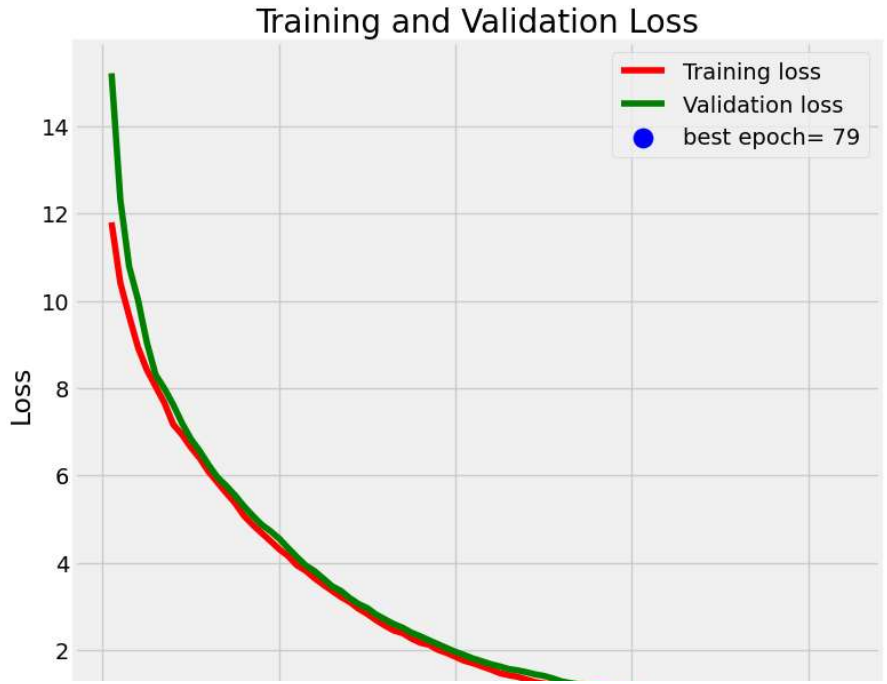
```
#info about optimizers
model.optimizer.get_config()
```

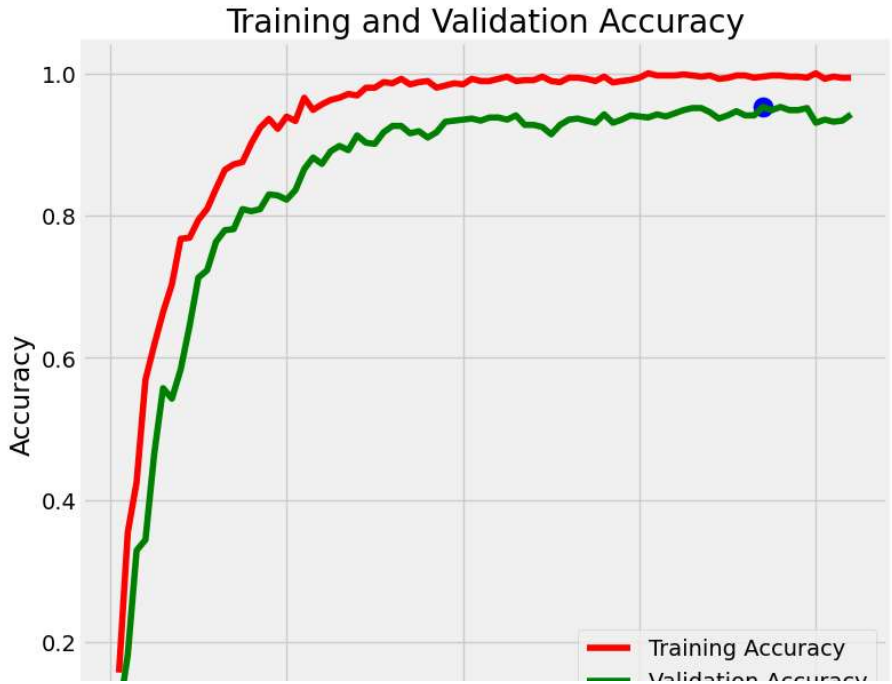
```
      {'name': 'Adamax',
       'weight_decay': None,
       'clipnorm': None,
       'global_clipnorm': None,
       'clipvalue': None,
       'use_ema': False,
       'ema_momentum': 0.99,
       'ema_overwrite_frequency': None,
       'jit_compile': True,
       'is_legacy_optimizer': False,
       'learning_rate': 0.001,
       'beta_1': 0.9,
       'beta_2': 0.999,
       'epsilon': 1e-07}


early_stop = EarlyStopping(monitor='val_loss',
                           patience=5,
                           verbose=1)

checkpoint = ModelCheckpoint('model_weights.h5',
                             monitor='val_loss',
                             save_best_only=True,
                             save_weights_only=True,
                             mode='min',
                             verbose=1)


history = model.fit(x= train_gen,
                    steps_per_epoch = 20,
                    epochs= 100,
                    callbacks=[early_stop, checkpoint],
                    validation_data= valid_gen)
```

```
20/20 [==============================] - ETA: 0s - loss: 0.5866 - accuracy: 0.9953
Epoch 77: val_loss improved from 0.74300 to 0.73964, saving model to model_weights.h5
20/20 [==============================] - 25s 1s/step - loss: 0.5866 - accuracy: 0.9953 - val_loss: 0.7396 - val_accuracy: 0.9481
Epoch 78/100
20/20 [==============================] - ETA: 0s - loss: 0.5656 - accuracy: 0.9953
Epoch 78: val_loss improved from 0.73964 to 0.72302, saving model to model_weights.h5
20/20 [==============================] - 24s 1s/step - loss: 0.5656 - accuracy: 0.9953 - val_loss: 0.7230 - val_accuracy: 0.9481
Epoch 79/100
20/20 [==============================] - ETA: 0s - loss: 0.5707 - accuracy: 0.9937
Epoch 79: val_loss improved from 0.72302 to 0.70574, saving model to model_weights.h5
20/20 [==============================] - 24s 1s/step - loss: 0.5707 - accuracy: 0.9937 - val_loss: 0.7057 - val_accuracy: 0.9511
Epoch 80/100
20/20 [==============================] - ETA: 0s - loss: 0.5522 - accuracy: 1.0000
Epoch 80: val_loss did not improve from 0.70574
20/20 [==============================] - 22s 1s/step - loss: 0.5522 - accuracy: 1.0000 - val_loss: 0.7860 - val_accuracy: 0.9304
Epoch 81/100
20/20 [==============================] - ETA: 0s - loss: 0.5689 - accuracy: 0.9922
Epoch 81: val_loss did not improve from 0.70574
20/20 [==============================] - 24s 1s/step - loss: 0.5689 - accuracy: 0.9922 - val_loss: 0.7592 - val_accuracy: 0.9348
Epoch 82/100
20/20 [==============================] - ETA: 0s - loss: 0.5530 - accuracy: 0.9953
Epoch 82: val_loss did not improve from 0.70574
20/20 [==============================] - 23s 1s/step - loss: 0.5530 - accuracy: 0.9953 - val_loss: 0.7472 - val_accuracy: 0.9319
Epoch 83/100
20/20 [==============================] - ETA: 0s - loss: 0.5519 - accuracy: 0.9935
Epoch 83: val_loss did not improve from 0.70574
20/20 [==============================] - 22s 1s/step - loss: 0.5519 - accuracy: 0.9935 - val_loss: 0.7513 - val_accuracy: 0.9333
Epoch 84/100
20/20 [==============================] - ETA: 0s - loss: 0.5419 - accuracy: 0.9937
Epoch 84: val_loss did not improve from 0.70574
20/20 [==============================] - 23s 1s/step - loss: 0.5419 - accuracy: 0.9937 - val_loss: 0.7315 - val_accuracy: 0.9422
Epoch 84: early stopping
```

```
def plot_training(hist):
    '''
    This function take training model and plot history of accuracy and losses with the best epoch in both of them.
    '''

    tr_acc = hist.history['accuracy']
    tr_loss = hist.history['loss']
    val_acc = hist.history['val_accuracy']
    val_loss = hist.history['val_loss']
    index_loss = np.argmin(val_loss)
    val_lowest = val_loss[index_loss]
    index_acc = np.argmax(val_acc)
    acc_highest = val_acc[index_acc]
    Epochs = [i+1 for i in range(len(tr_acc))]
    loss_label = f'best epoch= {str(index_loss + 1)}'
    acc_label = f'best epoch= {str(index_acc + 1)}'

    plt.figure(figsize= (20, 8))
    plt.style.use('fivethirtyeight')

    plt.subplot(1, 2, 1)
    plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
    plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
    plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
```

```
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
    plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
    plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout
    plt.show()


plot_training(history)
```





Evaluating the Model

```
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = model.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)
```

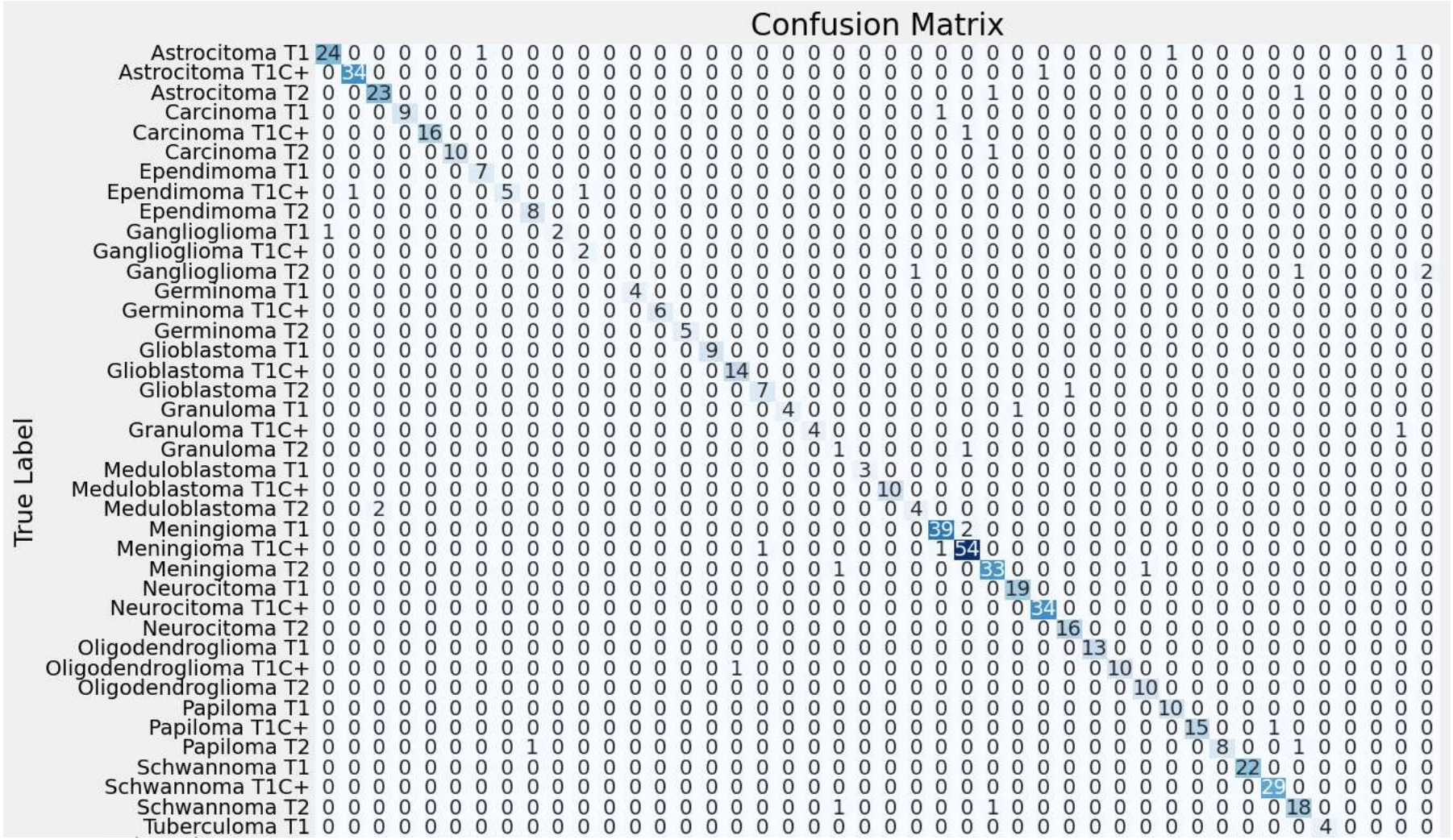
```
print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

```
9/9 [==============================] - 2s 246ms/step - loss: 0.4600 - accuracy: 1.0000
9/9 [==============================] - 4s 418ms/step - loss: 0.6851 - accuracy: 0.9549
9/9 [==============================] - 181s 22s/step - loss: 0.6970 - accuracy: 0.9467
Train Loss:  0.45998796820640564
Train Accuracy:  1.0
--------------------
Validation Loss:  0.685107946395874
Validation Accuracy:  0.9548611044883728
--------------------
Test Loss:  0.6970170736312866
Test Accuracy:  0.9466666579246521
```

```
y_pred = model.predict(test_gen)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = test_gen.classes
```

```
9/9 [==============================] - 5s 537ms/step
```

```
class_names = list(test_gen.class_indices.keys())

confusion_mtx = confusion_matrix(y_true_labels, y_pred_labels)

plt.figure(figsize=(12, 10))  # Increase the size of the plot

# Use the heatmap function to display the confusion matrix with better formatting
sns.heatmap(confusion_mtx, cmap="Blues", annot=True, fmt="d", xticklabels=class_names, yticklabels=class_names, cbar=False)

plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

# Rotate the tick labels for better readability
plt.xticks(rotation=90)
plt.yticks(rotation=0)

plt.show()
```



Generate Classification Report

```
report = classification_report(y_true_labels, y_pred_labels, target_names=class_names)

print("Classification Report: ")
print(report)
```

```
Classification Report:
                        precision    recall  f1-score   support

           Astrocitoma T1       0.96      0.89      0.92        27
         Astrocitoma T1C+       0.97      0.97      0.97        35
           Astrocitoma T2       0.92      0.92      0.92        25
```

```
                Carcinoma T1       1.00      0.90      0.95        10
               Carcinoma T1C+      1.00      0.94      0.97        17
                Carcinoma T2       1.00      0.91      0.95        11
               Ependimoma T1       0.88      1.00      0.93         7
              Ependimoma T1C+      1.00      0.71      0.83         7
               Ependimoma T2       0.89      1.00      0.94         8
            Ganglioglioma T1       1.00      0.67      0.80         3
           Ganglioglioma T1C+      0.50      1.00      0.67         2
            Ganglioglioma T2       0.00      0.00      0.00         4
                Germinoma T1       1.00      1.00      1.00         4
               Germinoma T1C+      1.00      1.00      1.00         6
                Germinoma T2       1.00      1.00      1.00         5
              Glioblastoma T1      1.00      1.00      1.00         9
             Glioblastoma T1C+     0.93      1.00      0.97        14
              Glioblastoma T2      0.88      0.88      0.88         8
                Granuloma T1       1.00      0.80      0.89         5
               Granuloma T1C+      1.00      0.80      0.89         5
                Granuloma T2       0.33      0.50      0.40         2
           Meduloblastoma T1       1.00      1.00      1.00         3
          Meduloblastoma T1C+      1.00      1.00      1.00        10
           Meduloblastoma T2       0.67      0.67      0.67         6
               Meningioma T1       0.95      0.95      0.95        41
              Meningioma T1C+      0.93      0.96      0.95        56
               Meningioma T2       0.92      0.94      0.93        35
              Neurocitoma T1       0.95      1.00      0.97        19
             Neurocitoma T1C+      0.97      1.00      0.99        34
              Neurocitoma T2       0.94      1.00      0.97        16
        Oligodendroglioma T1       1.00      1.00      1.00        13
       Oligodendroglioma T1C+      1.00      0.91      0.95        11
        Oligodendroglioma T2       0.91      1.00      0.95        10
                 Papiloma T1       0.91      1.00      0.95        10
                Papiloma T1C+      1.00      0.94      0.97        16
                 Papiloma T2       1.00      0.80      0.89        10
               Schwannoma T1       1.00      1.00      1.00        22
              Schwannoma T1C+      0.97      1.00      0.98        29
               Schwannoma T2       0.86      0.90      0.88        20
              Tuberculoma T1       1.00      1.00      1.00         4
             Tuberculoma T1C+      1.00      0.92      0.96        12
              Tuberculoma T2       1.00      0.80      0.89         5
                  _NORMAL T1       0.95      1.00      0.97        38
                  _NORMAL T2       0.95      1.00      0.98        41

                    accuracy                           0.95       675
                   macro avg       0.91      0.90      0.90       675
                weighted avg       0.94      0.95      0.94       675

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted
```

VGG19

```
tr_gen = ImageDataGenerator(preprocessing_function= scalar,
                            horizontal_flip= True)

ts_gen = ImageDataGenerator(preprocessing_function= scalar)
```

```
train_gen = tr_gen.flow_from_dataframe( train_df,
                                       x_col= 'image_path',
                                       y_col= 'label',
                                       target_size= img_size,
                                       class_mode= 'categorical',
                                       color_mode= color,
                                       shuffle= True,
                                       batch_size= batch_size)

valid_gen = ts_gen.flow_from_dataframe( valid_df,
                                       x_col= 'image_path',
                                       y_col= 'label',
                                       target_size= img_size,
                                       class_mode= 'categorical',
                                       color_mode= color,
                                       shuffle= True,
                                       batch_size= batch_size)

test_gen = ts_gen.flow_from_dataframe( test_df,
                                      x_col= 'image_path',
                                      y_col= 'label',
                                      target_size= img_size,
                                      class_mode= 'categorical',
                                      color_mode= color,
                                      shuffle= False,
                                      batch_size= test_batch_size)


class_count = len(list(train_gen.class_indices.keys()))



'''vgg19_base_model = tf.keras.applications.vgg19.VGG19(include_top= False,
                                                                  weights= "imagenet",
                                                                  input_shape= img_shape,
                                                                  pooling= 'max')



vgg19_base_model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy', metrics= ['accuracy'])

#vgg19_base_model.summary()'''


vgg19_model = VGG19(include_top=False, pooling='avg', weights='imagenet', input_shape = (img_shape))
for layer in vgg19_model.layers:
    layer.trainable = False
# Check if all layers except conv5 layers are not trainable
#for i, layer in enumerate(res_model.layers):
#    print(i, layer.name, "-", layer.trainable)


#info about optimizers
model.optimizer.get_config()
```

```
early_stop = EarlyStopping(monitor='val_loss',
                           patience=5,
                           verbose=1)

checkpoint = ModelCheckpoint('model_weights.h5',
                             monitor='val_loss',
                             save_best_only=True,
                             save_weights_only=True,
                             mode='min',
                             verbose=1)


'''history_vgg19 = vgg19_base_model.fit(x= train_gen,
                    epochs= 200,
                    callbacks=[early_stop, checkpoint],
                    validation_data= valid_gen)'''


history_vgg19 = model.fit(x= train_gen,
                    steps_per_epoch = 20,
                    epochs = 100,
                    verbose = 1,
                    validation_data = valid_gen,
                    callbacks = [early_stop, checkpoint])


# visualize the model accuracy for the training data and Validation data
plt.plot(history_vgg19.history['accuracy'], label='Train')
plt.plot(history_vgg19.history['val_accuracy'], label='Validation')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# visualize the model loss for the training data and Validation data
plt.plot(history_vgg19.history['loss'], label='Train')
plt.plot(history_vgg19.history['val_loss'], label='Validation')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()


result = model.evaluate(test_gen)



y_pred = model.predict(test_gen)

y_pred_labels = np.argmax(y_pred, axis=1)

y_true_labels = test_gen.classes


class_names = list(test_gen.class_indices.keys())
```

```
confusion_mtx = confusion_matrix(y_true_labels, y_pred_labels)

plt.figure(figsize=(10,8))
sns.heatmap(confusion_mtx, cmap="Blues", annot=True, fmt="d", xticklabels=class_names, yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```