```
#mount the drive

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

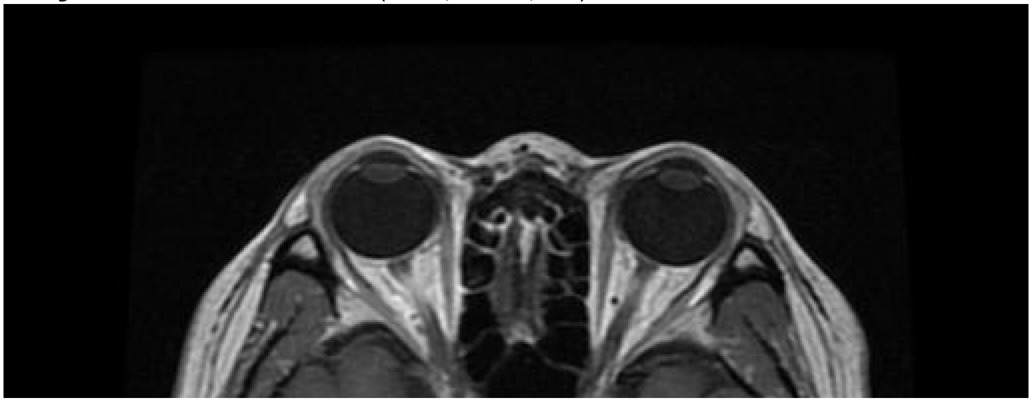
```
Mounted at /content/drive
```

```
#extract the dataset from zip

#!unzip /content/drive/MyDrive/brain_tumor_classification/archive.zip -d /content/driv
```

```
#import the required libraries

import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import os
import random

from tensorflow.keras.layers import *
```

```
#display a sample image

image_path = '/content/drive/MyDrive/brain_tumor_classification/Training/pituitary_tum
image_for_visualization = cv2.imread(image_path)
print('image dimensions are ', image_for_visualization.shape)
cv2_imshow(image_for_visualization)
```

```
image dimensions are  (512, 512, 3)
```



Defining the model architecture

Let's defined the model, to check the how it would work on our defined model. For our model :

```
epochs = 15

#dense model

dense_model = tf.keras.Sequential([tf.keras.layers.Input((256, 256, 3)),
                                  #tf.keras.layers.Dense(128, activation='relu'),
                                  #tf.keras.layers.Dense(64, activation='relu'),

                                  #tf.keras.layers.Dense(64, activation='relu'),
                                  #tf.keras.layers.Dense(32, activation='relu'),

                                  tf.keras.layers.Dense(32, activation='relu'),
                                  tf.keras.layers.Dense(16, activation='relu'),

                                  tf.keras.layers.Flatten(),
                                  tf.keras.layers.Dense(4, activation='softmax')
                                           ])

dense_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256, 256, 32)      128

 dense_1 (Dense)             (None, 256, 256, 16)      528

 flatten (Flatten)           (None, 1048576)           0

 dense_2 (Dense)             (None, 4)                 4194308

=================================================================
Total params: 4,194,964
```

```
Trainable params: 4,194,964
Non-trainable params: 0
_________________________________________________________________
```

```
#dense model

dense_model_with_dropout = tf.keras.Sequential([tf.keras.layers.Input((256, 256, 3)),
                                                #tf.keras.layers.Dense(128, activatio
                                                #tf.keras.layers.Dense(64, activation=
                                                #tf.keras.layers.Dropout(0.4),  #layer

                                                #tf.keras.layers.Dense(64, activation=
                                                #tf.keras.layers.Dense(32, activation=
                                                #tf.keras.layers.Dropout(0.4),

                                                tf.keras.layers.Dense(32, activation='
                                                tf.keras.layers.Dense(16, activation='
                                                tf.keras.layers.Dropout(0.4),

                                                tf.keras.layers.Flatten(),     #layer
                                                tf.keras.layers.Dense(4, activation='s

dense_model_with_dropout.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 256, 256, 32)      128

 dense_4 (Dense)             (None, 256, 256, 16)      528

 dropout (Dropout)           (None, 256, 256, 16)      0

 flatten_1 (Flatten)         (None, 1048576)           0

 dense_5 (Dense)             (None, 4)                 4194308

=================================================================
Total params: 4,194,964
Trainable params: 4,194,964
Non-trainable params: 0
_________________________________________________________________
```

```
# VGG16 example (off the shelf implementation)

model_vgg16 = tf.keras.applications.vgg16.VGG16(include_top=True,
                                                weights=None,
                                                input_shape=(256, 256, 3), classes=4,
                                                classifier_activation='softmax')
```

```
model_vgg16.compile(optimizer = tf.optimizers.Adam(),   # typically used optimizer fo
                    loss = 'categorical_crossentropy',        # used as the labels ar
                    metrics=['accuracy'])

model_vgg16.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 flatten (Flatten)           (None, 32768)             0

 fc1 (Dense)                 (None, 4096)              134221824

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 4)                 16388
```

```
=================================================================
Total params: 165,734,212
Trainable params: 165,734,212
Non-trainable params: 0
_________________________________________________________________
```

Creating the data generator

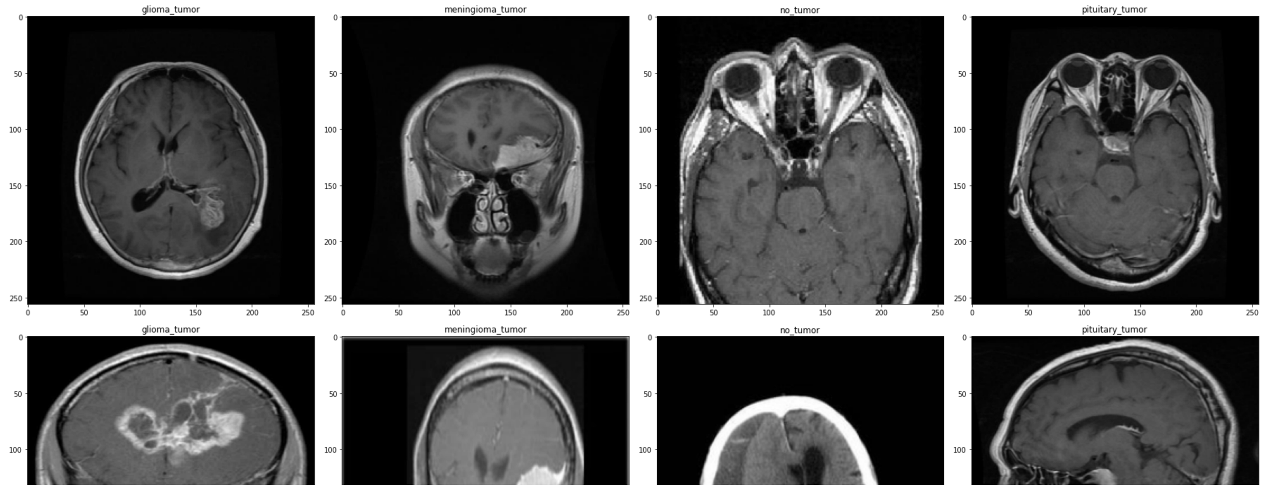
Defining the training, validation and test data generators

```
# path to the training directory
train_dir = '/content/drive/MyDrive/brain_tumor_classification/Training/'
test_dir = '/content/drive/MyDrive/brain_tumor_classification/Testing/'

# data augmentation to be applied in train_datagen [augmentation only applied on the t
train_datagen = ImageDataGenerator(rescale=1/255.,
                                   shear_range=0.20,
                                   zoom_range=0.20,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   validation_split=0.25)

test_datagen = ImageDataGenerator(rescale=1/255.)

# creating datagenerator for the training and validation data
train_generator = train_datagen.flow_from_directory(train_dir,  # This is the source d
                                                    target_size=(256, 256),
                                                    classes = ['glioma_tumor', 'mening
                                                    class_mode='categorical',
                                                    subset='training')

validation_generator = train_datagen.flow_from_directory(train_dir, # same directory a
                                                         target_size=(256, 256),
                                                         class_mode='categorical',
                                                         subset='validation') # set as

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(256, 256),
                                                  classes = ['glioma_tumor', 'meningio
                                                  class_mode='categorical')
```

```
Found 2155 images belonging to 4 classes.
Found 715 images belonging to 4 classes.
Found 394 images belonging to 4 classes.
```

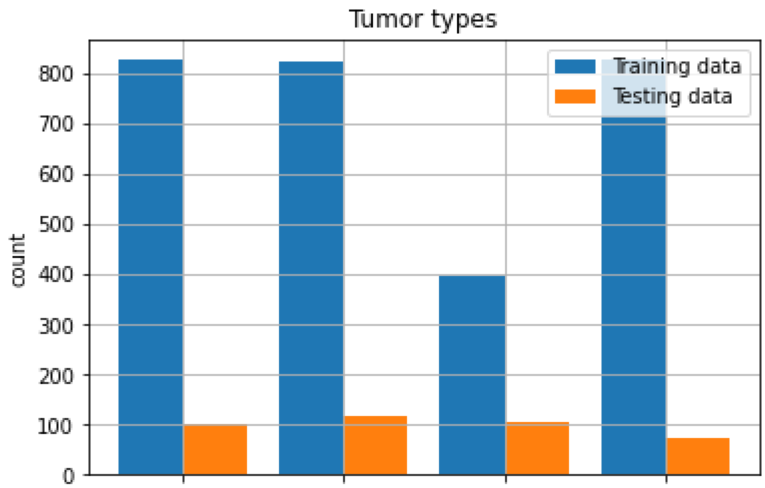
```
# visualization for training and testing data

def visualizing_data(list_of_image_path, directory_path):
```

```
    index = 0
    count = 1
    plt.figure(figsize=(25, 8))

    for image in list_of_image_path:
        path = directory_path + image
        l = os.listdir(path)
        img = l[index]
        img_for_visualization = cv2.imread(path+'/'+img)
        resized_img_for_visualization = cv2.resize(img_for_visualization, (256, 256))
        #print('image selected to be visualized ', path+'/'+img)
        plt.subplot(1,4,count)
        plt.imshow(resized_img_for_visualization)
        plt.title(image)
        count += 1

    plt.tight_layout()


training_dir = '/content/drive/MyDrive/brain_tumor_classification/Training/'
testing_dir = '/content/drive/MyDrive/brain_tumor_classification/Testing/'

train_tumor_types = os.listdir(training_dir)
testing_tumor_types = os.listdir(testing_dir)

visualizing_data(train_tumor_types, training_dir)
visualizing_data(testing_tumor_types, testing_dir)
```




```
# plot the bar chart

list_of_training_classes = os.listdir(train_dir)
list_of_testing_classes = os.listdir(test_dir)

count_of_number_of_images_training = []
count_of_number_of_images_testing = []

for images in list_of_training_classes:
    path = train_dir + '/' + images
    count_of_number_of_images_training.append(len(os.listdir(path)))

for images in list_of_testing_classes:
    path = test_dir + '/' + images
    count_of_number_of_images_testing.append(len(os.listdir(path)))

names_of_tumors = list_of_training_classes

X_axis = np.arange(len(names_of_tumors))
print(X_axis)

plt.bar(X_axis - 0.2, count_of_number_of_images_training, 0.4, label = 'Training data'
plt.bar(X_axis + 0.2, count_of_number_of_images_testing, 0.4, label = 'Testing data')

plt.xticks(X_axis, names_of_tumors, rotation=45)
plt.xlabel("names of tumors")
plt.ylabel("count")
plt.title("Tumor types")
plt.legend()
plt.grid()
plt.show()
```

```
[0 1 2 3]
```




```
count_of_number_of_images_training = [826, 822, 395, 827]
count_of_number_of_images_testing = [100, 115, 105, 74]

names_of_tumors = ['glioma', 'meningioma', 'no tumor', 'pituitary']

x_ticks = ['training', 'testing']

# define figure
plt.figure(figsize=(10, 4))
# numerical x
x = np.arange(0, len(x_ticks))
# plot bars

plt.bar(x[0], count_of_number_of_images_training[0], width=0.2, color='#1D2F6F')
plt.bar(x[0], count_of_number_of_images_training[1], bottom=826, width=0.2, color='#83
plt.bar(x[0], 826+822+count_of_number_of_images_training[2], bottom=826+822, width=0.2
plt.bar(x[0], 826+822+395+count_of_number_of_images_training[3], bottom=822+826+395, w

plt.bar(x[1], count_of_number_of_images_testing[0], width=0.2, color='#1D2F6F')
plt.bar(x[1], count_of_number_of_images_testing[1], bottom=100, width=0.2, color='#839
plt.bar(x[1], count_of_number_of_images_testing[2], bottom=100+115, width=0.2, color='
plt.bar(x[1], count_of_number_of_images_testing[3], bottom=100+115+105, width=0.2, col
'''
plt.bar(x[0], count_of_number_of_images_training[0], width=0.2)
plt.bar(x[0], count_of_number_of_images_training[1], bottom=826, width=0.2)
plt.bar(x[0], 826+822+count_of_number_of_images_training[2], bottom=826+822, width=0.2
plt.bar(x[0], 826+822+395+count_of_number_of_images_training[3], bottom=822+826+395, w

plt.bar(x[1], count_of_number_of_images_testing[0], width=0.2)
plt.bar(x[1], count_of_number_of_images_testing[1], bottom=100, width=0.2)
plt.bar(x[1], count_of_number_of_images_testing[2], bottom=100+115, width=0.2)
plt.bar(x[1], count_of_number_of_images_testing[3], bottom=100+115+105, width=0.2)
'''
plt.xticks(x, x_ticks)
```

```
# title and legend
plt.title('Tumor distribution', loc ='left')
plt.legend(['glioma', 'meningioma', 'no', 'pituitary'], ncol = 4)
plt.show()
```

Training the model

Here our model is being trained, and we are saving the model as "user_defined_model.h5"

```
dense_model.compile(optimizer = tf.optimizers.Adam(),   # optimizer for the learning p
                    loss = 'categorical_crossentropy',  # categorical : because one ho
                    metrics=['accuracy'])

dense_model_with_dropout.compile(optimizer = tf.optimizers.Adam(),   # optimizer for t
                    loss = 'categorical_crossentropy',  # categorical : because one ho
                    metrics=['accuracy'])

model_vgg16.compile(optimizer = tf.optimizers.Adam(),   # optimizer for the learning p
                    loss = 'categorical_crossentropy',  # categorical : because one ho
                    metrics=['accuracy'])


history_dense_model = dense_model.fit_generator(train_generator,
                                                steps_per_epoch = train_generator.samp
                                                validation_data = validation_generator
                                                validation_steps = validation_generato
                                                epochs = epochs)

# to save the trained model in .h5 file
dense_model.save('/content/dense_model.h5')
```

```
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Mo
      """
```

```
Epoch 1/15
67/67 [==============================] - 534s 8s/step - loss: 3.3866 - accuracy:
Epoch 2/15
67/67 [==============================] - 58s 873ms/step - loss: 1.0775 - accurac
Epoch 3/15
67/67 [==============================] - 58s 871ms/step - loss: 1.0631 - accurac
Epoch 4/15
67/67 [==============================] - 59s 877ms/step - loss: 1.0035 - accurac
Epoch 5/15
67/67 [==============================] - 58s 872ms/step - loss: 1.0326 - accurac
Epoch 6/15
67/67 [==============================] - 59s 875ms/step - loss: 0.9383 - accurac
Epoch 7/15
67/67 [==============================] - 58s 871ms/step - loss: 0.9270 - accurac
Epoch 8/15
67/67 [==============================] - 58s 872ms/step - loss: 0.9195 - accurac
Epoch 9/15
67/67 [==============================] - 59s 875ms/step - loss: 0.8814 - accurac
Epoch 10/15
67/67 [==============================] - 58s 873ms/step - loss: 0.8695 - accurac
Epoch 11/15
67/67 [==============================] - 58s 867ms/step - loss: 0.8330 - accurac
Epoch 12/15
67/67 [==============================] - 58s 870ms/step - loss: 0.8687 - accurac
Epoch 13/15
67/67 [==============================] - 58s 872ms/step - loss: 0.8339 - accurac
Epoch 14/15
67/67 [==============================] - 58s 870ms/step - loss: 0.8291 - accurac
Epoch 15/15
67/67 [==============================] - 58s 871ms/step - loss: 0.8270 - accurac
```

```
history_dense_model_with_dropout = dense_model_with_dropout.fit_generator(train_genera
                                                                           steps_per_ep
                                                                           validation_d
                                                                           validation_s
                                                                           epochs = epo

# to save the trained model in .h5 file
dense_model_with_dropout.save('/content/dense_model_with_dropout.h5')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Mo
  """
Epoch 1/15
67/67 [==============================] - 60s 889ms/step - loss: 3.1200 - accurac
Epoch 2/15
67/67 [==============================] - 58s 872ms/step - loss: 0.9956 - accurac
Epoch 3/15
67/67 [==============================] - 58s 869ms/step - loss: 0.9299 - accurac
Epoch 4/15
67/67 [==============================] - 58s 873ms/step - loss: 0.8762 - accurac
Epoch 5/15
67/67 [==============================] - 58s 868ms/step - loss: 0.8638 - accurac
Epoch 6/15
67/67 [==============================] - 59s 875ms/step - loss: 0.8645 - accurac
```

```
Epoch 7/15
67/67 [==============================] - 59s 878ms/step - loss: 0.8428 - accuracy
Epoch 8/15
67/67 [==============================] - 58s 873ms/step - loss: 0.8298 - accuracy
Epoch 9/15
67/67 [==============================] - 58s 872ms/step - loss: 0.8232 - accuracy
Epoch 10/15
67/67 [==============================] - 58s 868ms/step - loss: 0.8020 - accuracy
Epoch 11/15
67/67 [==============================] - 58s 868ms/step - loss: 0.8150 - accuracy
Epoch 12/15
67/67 [==============================] - 58s 862ms/step - loss: 0.7968 - accuracy
Epoch 13/15
67/67 [==============================] - 58s 863ms/step - loss: 0.7837 - accuracy
Epoch 14/15
67/67 [==============================] - 58s 860ms/step - loss: 0.7561 - accuracy
Epoch 15/15
67/67 [==============================] - 57s 859ms/step - loss: 0.7812 - accuracy
```

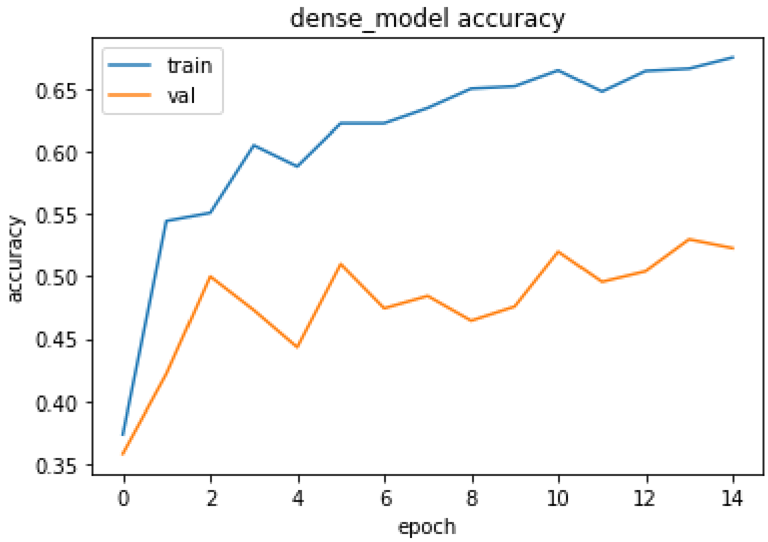
```
history_model_vgg16 = model_vgg16.fit_generator(train_generator,
                                                steps_per_epoch = train_generator.samp
                                                validation_data = validation_generator
                                                validation_steps = validation_generato
                                                epochs = epochs)

# to save the trained model in .h5 file
model_vgg16.save('/content/model_vgg16.h5')
```

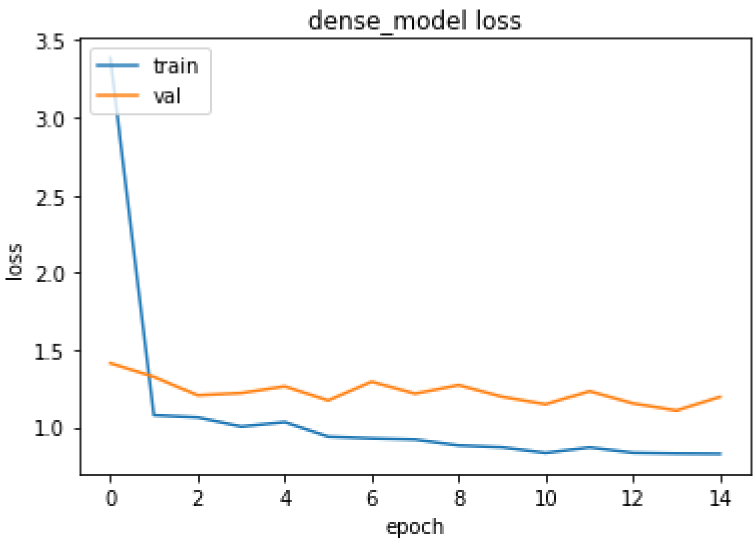
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Mod
  """
Epoch 1/15
67/67 [==============================] - 146s 2s/step - loss: 1.4495 - accuracy:
Epoch 2/15
67/67 [==============================] - 105s 2s/step - loss: 1.3511 - accuracy:
Epoch 3/15
67/67 [==============================] - 105s 2s/step - loss: 1.3499 - accuracy:
Epoch 4/15
67/67 [==============================] - 105s 2s/step - loss: 1.3520 - accuracy:
Epoch 5/15
67/67 [==============================] - 105s 2s/step - loss: 1.3501 - accuracy:
Epoch 6/15
67/67 [==============================] - 105s 2s/step - loss: 1.3518 - accuracy:
Epoch 7/15
67/67 [==============================] - 105s 2s/step - loss: 1.3505 - accuracy:
Epoch 8/15
67/67 [==============================] - 105s 2s/step - loss: 1.3497 - accuracy:
Epoch 9/15
67/67 [==============================] - 105s 2s/step - loss: 1.3501 - accuracy:
Epoch 10/15
67/67 [==============================] - 105s 2s/step - loss: 1.3503 - accuracy:
Epoch 11/15
67/67 [==============================] - 105s 2s/step - loss: 1.3512 - accuracy:
Epoch 12/15
67/67 [==============================] - 105s 2s/step - loss: 1.3489 - accuracy:
```

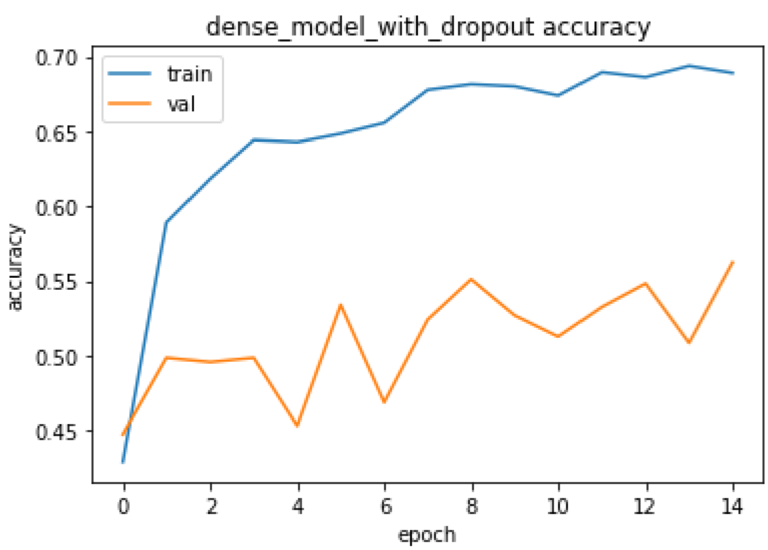
```
Epoch 13/15
67/67 [==============================] - 105s 2s/step - loss: 1.3495 - accuracy:
Epoch 14/15
67/67 [==============================] - 105s 2s/step - loss: 1.3492 - accuracy:
Epoch 15/15
67/67 [==============================] - 105s 2s/step - loss: 1.3503 - accuracy:
```

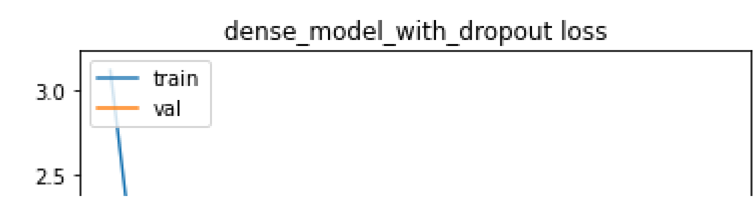
```
# plotting the performance of the model from the history object obtained after trainin

plt.plot(history_dense_model.history['accuracy'])
plt.plot(history_dense_model.history['val_accuracy'])
plt.title('dense_model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history_dense_model.history['loss'])
plt.plot(history_dense_model.history['val_loss'])
plt.title('dense_model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()


# plotting the performance of the model from the history object obtained after trainin

plt.plot(history_dense_model_with_dropout.history['accuracy'])
plt.plot(history_dense_model_with_dropout.history['val_accuracy'])
plt.title('dense_model_with_dropout accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history_dense_model_with_dropout.history['loss'])
plt.plot(history_dense_model_with_dropout.history['val_loss'])
plt.title('dense_model_with_dropout loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()


# plotting the performance of the model from the history object obtained after trainin

plt.plot(history_model_vgg16.history['accuracy'])
plt.plot(history_model_vgg16.history['val_accuracy'])
plt.title('model_vgg16 accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history_model_vgg16.history['loss'])
plt.plot(history_model_vgg16.history['val_loss'])
plt.title('model_vgg16 loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```










```
# evaluation code
```

```
dense_model.evaluate(test_generator)
```

```
13/13 [==============================] - 74s 6s/step - loss: 2.5000 - accuracy: 
[2.5000112056732178, 0.4314720928668976]
```

```
dense_model_with_dropout.evaluate(test_generator)
```

```
13/13 [==============================] - 2s 180ms/step - loss: 2.7540 - accuracy
[2.7540407180786133, 0.4238578677177429]
```

```
model_vgg16.evaluate(test_generator)
```

```
13/13 [==============================] - 11s 859ms/step - loss: 1.4497 - accurac
[1.449723243713379, 0.25380709767341614]
```

References :

https://www.kaggle.com/sartajbhuvaji/brain-tumor-classification-mri?select=Training
https://cs231n.github.io/convolutional-networks/
https://stats.stackexchange.com/questions/201569/what-is-the-difference-between-dropout-and-drop-connect
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout
https://arxiv.org/pdf/1512.03385.pdf
https://netron.app/
https://www.google.com/url?sa=i&url=https%3A%2F%2Fmedium.com%2Fdatatype%2Fdeeplearning-ai-cnn-week-2-97e075f8c801&psig=AOvVaw1t_6hriWqI__Z0IdOOGkdI&ust=1633887674153000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCJD84YXwvfMCFQAAAAAdAAAAABAJ
https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy
https://arxiv.org/pdf/1409.1556.pdf
https://www.kaggle.com/bonhart/brain-mri-data-visualization-unet-fpn
https://www.tensorflow.org/tutorials/images/transfer_learning