Brain Tumor Classification

Context

Brain tumor is known to be one of the most aggressive diseases that affect both children and adults. Of all primary Central Nervous System (CNS) tumors, brain tumors account for 85 to 90 percent. **Around 11,700 individuals are diagnosed with a brain tumor every year.** For individuals with a cancerous brain or CNS tumor, **the 5-year survival rate is around 34 percent for men and 36 percent for women.** Brain tumors are classified into Benign Tumors, Pituitary Tumors, Malignant Tumors etc. In order to increase the life expectancy of patients, adequate care, preparation and reliable diagnostics are required in the treatment process.

Magnetic Resonance Imaging (MRI) is the best way to identify brain tumors. A huge amount of image data is produced through MRI Scans. However, there are several anomalies in the tumor size and location (s). This makes it very difficult to completely comprehend the nature of the tumor. **A trained neurosurgeon is usually needed for MRI image analysis. The lack of qualified doctors and the lack of knowledge about tumors makes it very difficult and time-consuming for clinical facilities in developing countries to perform MRI studies.** Due to the level of difficulty involved in comprehending the nature of brain tumors and their properties, manual analyses can be highly error-prone. That makes an automated MRI analysis system crucial to solve this problem.

Applications of automated classification techniques using Machine Learning (ML) and Artificial Intelligence (AI) algorithms have consistently shown better performance than manual classification. It would therefore be highly beneficial to write an algorithm that performs **detection and classification of brain tumors using Deep Learning Algorithms.**

Dataset

The dataset folder contains MRI data. The images are already split into Training and Testing folders. Each folder has more four subfolders named **`"glioma_tumor", "meningioma_tumor", "no_tumor"`** and **`"pituitary_tumor"`**. These folders have MRI images of the respective tumor classes.

Instructions to access the data through Google Colab:

Follow the below steps:

1) Download the zip file from Olympus and extract it in your local system.

2) Upload the extracted folder into your drive.

3) Mount your Google Drive using the code below.

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

4) Now, you can read the dataset as mentioned in the code below.

Problem Statement

To build a classification model that can take images of MRI scans as input and can classify them into one of the following types of tumor:

`glioma_tumor`, **`meningioma_tumor`**, **`pituitary_tumor`** and **`no_tumor`**.

In [ ]:

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

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

Importing the libraries

In [ ]:

```python
#Reading the training images from the path and labelling them into the given categories
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import os
import seaborn as sns # for data visualization
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential #sequential api for sequential model
```

```
from tensorflow.keras.layers import Dense, Dropout, Flatten #importing different layers
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Activation, Input, LeakyReLU,Activa
from tensorflow.keras import backend as K
from tensorflow.keras.utils import to_categorical #to perform one-hot encoding
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop,Adam #optimiers for optimizing the model
from keras.callbacks import EarlyStopping  #regularization method to prevent the overfitting
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras import losses, optimizers
```

Reading the Training Data

In [ ]:

```
DATADIR = r"/content/drive/MyDrive/Data Set Brain Tumor/Training"
CATEGORIES = ["glioma_tumor","meningioma_tumor","no_tumor","pituitary_tumor"]

#Storing all the training images
training_data = []

def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR,category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)# Converting image to greyscal
                new_array = cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
                training_data.append([new_array,class_num])
            except Exception as e:
                pass
create_training_data()
```

Reading the Testing Dataset

In [ ]:

```
DATADIR = r"/content/drive/MyDrive/Data Set Brain Tumor/Testing"
CATEGORIES = ["glioma_tumor","meningioma_tumor","no_tumor","pituitary_tumor"]

#Storing all the training images
testing_data = []

def create_testing_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR,category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)# Converting image to greyscal
                new_array = cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
                testing_data.append([new_array,class_num])
            except Exception as e:
                pass
create_testing_data()
```

Data Preprocessing

In [ ]:

```
# Separating the images and labels
X_train = []
y_train = []
np.random.shuffle(training_data)
for features,label in training_data:
    X_train.append(features)
    y_train.append(label)
X_train= np.array(X_train)
print(X_train.shape)

# Normalizing pixel values
X_train = X_train/255.0
# image reshaping
X_train = X_train.reshape(-1,150,150,1)
```

```
(2881, 150, 150)
```

In [ ]:

```
X_test = []
y_test = []

np.random.shuffle(testing_data)
IMG_SIZE = 150
```

```
for features,label in testing_data:
    X_test.append(features)
    y_test.append(label)
X_test = np.array(X_test).reshape(-1,IMG_SIZE,IMG_SIZE)
print(X_test.shape)
X_test = X_test/255.0
X_test = X_test.reshape(-1,150,150,1)
```

```
(402, 150, 150)
```

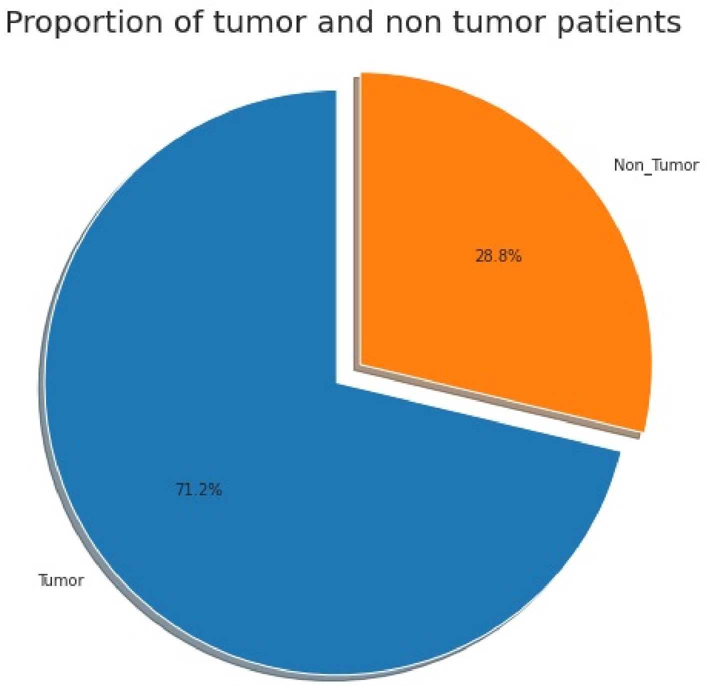
Exploratory Data Analysis

In [ ]:

```
#creating the dataframe to plot the pie chart
df=pd.DataFrame(y_train,columns=['Suffering'])
```

In [ ]:

```
#plotting the pie chart
labels = 'Tumor', 'Non_Tumor'
sizes = [df.Suffering[df['Suffering']!=0].count(), df.Suffering[df['Suffering']==0].count()]
explode = (0, 0.1)
fig1, ax1 = plt.subplots(figsize=(10, 8))
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.title("Proportion of tumor and non tumor patients ", size = 20)
plt.show()
```



The above plot shows that **this dataset is imbalanced**, because **71.2% of the images are those of tumors** - Majority class: glioma, meningioma and pituitary tumors, while approx. **28.8% of the images** belong to the **non-tumor category** (Minority class).

Let's visualize MRI images randomly from each of the three classes. The Image matrix is plotted and each row represents three single channel images corresponding to one class. We have read single channel images in order to reduce complexity.

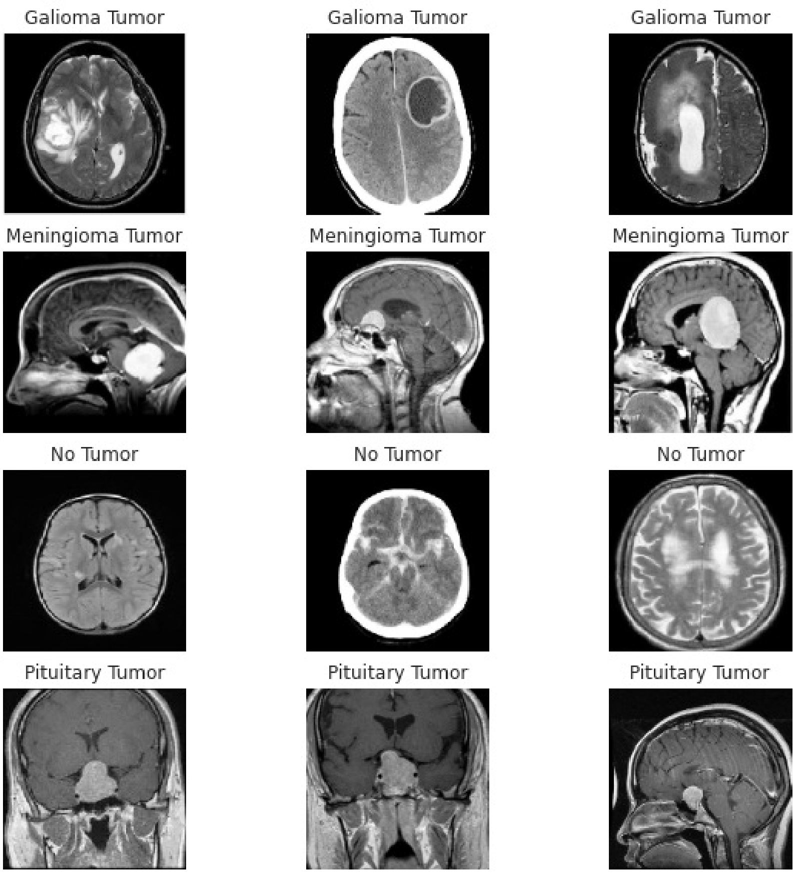
In [ ]:

```
#train_dir = 'DATA/train' # image folder
import os
# get the list of jpegs from sub image class folders
glioma_tumor_imgs = [fn for fn in os.listdir(f'{DATADIR}/{CATEGORIES[0]}') ]
meningioma_tumor_imgs = [fn for fn in os.listdir(f'{DATADIR}/{CATEGORIES[1]}') ]
no_tumor_imgs = [fn for fn in os.listdir(f'{DATADIR}/{CATEGORIES[2]}') ]
pituitary_tumor_imgs = [fn for fn in os.listdir(f'{DATADIR}/{CATEGORIES[3]}') ]

# randomly select 3 of each
select_gal = np.random.choice(glioma_tumor_imgs, 3, replace = False)
select_menin = np.random.choice(meningioma_tumor_imgs, 3, replace = False)
select_no_t = np.random.choice(no_tumor_imgs, 3, replace = False)
select_pit = np.random.choice(pituitary_tumor_imgs, 3, replace = False)
```

In [ ]:

```
from keras.preprocessing import image
```

```python
from keras.preprocessing import image
# plotting 2 x 3 image matrix
fig = plt.figure(figsize = (10,10))
for i in range(12):
    if i < 3:
        fp = f'{DATADIR}/{CATEGORIES[0]}/{select_gal[i]}'
        label = 'Galioma Tumor'
    if i>=3 and i<6:
        fp = f'{DATADIR}/{CATEGORIES[1]}/{select_menin[i-3]}'
        label = 'Meningioma Tumor'
    if i>=6 and i<9:
        fp = f'{DATADIR}/{CATEGORIES[2]}/{select_no_t[i-6]}'
        label = 'No Tumor'
    if i>=9 and i<12:
        fp = f'{DATADIR}/{CATEGORIES[3]}/{select_pit[i-9]}'
        label = 'Pituitary Tumor'
    ax = fig.add_subplot(4, 3, i+1)

    # to plot without rescaling, remove target_size
    fn = image.load_img(fp, target_size = (150,150), color_mode='grayscale')
    plt.imshow(fn, cmap='Greys_r')
    plt.title(label)
    plt.axis('off')
plt.show()

# also check the number of files here
```



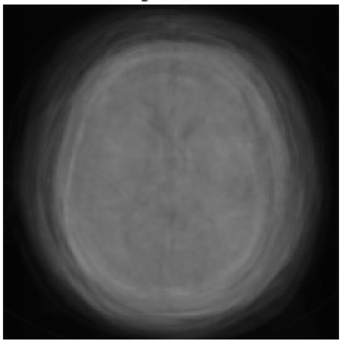
Finding the mean images for each class of tumor:

In [ ]:

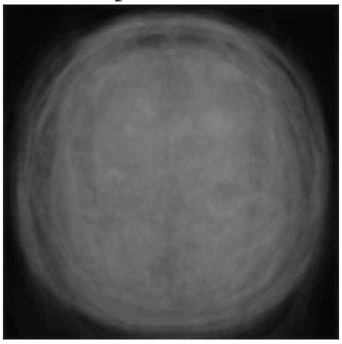
```python
def find_mean_img(full_mat, title):
    # calculate the average
    mean_img = np.mean(full_mat, axis = 0)
    # reshape it back to a matrix
    mean_img = mean_img.reshape((150,150))
    plt.imshow(mean_img, vmin=0, vmax=255, cmap='Greys_r')
    plt.title(f'Average {title}')
    plt.axis('off')
    plt.show()
    return mean_img

galioma_data=[]
menin_data=[]

no_tumor_data=[]

pitu_data=[]

for cat in CATEGORIES:
        path = os.path.join(DATADIR,cat)
```

```
        for img in os.listdir(path):

                img_array = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)# Converting image to greyscal
                new_array = cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
                if cat==CATEGORIES[0]:
                      galioma_data.append([new_array])
                if cat==CATEGORIES[1]:
                      menin_data.append([new_array])
                if cat==CATEGORIES[2]:
                      no_tumor_data.append([new_array])
                if cat==CATEGORIES[3]:
                      pitu_data.append([new_array])



norm_mean = find_mean_img(np.array(no_tumor_data), 'No Tumor')
gali_mean = find_mean_img(np.array(galioma_data), 'Galioma Tumor')
menin_mean = find_mean_img(np.array(menin_data), 'Meningioma Tumor')
Pitu_mean = find_mean_img(np.array(pitu_data), 'Pituitary Tumor')
```

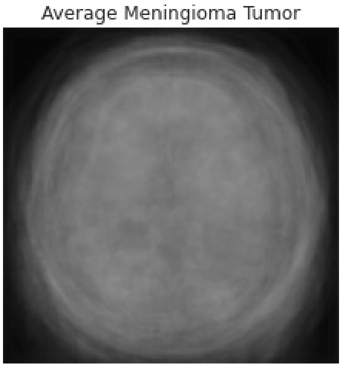
Average No Tumor



Average Galioma Tumor



Average Meningioma Tumor



Average Pituitary Tumor

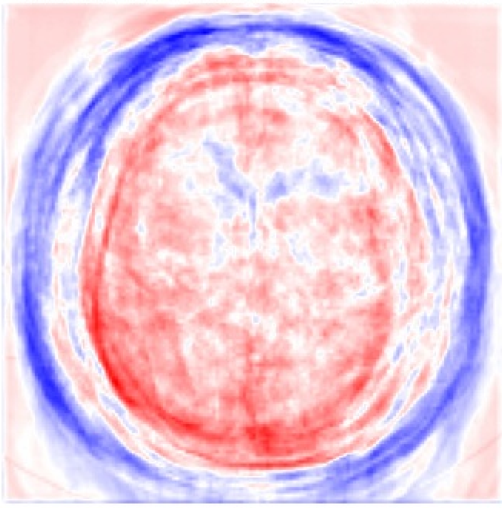
Contrast Difference

Contrast Difference

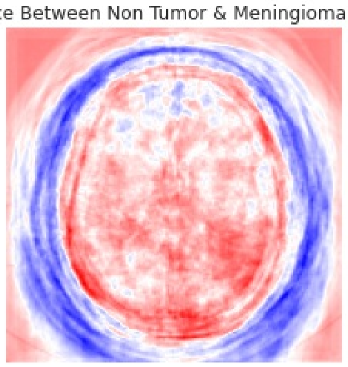
In [ ]:

```python
fig = plt.figure(figsize = (8,6))

for i in enumerate([gali_mean,menin_mean,Pitu_mean]):

      contrast_mean = norm_mean - i[1]
      plt.imshow(contrast_mean, cmap='bwr')
      if i[0]==0:
          plt.title(f'Difference Between Non Tumor & Galioma Average')
      if i[0]==1:
          plt.title(f'Difference Between Non Tumor & Meningioma Average')
      if i[0]==2:
          plt.title(f'Difference Between Non Tumor & Pituitary Average')
      plt.axis('off')
      plt.show()
```

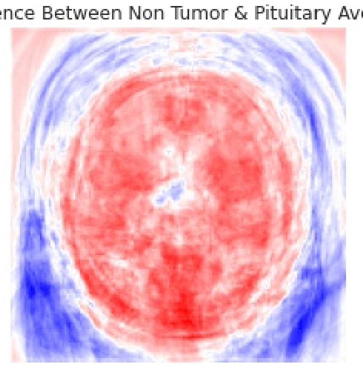
Difference Between Non Tumor & Galioma Average



Difference Between Non Tumor & Meningioma Average



Difference Between Non Tumor & Pituitary Average



As we can see from the contrast difference between the No tumor image and the tumor image, the blue area represents the negative values, meaning the size of the tumorous brain is more than non-tumourous brain.

One-Hot Encoding

In [ ]:

```python
encoded = to_categorical(np.array(y_train))
y_train_e=encoded
encoded_test = to_categorical(np.array(y_test))
y_test_e=encoded_test
```

In [ ]:

```
print(y_train_e.shape)
print(y_test_e.shape)
```

```
(2881, 4)
(402, 4)
```

Model Building

We will be using two types of Deep Neural Networks:

- **ANN** (Artificial Neural Network - fully connected)
- **CNN** (Convolutional Neural Network)

ANN

In [ ]:

```
#Build the model
# 3 layers, 1 layer to flatten the image to a 28 x 28 = 784 vector
#           1 layer with 128 neurons and relu function
#           1 layer with 10 neurons and softmax function
#Create the neural network model
def create_model():
        model_ann = keras.Sequential([
            keras.layers.Flatten(input_shape=(150,150)),
            keras.layers.Dense(500,kernel_initializer='he_uniform', activation=tf.nn.relu),
            keras.layers.Dense(700,kernel_initializer='he_uniform', activation=tf.nn.relu),
            keras.layers.Dense(4, kernel_initializer='random_uniform',activation=tf.nn.softmax)
        ])
        #Compile the model
        #The loss function measures how well the model did on training , and then tries
        #to improve on it using the optimizer
        model_ann.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        return model_ann
```

In [ ]:

```
model_ann=create_model()
model_ann.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 22500)             0
_________________________________________________________________
dense_12 (Dense)             (None, 500)               11250500
_________________________________________________________________
dense_13 (Dense)             (None, 700)               350700
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 2804
=================================================================
Total params: 11,604,004
Trainable params: 11,604,004
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
#Train the model
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
history=model_ann.fit(X_train,
          y_train_e,  #It expects integers because of the sparse_categorical_crossentropy loss function
          epochs=200, #number of iterations over the entire dataset to train on
          batch_size=64,validation_split=0.20,callbacks=[es, mc],use_multiprocessing=True)#number of samples per
```

```
Epoch 1/200
36/36 [==============================] - 4s 105ms/step - loss: 3.3193 - accuracy: 0.3551 - val_loss: 1.0074 - val
_accuracy: 0.6014

Epoch 00001: val_accuracy improved from -inf to 0.60139, saving model to best_model.h5
Epoch 2/200
36/36 [==============================] - 4s 102ms/step - loss: 0.8979 - accuracy: 0.6244 - val_loss: 0.8897 - val
_accuracy: 0.6499
```

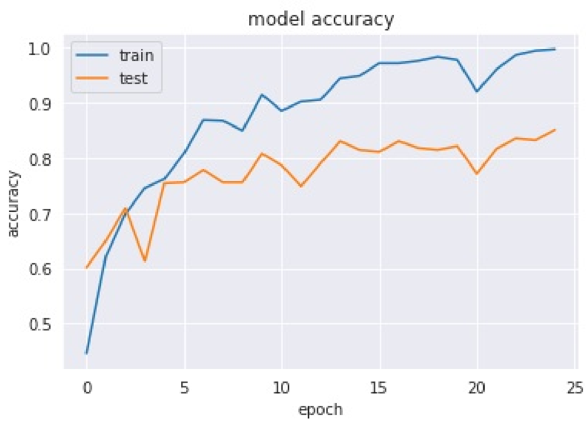
```
Epoch 00002: val_accuracy improved from 0.60139 to 0.64991, saving model to best_model.h5
Epoch 3/200
36/36 [==============================] - 4s 102ms/step - loss: 0.7619 - accuracy: 0.6962 - val_loss: 0.7694 - val
_accuracy: 0.7088

Epoch 00003: val_accuracy improved from 0.64991 to 0.70884, saving model to best_model.h5
Epoch 4/200
36/36 [==============================] - 4s 101ms/step - loss: 0.6097 - accuracy: 0.7548 - val_loss: 0.8904 - val
_accuracy: 0.6135

Epoch 00004: val_accuracy did not improve from 0.70884
Epoch 5/200
36/36 [==============================] - 4s 99ms/step - loss: 0.5722 - accuracy: 0.7432 - val_loss: 0.6688 - val_
accuracy: 0.7539

Epoch 00005: val_accuracy improved from 0.70884 to 0.75390, saving model to best_model.h5
Epoch 6/200
36/36 [==============================] - 4s 101ms/step - loss: 0.4647 - accuracy: 0.7991 - val_loss: 0.6752 - val
_accuracy: 0.7556

Epoch 00006: val_accuracy improved from 0.75390 to 0.75563, saving model to best_model.h5
Epoch 7/200
36/36 [==============================] - 4s 104ms/step - loss: 0.3324 - accuracy: 0.8769 - val_loss: 0.7271 - val
_accuracy: 0.7782

Epoch 00007: val_accuracy improved from 0.75563 to 0.77816, saving model to best_model.h5
Epoch 8/200
36/36 [==============================] - 4s 103ms/step - loss: 0.2997 - accuracy: 0.8852 - val_loss: 0.7827 - val
_accuracy: 0.7556

Epoch 00008: val_accuracy did not improve from 0.77816
Epoch 9/200
36/36 [==============================] - 4s 100ms/step - loss: 0.3717 - accuracy: 0.8438 - val_loss: 0.7442 - val
_accuracy: 0.7556

Epoch 00009: val_accuracy did not improve from 0.77816
Epoch 10/200
36/36 [==============================] - 4s 100ms/step - loss: 0.2557 - accuracy: 0.8944 - val_loss: 0.6994 - val
_accuracy: 0.8076

Epoch 00010: val_accuracy improved from 0.77816 to 0.80763, saving model to best_model.h5
Epoch 11/200
36/36 [==============================] - 4s 103ms/step - loss: 0.2493 - accuracy: 0.9045 - val_loss: 0.7145 - val
_accuracy: 0.7868

Epoch 00011: val_accuracy did not improve from 0.80763
Epoch 12/200
36/36 [==============================] - 4s 99ms/step - loss: 0.2031 - accuracy: 0.9190 - val_loss: 1.1767 - val_
accuracy: 0.7487

Epoch 00012: val_accuracy did not improve from 0.80763
Epoch 13/200
36/36 [==============================] - 4s 100ms/step - loss: 0.2665 - accuracy: 0.9026 - val_loss: 0.7499 - val
_accuracy: 0.7903

Epoch 00013: val_accuracy did not improve from 0.80763
Epoch 14/200
36/36 [==============================] - 4s 100ms/step - loss: 0.2062 - accuracy: 0.9258 - val_loss: 0.7135 - val
_accuracy: 0.8302

Epoch 00014: val_accuracy improved from 0.80763 to 0.83016, saving model to best_model.h5
Epoch 15/200
36/36 [==============================] - 4s 105ms/step - loss: 0.1206 - accuracy: 0.9604 - val_loss: 0.7134 - val
_accuracy: 0.8146

Epoch 00015: val_accuracy did not improve from 0.83016
Epoch 16/200
36/36 [==============================] - 4s 100ms/step - loss: 0.0925 - accuracy: 0.9702 - val_loss: 0.7695 - val
_accuracy: 0.8111

Epoch 00016: val_accuracy did not improve from 0.83016
Epoch 17/200
36/36 [==============================] - 4s 100ms/step - loss: 0.0729 - accuracy: 0.9777 - val_loss: 0.7261 - val
_accuracy: 0.8302

Epoch 00017: val_accuracy did not improve from 0.83016
Epoch 18/200
36/36 [==============================] - 4s 99ms/step - loss: 0.0741 - accuracy: 0.9786 - val_loss: 0.7629 - val_
accuracy: 0.8180

Epoch 00018: val_accuracy did not improve from 0.83016
Epoch 19/200
36/36 [==============================] - 4s 100ms/step - loss: 0.0520 - accuracy: 0.9855 - val_loss: 0.8668 - val
```

```
_accuracy: 0.8146

Epoch 00019: val_accuracy did not improve from 0.83016
Epoch 20/200
36/36 [==============================] - 4s 100ms/step - loss: 0.0663 - accuracy: 0.9783 - val_loss: 0.7929 - val
_accuracy: 0.8215

Epoch 00020: val_accuracy did not improve from 0.83016
Epoch 21/200
36/36 [==============================] - 4s 100ms/step - loss: 0.1997 - accuracy: 0.9391 - val_loss: 1.0446 - val
_accuracy: 0.7712

Epoch 00021: val_accuracy did not improve from 0.83016
Epoch 22/200
36/36 [==============================] - 4s 100ms/step - loss: 0.1164 - accuracy: 0.9513 - val_loss: 0.8863 - val
_accuracy: 0.8163

Epoch 00022: val_accuracy did not improve from 0.83016
Epoch 23/200
36/36 [==============================] - 4s 100ms/step - loss: 0.0605 - accuracy: 0.9793 - val_loss: 0.7774 - val
_accuracy: 0.8354

Epoch 00023: val_accuracy improved from 0.83016 to 0.83536, saving model to best_model.h5
Epoch 24/200
36/36 [==============================] - 4s 102ms/step - loss: 0.0274 - accuracy: 0.9953 - val_loss: 0.8244 - val
_accuracy: 0.8319

Epoch 00024: val_accuracy did not improve from 0.83536
Epoch 25/200
36/36 [==============================] - 4s 100ms/step - loss: 0.0149 - accuracy: 0.9975 - val_loss: 0.8311 - val
_accuracy: 0.8510

Epoch 00025: val_accuracy improved from 0.83536 to 0.85095, saving model to best_model.h5
Epoch 00025: early stopping
```

In [ ]:

```
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



In [ ]:

```
import numpy
model_ann.evaluate(X_test,y_test_e)
```

```
13/13 [==============================] - 0s 21ms/step - loss: 5.6083 - accuracy: 0.7065
```

Out[ ]:

```
[5.608260154724121, 0.7064676880836487]
```

As we see here, the **ANN does not show a good test accuracy**, since ANNs are unable to capture spatial correlation characteristics of the image.

Let's try Convolutional Neural Networks, which take in the whole image as a 2D matrix instead.

Convolutional Neural Network (CNN)

Model 1: CNN with Dropout

In [ ]:
```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
model = Sequential()
#
y_train=np.array(y_train)
model.add(Conv2D(filters = 64, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (150,150,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
#
model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
#
model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.3))
#
model.add(Conv2D(filters = 128, kernel_size = (2,2),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.3))

#
model.add(Conv2D(filters = 256, kernel_size = (2,2),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.3))

#
model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(4, activation = "softmax"))
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
epochs = 200
batch_size = 64



es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)

history=model.fit(X_train,
          y_train_e,  #It expects integers because of the sparse_categorical_crossentropy loss function
          epochs=30, #number of iterations over the entire dataset to train on
          batch_size=64,validation_split=0.20,callbacks=[es, mc],use_multiprocessing=True)#number of samples per
```

```
Epoch 1/30
36/36 [==============================] - 248s 7s/step - loss: 1.3788 - accuracy: 0.3474 - val_loss: 1.2375 - val_
accuracy: 0.4887

Epoch 00001: val_accuracy improved from -inf to 0.48873, saving model to best_model.h5
Epoch 2/30
36/36 [==============================] - 248s 7s/step - loss: 1.1186 - accuracy: 0.5234 - val_loss: 1.0731 - val_
accuracy: 0.5269

Epoch 00002: val_accuracy improved from 0.48873 to 0.52686, saving model to best_model.h5
Epoch 3/30
36/36 [==============================] - 245s 7s/step - loss: 0.8693 - accuracy: 0.6094 - val_loss: 0.7816 - val_
accuracy: 0.6603

Epoch 00003: val_accuracy improved from 0.52686 to 0.66031, saving model to best_model.h5
Epoch 4/30
36/36 [==============================] - 245s 7s/step - loss: 0.7257 - accuracy: 0.6775 - val_loss: 0.7234 - val_
accuracy: 0.6794

Epoch 00004: val_accuracy improved from 0.66031 to 0.67938, saving model to best_model.h5
Epoch 5/30
36/36 [==============================] - 246s 7s/step - loss: 0.6721 - accuracy: 0.7044 - val_loss: 0.6501 - val_
accuracy: 0.7123

Epoch 00005: val_accuracy improved from 0.67938 to 0.71231, saving model to best_model.h5
Epoch 6/30
36/36 [==============================] - 245s 7s/step - loss: 0.6198 - accuracy: 0.7445 - val_loss: 0.6855 - val_
```

```
accuracy: 0.7071

Epoch 00006: val_accuracy did not improve from 0.71231
Epoch 7/30
36/36 [==============================] - 247s 7s/step - loss: 0.5527 - accuracy: 0.7802 - val_loss: 0.5529 - val_
accuracy: 0.7678

Epoch 00007: val_accuracy improved from 0.71231 to 0.76776, saving model to best_model.h5
Epoch 8/30
36/36 [==============================] - 245s 7s/step - loss: 0.4773 - accuracy: 0.8126 - val_loss: 0.5864 - val_
accuracy: 0.7348

Epoch 00008: val_accuracy did not improve from 0.76776
Epoch 9/30
36/36 [==============================] - 245s 7s/step - loss: 0.4603 - accuracy: 0.8136 - val_loss: 0.4864 - val_
accuracy: 0.7834

Epoch 00009: val_accuracy improved from 0.76776 to 0.78336, saving model to best_model.h5
Epoch 10/30
36/36 [==============================] - 246s 7s/step - loss: 0.4241 - accuracy: 0.8310 - val_loss: 0.4254 - val_
accuracy: 0.8336

Epoch 00010: val_accuracy improved from 0.78336 to 0.83362, saving model to best_model.h5
Epoch 11/30
36/36 [==============================] - 245s 7s/step - loss: 0.3908 - accuracy: 0.8414 - val_loss: 0.4561 - val_
accuracy: 0.7938

Epoch 00011: val_accuracy did not improve from 0.83362
Epoch 12/30
36/36 [==============================] - 246s 7s/step - loss: 0.3299 - accuracy: 0.8647 - val_loss: 0.3920 - val_
accuracy: 0.8406

Epoch 00012: val_accuracy improved from 0.83362 to 0.84055, saving model to best_model.h5
Epoch 13/30
36/36 [==============================] - 245s 7s/step - loss: 0.2996 - accuracy: 0.8751 - val_loss: 0.3916 - val_
accuracy: 0.8458

Epoch 00013: val_accuracy improved from 0.84055 to 0.84575, saving model to best_model.h5
Epoch 14/30
36/36 [==============================] - 246s 7s/step - loss: 0.2962 - accuracy: 0.8826 - val_loss: 0.3829 - val_
accuracy: 0.8458

Epoch 00014: val_accuracy did not improve from 0.84575
Epoch 15/30
36/36 [==============================] - 246s 7s/step - loss: 0.2542 - accuracy: 0.9007 - val_loss: 0.3054 - val_
accuracy: 0.8908

Epoch 00015: val_accuracy improved from 0.84575 to 0.89081, saving model to best_model.h5
Epoch 16/30
36/36 [==============================] - 246s 7s/step - loss: 0.2264 - accuracy: 0.9130 - val_loss: 0.3509 - val_
accuracy: 0.8475

Epoch 00016: val_accuracy did not improve from 0.89081
Epoch 17/30
36/36 [==============================] - 246s 7s/step - loss: 0.1868 - accuracy: 0.9250 - val_loss: 0.3109 - val_
accuracy: 0.8700

Epoch 00017: val_accuracy did not improve from 0.89081
Epoch 18/30
36/36 [==============================] - 243s 7s/step - loss: 0.1561 - accuracy: 0.9411 - val_loss: 0.3432 - val_
accuracy: 0.8804

Epoch 00018: val_accuracy did not improve from 0.89081
Epoch 19/30
36/36 [==============================] - 243s 7s/step - loss: 0.2057 - accuracy: 0.9176 - val_loss: 0.2881 - val_
accuracy: 0.8787

Epoch 00019: val_accuracy did not improve from 0.89081
Epoch 20/30
36/36 [==============================] - 245s 7s/step - loss: 0.1566 - accuracy: 0.9411 - val_loss: 0.2539 - val_
accuracy: 0.9185

Epoch 00020: val_accuracy improved from 0.89081 to 0.91854, saving model to best_model.h5
Epoch 21/30
36/36 [==============================] - 244s 7s/step - loss: 0.1509 - accuracy: 0.9481 - val_loss: 0.3033 - val_
accuracy: 0.8908

Epoch 00021: val_accuracy did not improve from 0.91854
Epoch 22/30
36/36 [==============================] - 245s 7s/step - loss: 0.1453 - accuracy: 0.9537 - val_loss: 0.2714 - val_
accuracy: 0.9133

Epoch 00022: val_accuracy did not improve from 0.91854
```

```
Epoch 23/30
36/36 [==============================] - 244s 7s/step - loss: 0.1736 - accuracy: 0.9314 - val_loss: 0.2806 - val_
accuracy: 0.8943

Epoch 00023: val_accuracy did not improve from 0.91854
Epoch 24/30
36/36 [==============================] - 243s 7s/step - loss: 0.1639 - accuracy: 0.9385 - val_loss: 0.2538 - val_
accuracy: 0.9220

Epoch 00024: val_accuracy improved from 0.91854 to 0.92201, saving model to best_model.h5
Epoch 25/30
36/36 [==============================] - 245s 7s/step - loss: 0.0928 - accuracy: 0.9623 - val_loss: 0.2578 - val_
accuracy: 0.9151

Epoch 00025: val_accuracy did not improve from 0.92201
Epoch 26/30
36/36 [==============================] - 245s 7s/step - loss: 0.0979 - accuracy: 0.9626 - val_loss: 0.2342 - val_
accuracy: 0.9116

Epoch 00026: val_accuracy did not improve from 0.92201
Epoch 27/30
36/36 [==============================] - 246s 7s/step - loss: 0.0972 - accuracy: 0.9649 - val_loss: 0.2916 - val_
accuracy: 0.9064

Epoch 00027: val_accuracy did not improve from 0.92201
Epoch 28/30
36/36 [==============================] - 245s 7s/step - loss: 0.0894 - accuracy: 0.9704 - val_loss: 0.2201 - val_
accuracy: 0.9272

Epoch 00028: val_accuracy improved from 0.92201 to 0.92721, saving model to best_model.h5
Epoch 29/30
36/36 [==============================] - 245s 7s/step - loss: 0.0770 - accuracy: 0.9732 - val_loss: 0.3797 - val_
accuracy: 0.8683

Epoch 00029: val_accuracy did not improve from 0.92721
Epoch 30/30
36/36 [==============================] - 245s 7s/step - loss: 0.0858 - accuracy: 0.9692 - val_loss: 0.2176 - val_
accuracy: 0.9376

Epoch 00030: val_accuracy improved from 0.92721 to 0.93761, saving model to best_model.h5
```

In [ ]:

```
model.evaluate(X_test,np.array(y_test_e))
```

```
13/13 [==============================] - 11s 807ms/step - loss: 4.0493 - accuracy: 0.6915
```

Out[ ]:

```
[4.049304485321045, 0.6915422677993774]
```

Convolutional Neural Network (CNN)

Model 2: CNN with Dropout after Convolution and having two Dense layers with 16 & 8 units respectively

Since CNN Model 1 does not appear to have good test accuracy and appears to be overfitting on the training dataset, let's use CNN Model 2, which has a different architecture that should generalize well and not overfit.

In [ ]:

```
class conv_Layers:

  def __init__(self, nfilters, kernel_size, stride=1,
               pool_size=2, leakyrelu_slope=0.1, dropc=0.0, bnorm=False):
    self.nfilters = nfilters
    self.kernel_size = kernel_size
    self.stride = stride
    self.pool_size = pool_size
    self.leakyrelu_slope = leakyrelu_slope
    self.dropfrac = dropc
    self.bnorm = bnorm

  def __call__(self, x):
    x = Conv2D(self.nfilters, kernel_size=self.kernel_size,
               strides=self.stride, padding='same')(x)
    x = LeakyReLU(self.leakyrelu_slope)(x)
    if (self.dropfrac > 0.0):
      x = Dropout(self.dropfrac)(x)
    if (self.bnorm):
      x = BatchNormalization()(x)
    x = MaxPool2D(self.pool_size)(x)
    return x
```

```
class dense_Layers:

  def __init__(self, nunits, leakyrelu_slope=0.1, dropd=0.0, bnorm=False):
    self.nunits = nunits
    self.leakyrelu_slope = leakyrelu_slope
    self.dropfrac = dropd
    self.bnorm = bnorm

  def __call__(self, x):
    x = Dense(self.nunits)(x)
    x = LeakyReLU(self.leakyrelu_slope)(x)
    if (self.dropfrac > 0.0):
      x = Dropout(self.dropfrac)(x)
    if (self.bnorm):
      x = BatchNormalization()(x)
    return x

def LNmodel(in_shape, conv_filters, dense_filters, kernel_size, num_classes, lr,
            stride=1, pool_size=2, leakyrelu_slope=0.1, dropc=0.0, dropd=0.0, bnorm=False):

  in_shape = X_train.shape[1:]
  i = Input(shape=in_shape)
  for ncl, nconvfilters in enumerate(conv_filters):
    if (ncl==0):
      x = conv_Layers(nconvfilters, kernel_size,
                      stride, pool_size, leakyrelu_slope, dropc, bnorm)(i)
    else:
      x = conv_Layers(nconvfilters, kernel_size,
                      stride, pool_size, leakyrelu_slope, dropc, bnorm)(x)

  x = Flatten()(x)

  for ndl, ndunits in enumerate(dense_filters):
    x = dense_Layers(ndunits, leakyrelu_slope, dropd, bnorm)(x)

  x = Dense(num_classes, activation='softmax')(x)

  ln_model  = Model(inputs=i, outputs=x)
  adam = optimizers.Adam(lr=lr)
  ln_model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
  return ln_model
```

In [ ]:

```
lr = 0.001
kernelsize = 5
in_shape= X_train.shape[1:]
model_ln3 = LNmodel(in_shape, [8,16], [16,8], kernelsize, 4, lr,
                    stride=1, pool_size=2, leakyrelu_slope=0.1, dropc=0.25,
                    dropd=0.5, bnorm=False)
model_ln3.summary()
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 150, 150, 1)]     0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 150, 150, 8)       208
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)   (None, 150, 150, 8)       0
_________________________________________________________________
dropout_8 (Dropout)          (None, 150, 150, 8)       0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 75, 75, 8)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 75, 75, 16)        3216
_________________________________________________________________
leaky_re_lu_18 (LeakyReLU)   (None, 75, 75, 16)        0
_________________________________________________________________
dropout_9 (Dropout)          (None, 75, 75, 16)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 37, 37, 16)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 21904)             0
_________________________________________________________________
dense_12 (Dense)             (None, 16)                350480
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)   (None, 16)                0
_________________________________________________________________
dropout_10 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_13 (Dense)             (None, 8)                 136
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 8)                 0
_________________________________________________________________
dropout_11 (Dropout)         (None, 8)                 0
```

```
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 36
=================================================================
Total params: 354,076
Trainable params: 354,076
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)

history_model_ln3 = model_ln3.fit(X_train, y_train_e,
                                  validation_split=0.1,
                                  verbose=1, batch_size=256,
                                  shuffle=True, epochs=60,callbacks=[es,mc])
```

```
Epoch 1/60
11/11 [==============================] - 47s 4s/step - loss: 1.3705 - accuracy: 0.3164 - val_loss: 1.3379 - val_a
ccuracy: 0.5813

Epoch 00001: val_accuracy improved from -inf to 0.58131, saving model to best_model.h5
Epoch 2/60
11/11 [==============================] - 46s 4s/step - loss: 1.2839 - accuracy: 0.3731 - val_loss: 1.3757 - val_a
ccuracy: 0.3702

Epoch 00002: val_accuracy did not improve from 0.58131
Epoch 3/60
11/11 [==============================] - 45s 4s/step - loss: 1.2383 - accuracy: 0.4029 - val_loss: 1.2522 - val_a
ccuracy: 0.7612

Epoch 00003: val_accuracy improved from 0.58131 to 0.76125, saving model to best_model.h5
Epoch 4/60
11/11 [==============================] - 48s 4s/step - loss: 1.2123 - accuracy: 0.4020 - val_loss: 1.1360 - val_a
ccuracy: 0.8443

Epoch 00004: val_accuracy improved from 0.76125 to 0.84429, saving model to best_model.h5
Epoch 5/60
11/11 [==============================] - 45s 4s/step - loss: 1.1890 - accuracy: 0.4030 - val_loss: 1.1812 - val_a
ccuracy: 0.8304

Epoch 00005: val_accuracy did not improve from 0.84429
Epoch 6/60
11/11 [==============================] - 45s 4s/step - loss: 1.1371 - accuracy: 0.4491 - val_loss: 1.1282 - val_a
ccuracy: 0.7785

Epoch 00006: val_accuracy did not improve from 0.84429
Epoch 7/60
11/11 [==============================] - 45s 4s/step - loss: 1.1019 - accuracy: 0.4793 - val_loss: 1.0688 - val_a
ccuracy: 0.8201

Epoch 00007: val_accuracy did not improve from 0.84429
Epoch 8/60
11/11 [==============================] - 45s 4s/step - loss: 1.0705 - accuracy: 0.4792 - val_loss: 0.9978 - val_a
ccuracy: 0.7785

Epoch 00008: val_accuracy did not improve from 0.84429
Epoch 9/60
11/11 [==============================] - 45s 4s/step - loss: 1.0626 - accuracy: 0.4948 - val_loss: 1.0221 - val_a
ccuracy: 0.8166

Epoch 00009: val_accuracy did not improve from 0.84429
Epoch 10/60
11/11 [==============================] - 45s 4s/step - loss: 1.0476 - accuracy: 0.5062 - val_loss: 0.9183 - val_a
ccuracy: 0.8374

Epoch 00010: val_accuracy did not improve from 0.84429
Epoch 11/60
11/11 [==============================] - 45s 4s/step - loss: 1.0051 - accuracy: 0.5204 - val_loss: 1.0510 - val_a
ccuracy: 0.8097

Epoch 00011: val_accuracy did not improve from 0.84429
Epoch 12/60
11/11 [==============================] - 45s 4s/step - loss: 0.9978 - accuracy: 0.5194 - val_loss: 0.9376 - val_a
ccuracy: 0.8028

Epoch 00012: val_accuracy did not improve from 0.84429
Epoch 13/60
11/11 [==============================] - 46s 4s/step - loss: 0.9921 - accuracy: 0.5275 - val_loss: 0.8302 - val_a
ccuracy: 0.8997

Epoch 00013: val_accuracy improved from 0.84429 to 0.89965, saving model to best_model.h5
```

```
Epoch 14/60
11/11 [==============================] - 46s 4s/step - loss: 0.9412 - accuracy: 0.5499 - val_loss: 0.9101 - val_a
ccuracy: 0.7682

Epoch 00014: val_accuracy did not improve from 0.89965
Epoch 15/60
11/11 [==============================] - 46s 4s/step - loss: 0.9342 - accuracy: 0.5665 - val_loss: 0.7685 - val_a
ccuracy: 0.8512

Epoch 00015: val_accuracy did not improve from 0.89965
Epoch 16/60
11/11 [==============================] - 45s 4s/step - loss: 0.9022 - accuracy: 0.5663 - val_loss: 0.8105 - val_a
ccuracy: 0.8304

Epoch 00016: val_accuracy did not improve from 0.89965
Epoch 17/60
11/11 [==============================] - 46s 4s/step - loss: 0.8916 - accuracy: 0.5876 - val_loss: 0.8017 - val_a
ccuracy: 0.8478

Epoch 00017: val_accuracy did not improve from 0.89965
Epoch 18/60
11/11 [==============================] - 48s 4s/step - loss: 0.8820 - accuracy: 0.5792 - val_loss: 0.6499 - val_a
ccuracy: 0.8927

Epoch 00018: val_accuracy did not improve from 0.89965
Epoch 19/60
11/11 [==============================] - 45s 4s/step - loss: 0.8314 - accuracy: 0.6177 - val_loss: 0.7377 - val_a
ccuracy: 0.8651

Epoch 00019: val_accuracy did not improve from 0.89965
Epoch 20/60
11/11 [==============================] - 46s 4s/step - loss: 0.8453 - accuracy: 0.6254 - val_loss: 0.4269 - val_a
ccuracy: 0.9273

Epoch 00020: val_accuracy improved from 0.89965 to 0.92734, saving model to best_model.h5
Epoch 21/60
11/11 [==============================] - 45s 4s/step - loss: 0.8087 - accuracy: 0.6476 - val_loss: 0.5864 - val_a
ccuracy: 0.8997

Epoch 00021: val_accuracy did not improve from 0.92734
Epoch 22/60
11/11 [==============================] - 45s 4s/step - loss: 0.7597 - accuracy: 0.6588 - val_loss: 0.5198 - val_a
ccuracy: 0.8893

Epoch 00022: val_accuracy did not improve from 0.92734
Epoch 23/60
11/11 [==============================] - 45s 4s/step - loss: 0.7553 - accuracy: 0.6507 - val_loss: 0.6647 - val_a
ccuracy: 0.8339

Epoch 00023: val_accuracy did not improve from 0.92734
Epoch 24/60
11/11 [==============================] - 45s 4s/step - loss: 0.7433 - accuracy: 0.6638 - val_loss: 0.4627 - val_a
ccuracy: 0.9135

Epoch 00024: val_accuracy did not improve from 0.92734
Epoch 25/60
11/11 [==============================] - 45s 4s/step - loss: 0.7381 - accuracy: 0.6747 - val_loss: 0.4081 - val_a
ccuracy: 0.9273

Epoch 00025: val_accuracy did not improve from 0.92734
Epoch 26/60
11/11 [==============================] - 46s 4s/step - loss: 0.7241 - accuracy: 0.6709 - val_loss: 0.4913 - val_a
ccuracy: 0.8789

Epoch 00026: val_accuracy did not improve from 0.92734
Epoch 27/60
11/11 [==============================] - 46s 4s/step - loss: 0.6917 - accuracy: 0.6833 - val_loss: 0.5234 - val_a
ccuracy: 0.8754

Epoch 00027: val_accuracy did not improve from 0.92734
Epoch 28/60
11/11 [==============================] - 46s 4s/step - loss: 0.6781 - accuracy: 0.7068 - val_loss: 0.5845 - val_a
ccuracy: 0.8374

Epoch 00028: val_accuracy did not improve from 0.92734
Epoch 29/60
11/11 [==============================] - 45s 4s/step - loss: 0.6374 - accuracy: 0.7142 - val_loss: 0.5318 - val_a
ccuracy: 0.9100

Epoch 00029: val_accuracy did not improve from 0.92734
Epoch 30/60
11/11 [==============================] - 45s 4s/step - loss: 0.6560 - accuracy: 0.7117 - val_loss: 0.5590 - val_a
ccuracy: 0.8685
```

```
Epoch 00030: val_accuracy did not improve from 0.92734
Epoch 31/60
11/11 [==============================] - 45s 4s/step - loss: 0.6452 - accuracy: 0.7163 - val_loss: 0.4844 - val_a
ccuracy: 0.8789

Epoch 00031: val_accuracy did not improve from 0.92734
Epoch 32/60
11/11 [==============================] - 45s 4s/step - loss: 0.6472 - accuracy: 0.7076 - val_loss: 0.3750 - val_a
ccuracy: 0.8997

Epoch 00032: val_accuracy did not improve from 0.92734
Epoch 33/60
11/11 [==============================] - 45s 4s/step - loss: 0.6160 - accuracy: 0.7390 - val_loss: 0.4628 - val_a
ccuracy: 0.8685

Epoch 00033: val_accuracy did not improve from 0.92734
Epoch 34/60
11/11 [==============================] - 45s 4s/step - loss: 0.6095 - accuracy: 0.7236 - val_loss: 0.5180 - val_a
ccuracy: 0.8651

Epoch 00034: val_accuracy did not improve from 0.92734
Epoch 35/60
11/11 [==============================] - 45s 4s/step - loss: 0.6169 - accuracy: 0.7286 - val_loss: 0.4178 - val_a
ccuracy: 0.8893

Epoch 00035: val_accuracy did not improve from 0.92734
Epoch 36/60
11/11 [==============================] - 45s 4s/step - loss: 0.5921 - accuracy: 0.7427 - val_loss: 0.4645 - val_a
ccuracy: 0.8858

Epoch 00036: val_accuracy did not improve from 0.92734
Epoch 37/60
11/11 [==============================] - 45s 4s/step - loss: 0.5959 - accuracy: 0.7367 - val_loss: 0.2731 - val_a
ccuracy: 0.9377

Epoch 00037: val_accuracy improved from 0.92734 to 0.93772, saving model to best_model.h5
Epoch 38/60
11/11 [==============================] - 45s 4s/step - loss: 0.5583 - accuracy: 0.7574 - val_loss: 0.3996 - val_a
ccuracy: 0.8962

Epoch 00038: val_accuracy did not improve from 0.93772
Epoch 39/60
11/11 [==============================] - 45s 4s/step - loss: 0.5825 - accuracy: 0.7418 - val_loss: 0.3279 - val_a
ccuracy: 0.9343

Epoch 00039: val_accuracy did not improve from 0.93772
Epoch 40/60
11/11 [==============================] - 45s 4s/step - loss: 0.5306 - accuracy: 0.7879 - val_loss: 0.2734 - val_a
ccuracy: 0.9377

Epoch 00040: val_accuracy did not improve from 0.93772
Epoch 41/60
11/11 [==============================] - 46s 4s/step - loss: 0.5560 - accuracy: 0.7704 - val_loss: 0.3186 - val_a
ccuracy: 0.9273

Epoch 00041: val_accuracy did not improve from 0.93772
Epoch 42/60
11/11 [==============================] - 46s 4s/step - loss: 0.5405 - accuracy: 0.7716 - val_loss: 0.4026 - val_a
ccuracy: 0.9031

Epoch 00042: val_accuracy did not improve from 0.93772
Epoch 43/60
11/11 [==============================] - 46s 4s/step - loss: 0.5484 - accuracy: 0.7593 - val_loss: 0.3695 - val_a
ccuracy: 0.9066

Epoch 00043: val_accuracy did not improve from 0.93772
Epoch 44/60
11/11 [==============================] - 46s 4s/step - loss: 0.5058 - accuracy: 0.7827 - val_loss: 0.3230 - val_a
ccuracy: 0.9135

Epoch 00044: val_accuracy did not improve from 0.93772
Epoch 45/60
11/11 [==============================] - 45s 4s/step - loss: 0.5260 - accuracy: 0.7697 - val_loss: 0.2114 - val_a
ccuracy: 0.9585

Epoch 00045: val_accuracy improved from 0.93772 to 0.95848, saving model to best_model.h5
Epoch 46/60
11/11 [==============================] - 48s 4s/step - loss: 0.5209 - accuracy: 0.7825 - val_loss: 0.3290 - val_a
ccuracy: 0.9273

Epoch 00046: val_accuracy did not improve from 0.95848
Epoch 47/60
```

```
11/11 [==============================] - 45s 4s/step - loss: 0.4950 - accuracy: 0.7837 - val_loss: 0.3668 - val_a
ccuracy: 0.8858

Epoch 00047: val_accuracy did not improve from 0.95848
Epoch 48/60
11/11 [==============================] - 45s 4s/step - loss: 0.5102 - accuracy: 0.7820 - val_loss: 0.3142 - val_a
ccuracy: 0.9031

Epoch 00048: val_accuracy did not improve from 0.95848
Epoch 49/60
11/11 [==============================] - 45s 4s/step - loss: 0.4737 - accuracy: 0.8015 - val_loss: 0.3086 - val_a
ccuracy: 0.9204

Epoch 00049: val_accuracy did not improve from 0.95848
Epoch 50/60
11/11 [==============================] - 46s 4s/step - loss: 0.4752 - accuracy: 0.7988 - val_loss: 0.2486 - val_a
ccuracy: 0.9135

Epoch 00050: val_accuracy did not improve from 0.95848
Epoch 51/60
11/11 [==============================] - 46s 4s/step - loss: 0.4718 - accuracy: 0.7965 - val_loss: 0.2184 - val_a
ccuracy: 0.9377

Epoch 00051: val_accuracy did not improve from 0.95848
Epoch 52/60
11/11 [==============================] - 46s 4s/step - loss: 0.4591 - accuracy: 0.8159 - val_loss: 0.2867 - val_a
ccuracy: 0.9100

Epoch 00052: val_accuracy did not improve from 0.95848
Epoch 53/60
11/11 [==============================] - 46s 4s/step - loss: 0.4436 - accuracy: 0.8175 - val_loss: 0.2289 - val_a
ccuracy: 0.9343

Epoch 00053: val_accuracy did not improve from 0.95848
Epoch 54/60
11/11 [==============================] - 46s 4s/step - loss: 0.4212 - accuracy: 0.8236 - val_loss: 0.2998 - val_a
ccuracy: 0.8962

Epoch 00054: val_accuracy did not improve from 0.95848
Epoch 55/60
11/11 [==============================] - 45s 4s/step - loss: 0.4284 - accuracy: 0.8352 - val_loss: 0.2897 - val_a
ccuracy: 0.9031

Epoch 00055: val_accuracy did not improve from 0.95848
Epoch 56/60
11/11 [==============================] - 45s 4s/step - loss: 0.4278 - accuracy: 0.8158 - val_loss: 0.2320 - val_a
ccuracy: 0.9308

Epoch 00056: val_accuracy did not improve from 0.95848
Epoch 57/60
11/11 [==============================] - 45s 4s/step - loss: 0.3913 - accuracy: 0.8391 - val_loss: 0.2346 - val_a
ccuracy: 0.9343

Epoch 00057: val_accuracy did not improve from 0.95848
Epoch 58/60
11/11 [==============================] - 46s 4s/step - loss: 0.3778 - accuracy: 0.8417 - val_loss: 0.4153 - val_a
ccuracy: 0.8616

Epoch 00058: val_accuracy did not improve from 0.95848
Epoch 59/60
11/11 [==============================] - 46s 4s/step - loss: 0.3933 - accuracy: 0.8294 - val_loss: 0.2222 - val_a
ccuracy: 0.9273

Epoch 00059: val_accuracy did not improve from 0.95848
Epoch 60/60
11/11 [==============================] - 45s 4s/step - loss: 0.3806 - accuracy: 0.8598 - val_loss: 0.2872 - val_a
ccuracy: 0.9031

Epoch 00060: val_accuracy did not improve from 0.95848
```

This model unfortunately **does not have a good test accuracy,** and it appears to be **underfitting on the training dataset.** That means we need to increase the complexity of the model in our next attempt.

Convolutional Neural Network (CNN)

Model 3: CNN with Dropout after Convolution and having two Dense layers with 512 & 256 Units respectively

In [ ]:

```
lr = 0.001
kernelsize = 5
```

```python
model_ln4 = LNmodel(in_shape, [8,16], [512,256], kernelsize, 4, lr,
                    stride=1, pool_size=2, leakyrelu_slope=0.1, dropc=0.25,
                    dropd=0.5, bnorm=False)
model_ln4.summary()
```

```
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_7 (InputLayer)         [(None, 150, 150, 1)]     0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 150, 150, 8)       208       
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 150, 150, 8)       0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 150, 150, 8)       0         
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 75, 75, 8)         0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 75, 75, 16)        3216      
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 75, 75, 16)        0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 75, 75, 16)        0         
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 37, 37, 16)        0         
_________________________________________________________________
flatten_6 (Flatten)          (None, 21904)             0         
_________________________________________________________________
dense_15 (Dense)             (None, 512)               11215360  
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 512)               0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 256)               131328    
_________________________________________________________________
leaky_re_lu_24 (LeakyReLU)   (None, 256)               0         
_________________________________________________________________
dropout_15 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 4)                 1028      
=================================================================
Total params: 11,351,140
Trainable params: 11,351,140
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```python
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)

history_model_ln4 = model_ln4.fit(X_train, y_train_e,
                                  validation_split=0.1,
                                  verbose=1, batch_size=512,
                                  shuffle=True, epochs=40,callbacks=[es,mc])
```

```
Epoch 1/40
6/6 [==============================] - 50s 8s/step - loss: 1.0857 - accuracy: 0.5147 - val_loss: 1.5517 - val_acc
uracy: 0.3149

Epoch 00001: val_accuracy improved from -inf to 0.31488, saving model to best_model.h5
Epoch 2/40
6/6 [==============================] - 50s 8s/step - loss: 0.9277 - accuracy: 0.5868 - val_loss: 0.9219 - val_acc
uracy: 0.8097

Epoch 00002: val_accuracy improved from 0.31488 to 0.80969, saving model to best_model.h5
Epoch 3/40
6/6 [==============================] - 50s 8s/step - loss: 0.7879 - accuracy: 0.6539 - val_loss: 0.7665 - val_acc
uracy: 0.8201

Epoch 00003: val_accuracy improved from 0.80969 to 0.82007, saving model to best_model.h5
Epoch 4/40
6/6 [==============================] - 50s 8s/step - loss: 0.7011 - accuracy: 0.6948 - val_loss: 0.7414 - val_acc
uracy: 0.7716

Epoch 00004: val_accuracy did not improve from 0.82007
Epoch 5/40
6/6 [==============================] - 50s 8s/step - loss: 0.6169 - accuracy: 0.7311 - val_loss: 0.7731 - val_acc
uracy: 0.7958

Epoch 00005: val_accuracy did not improve from 0.82007
```

```
Epoch 6/40
6/6 [==============================] - 49s 8s/step - loss: 0.5403 - accuracy: 0.7782 - val_loss: 0.6011 - val_acc
uracy: 0.8443

Epoch 00006: val_accuracy improved from 0.82007 to 0.84429, saving model to best_model.h5
Epoch 7/40
6/6 [==============================] - 49s 8s/step - loss: 0.4738 - accuracy: 0.8067 - val_loss: 0.3903 - val_acc
uracy: 0.9308

Epoch 00007: val_accuracy improved from 0.84429 to 0.93080, saving model to best_model.h5
Epoch 8/40
6/6 [==============================] - 49s 8s/step - loss: 0.4236 - accuracy: 0.8256 - val_loss: 0.5542 - val_acc
uracy: 0.8478

Epoch 00008: val_accuracy did not improve from 0.93080
Epoch 9/40
6/6 [==============================] - 49s 8s/step - loss: 0.3607 - accuracy: 0.8569 - val_loss: 0.3232 - val_acc
uracy: 0.9377

Epoch 00009: val_accuracy improved from 0.93080 to 0.93772, saving model to best_model.h5
Epoch 10/40
6/6 [==============================] - 49s 8s/step - loss: 0.2951 - accuracy: 0.8835 - val_loss: 0.3687 - val_acc
uracy: 0.9066

Epoch 00010: val_accuracy did not improve from 0.93772
Epoch 11/40
6/6 [==============================] - 49s 8s/step - loss: 0.2795 - accuracy: 0.8900 - val_loss: 0.3177 - val_acc
uracy: 0.9135

Epoch 00011: val_accuracy did not improve from 0.93772
Epoch 12/40
6/6 [==============================] - 51s 8s/step - loss: 0.2590 - accuracy: 0.9082 - val_loss: 0.3140 - val_acc
uracy: 0.9135

Epoch 00012: val_accuracy did not improve from 0.93772
Epoch 13/40
6/6 [==============================] - 48s 8s/step - loss: 0.2434 - accuracy: 0.9174 - val_loss: 0.3305 - val_acc
uracy: 0.8997

Epoch 00013: val_accuracy did not improve from 0.93772
Epoch 14/40
6/6 [==============================] - 48s 8s/step - loss: 0.2080 - accuracy: 0.9201 - val_loss: 0.2678 - val_acc
uracy: 0.9204

Epoch 00014: val_accuracy did not improve from 0.93772
Epoch 15/40
6/6 [==============================] - 49s 8s/step - loss: 0.1690 - accuracy: 0.9417 - val_loss: 0.2676 - val_acc
uracy: 0.9239

Epoch 00015: val_accuracy did not improve from 0.93772
Epoch 16/40
6/6 [==============================] - 49s 8s/step - loss: 0.1455 - accuracy: 0.9471 - val_loss: 0.2155 - val_acc
uracy: 0.9343

Epoch 00016: val_accuracy did not improve from 0.93772
Epoch 17/40
6/6 [==============================] - 49s 8s/step - loss: 0.1396 - accuracy: 0.9537 - val_loss: 0.4320 - val_acc
uracy: 0.8754

Epoch 00017: val_accuracy did not improve from 0.93772
Epoch 18/40
6/6 [==============================] - 49s 8s/step - loss: 0.1238 - accuracy: 0.9591 - val_loss: 0.1803 - val_acc
uracy: 0.9481

Epoch 00018: val_accuracy improved from 0.93772 to 0.94810, saving model to best_model.h5
Epoch 19/40
6/6 [==============================] - 49s 8s/step - loss: 0.0963 - accuracy: 0.9684 - val_loss: 0.2995 - val_acc
uracy: 0.9170

Epoch 00019: val_accuracy did not improve from 0.94810
Epoch 20/40
6/6 [==============================] - 49s 8s/step - loss: 0.0768 - accuracy: 0.9761 - val_loss: 0.2468 - val_acc
uracy: 0.9343

Epoch 00020: val_accuracy did not improve from 0.94810
Epoch 21/40
6/6 [==============================] - 49s 8s/step - loss: 0.0711 - accuracy: 0.9730 - val_loss: 0.1880 - val_acc
uracy: 0.9550

Epoch 00021: val_accuracy improved from 0.94810 to 0.95502, saving model to best_model.h5
Epoch 22/40
6/6 [==============================] - 49s 8s/step - loss: 0.1018 - accuracy: 0.9645 - val_loss: 0.2639 - val_acc
uracy: 0.9273
```

```
Epoch 00022: val_accuracy did not improve from 0.95502
Epoch 23/40
6/6 [==============================] - 49s 8s/step - loss: 0.0923 - accuracy: 0.9676 - val_loss: 0.1783 - val_acc
uracy: 0.9481

Epoch 00023: val_accuracy did not improve from 0.95502
Epoch 24/40
6/6 [==============================] - 51s 8s/step - loss: 0.0601 - accuracy: 0.9846 - val_loss: 0.1802 - val_acc
uracy: 0.9585

Epoch 00024: val_accuracy improved from 0.95502 to 0.95848, saving model to best_model.h5
Epoch 25/40
6/6 [==============================] - 50s 8s/step - loss: 0.0571 - accuracy: 0.9842 - val_loss: 0.2453 - val_acc
uracy: 0.9308

Epoch 00025: val_accuracy did not improve from 0.95848
Epoch 26/40
6/6 [==============================] - 49s 8s/step - loss: 0.0574 - accuracy: 0.9826 - val_loss: 0.1726 - val_acc
uracy: 0.9550

Epoch 00026: val_accuracy did not improve from 0.95848
Epoch 27/40
6/6 [==============================] - 49s 8s/step - loss: 0.0406 - accuracy: 0.9861 - val_loss: 0.1922 - val_acc
uracy: 0.9481

Epoch 00027: val_accuracy did not improve from 0.95848
Epoch 28/40
6/6 [==============================] - 49s 8s/step - loss: 0.0357 - accuracy: 0.9888 - val_loss: 0.2127 - val_acc
uracy: 0.9481

Epoch 00028: val_accuracy did not improve from 0.95848
Epoch 29/40
6/6 [==============================] - 49s 8s/step - loss: 0.0271 - accuracy: 0.9931 - val_loss: 0.2286 - val_acc
uracy: 0.9446

Epoch 00029: val_accuracy did not improve from 0.95848
Epoch 30/40
6/6 [==============================] - 49s 8s/step - loss: 0.0265 - accuracy: 0.9907 - val_loss: 0.2496 - val_acc
uracy: 0.9239

Epoch 00030: val_accuracy did not improve from 0.95848
Epoch 31/40
6/6 [==============================] - 49s 8s/step - loss: 0.0322 - accuracy: 0.9877 - val_loss: 0.1698 - val_acc
uracy: 0.9550

Epoch 00031: val_accuracy did not improve from 0.95848
Epoch 32/40
6/6 [==============================] - 49s 8s/step - loss: 0.0342 - accuracy: 0.9888 - val_loss: 0.2032 - val_acc
uracy: 0.9446

Epoch 00032: val_accuracy did not improve from 0.95848
Epoch 33/40
6/6 [==============================] - 49s 8s/step - loss: 0.0425 - accuracy: 0.9880 - val_loss: 0.2496 - val_acc
uracy: 0.9308

Epoch 00033: val_accuracy did not improve from 0.95848
Epoch 34/40
6/6 [==============================] - 49s 8s/step - loss: 0.0263 - accuracy: 0.9938 - val_loss: 0.2398 - val_acc
uracy: 0.9412

Epoch 00034: val_accuracy did not improve from 0.95848
Epoch 35/40
6/6 [==============================] - 49s 8s/step - loss: 0.0243 - accuracy: 0.9954 - val_loss: 0.2062 - val_acc
uracy: 0.9516

Epoch 00035: val_accuracy did not improve from 0.95848
Epoch 36/40
6/6 [==============================] - 49s 8s/step - loss: 0.0231 - accuracy: 0.9927 - val_loss: 0.2284 - val_acc
uracy: 0.9412

Epoch 00036: val_accuracy did not improve from 0.95848
Epoch 37/40
6/6 [==============================] - 50s 8s/step - loss: 0.0203 - accuracy: 0.9950 - val_loss: 0.2200 - val_acc
uracy: 0.9516

Epoch 00037: val_accuracy did not improve from 0.95848
Epoch 38/40
6/6 [==============================] - 49s 8s/step - loss: 0.0167 - accuracy: 0.9958 - val_loss: 0.1870 - val_acc
uracy: 0.9585

Epoch 00038: val_accuracy did not improve from 0.95848
Epoch 39/40
```

```
6/6 [==============================] - 49s 8s/step - loss: 0.0157 - accuracy: 0.9969 - val_loss: 0.1920 - val_acc
uracy: 0.9585

Epoch 00039: val_accuracy did not improve from 0.95848
Epoch 40/40
6/6 [==============================] - 49s 8s/step - loss: 0.0115 - accuracy: 0.9965 - val_loss: 0.1988 - val_acc
uracy: 0.9550

Epoch 00040: val_accuracy did not improve from 0.95848
```

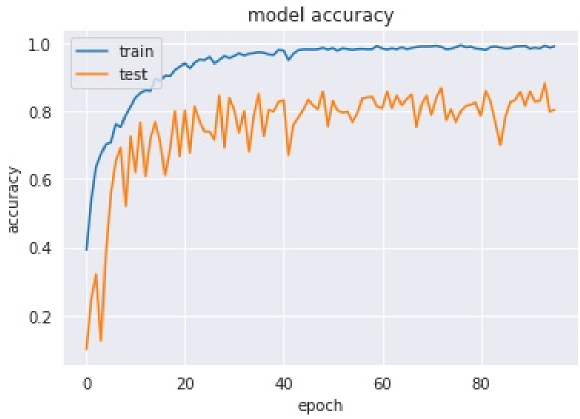
Plotting the Train & Test Accuracy

CNN Model 1

In [ ]:

```python
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

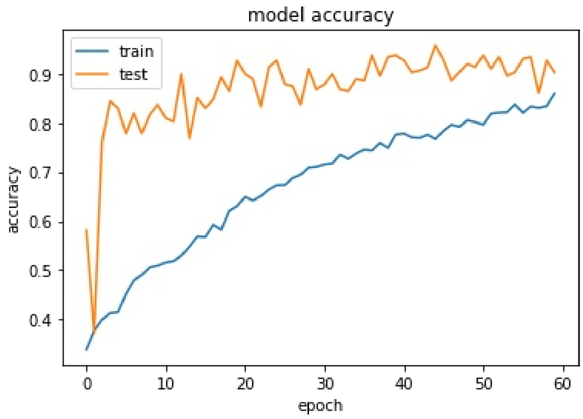


CNN Model 2

In [ ]:

```python
print(history_model_ln3.history.keys())
# summarize history for accuracy
plt.plot(history_model_ln3.history['accuracy'])
plt.plot(history_model_ln3.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

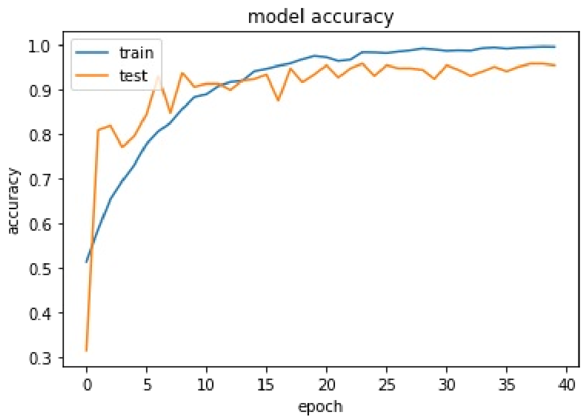


CNN Model 3

In [ ]:

```python
print(history_model_ln4.history.keys())
# summarize history for accuracy
plt.plot(history_model_ln4.history['accuracy'])
plt.plot(history_model_ln4.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



Model Evaluation

CNN Model 1

In [ ]:

```python
model.evaluate(X_test,y_test_e)
```

```
13/13 [==============================] - 10s 747ms/step - loss: 4.8989 - accuracy: 0.6866
```

Out[ ]:

```
[4.8988800048828125, 0.6865671873092651]
```

CNN Model 2

In [ ]:

```python
model_ln3.evaluate(X_test,y_test_e)
```

```
13/13 [==============================] - 2s 143ms/step - loss: 1.5445 - accuracy: 0.7463
```

Out[ ]:

```
[1.544519066810608, 0.746268630027771]
```

CNN Model 3

In [ ]:

```python
model_ln4.evaluate(X_test,y_test_e)
```

```
13/13 [==============================] - 2s 156ms/step - loss: 2.3007 - accuracy: 0.7637
```

Out[ ]:

```
[2.3007419109344482, 0.7636815905570984]
```

Unfortunately, **we cannot decide the best model based on test accuracy here** because we are dealing with an imbalanced dataset, so we are more concerned with **Precision and Recall**. Since these two metrics are both quite important in this scenario, we will also check the **F1 score** to try to achieve a good balance between Precision and Recall.

Plotting the confusion matrix for the two best models

As we can see, **Model 2 and Model 3 seem to be generalizing well** because they both have a good Holdout set Accuracy. **Let us compute**

As we can see, Model 2 and Model 3 seem to be generalizing well because they both have a good Holdout set Accuracy. Let us compute **the confusion matrix** for these two models to understand the distribution of True Positives, False Positives, False Negatives and True Negatives.

CNN Model 2

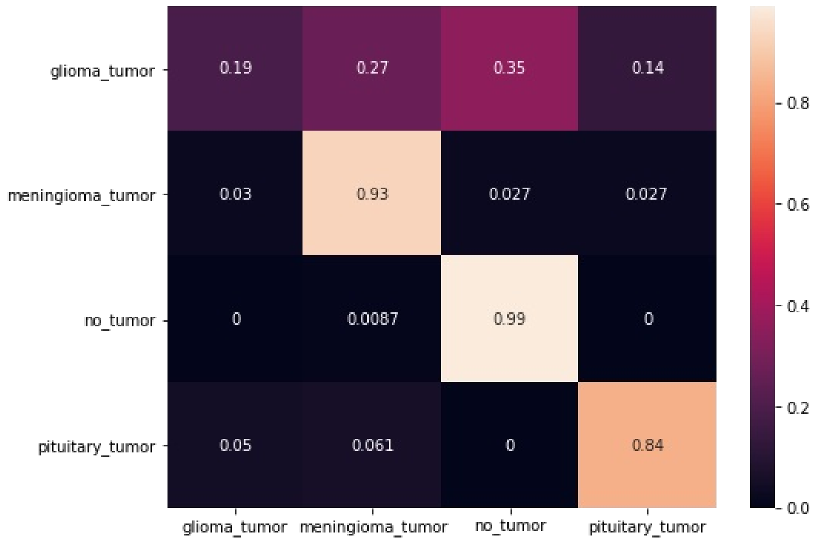
In [ ]:
```
# Test Prediction
y_test_pred_ln3 = model_ln3.predict(X_test)
y_test_pred_classes_ln3 = np.argmax(y_test_pred_ln3, axis=1)
y_test_pred_prob_ln3 = np.max(y_test_pred_ln3, axis=1)
```

In [ ]:
```
# Test Accuracy
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix
accuracy_score(np.array(y_test), y_test_pred_classes_ln3)
```

Out[ ]:
```
0.746268656716418
```

In [ ]:
```
cf_matrix = confusion_matrix(np.array(y_test), y_test_pred_classes_ln3)

# Confusion matrix normalized per category true value
cf_matrix_n1 = cf_matrix/np.sum(cf_matrix, axis=1)
plt.figure(figsize=(8,6))
sns.heatmap(cf_matrix_n1, xticklabels=CATEGORIES, yticklabels=CATEGORIES, annot=True)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb28293e630>
```



CNN Model 3

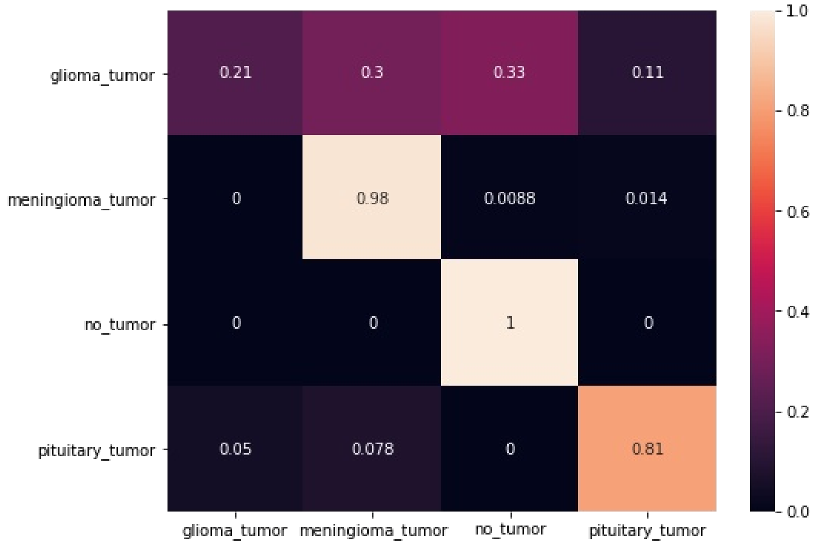
In [ ]:
```
#Test Prediction

y_test_pred_ln4 = model_ln4.predict(X_test)
y_test_pred_classes_ln4 = np.argmax(y_test_pred_ln4, axis=1)
y_test_pred_prob_ln4 = np.max(y_test_pred_ln4, axis=1)
```

In [ ]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix
accuracy_score(np.array(y_test), y_test_pred_classes_ln4)
```

Out[ ]:
```
0.763681592039801
```

In [ ]:
```
cf_matrix = confusion_matrix(np.array(y_test), y_test_pred_classes_ln4)

# Confusion matrix normalized per category true value
cf_matrix_n1 = cf_matrix/np.sum(cf_matrix, axis=1)
plt.figure(figsize=(8,6))
sns.heatmap(cf_matrix_n1, xticklabels=CATEGORIES, yticklabels=CATEGORIES, annot=True)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb27e918588>
```



The above two confusion matrices show that the models seem to be working well. **Let's calculate the F1 score** (the harmonic mean of precision and recall), which is used as an evaluation metric for imbalanced datasets.

Classification Report for each class

- **Precision:** precision is the fraction of relevant instances among the retrieved instances.
- **Recall:** recall is the fraction of relevant instances that were retrieved.
- **F-beta score:** The F-beta score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. The beta parameter determines the weight of recall in the combined score.

The order of printing the above metrices for each class is as follows:

- **Glioma Tumor**
- **Meningioma Tumor**
- **Non Tumor**
- **Pituitary Tumor**

CNN Model 2

```
In [ ]:
from sklearn.metrics import precision_recall_fscore_support

p=precision_recall_fscore_support(np.array(y_test), y_test_pred_classes_ln3, average=None,labels=list(np.unique(y

print(" Precision is {}\n Recall is {} \n f_beta Score is {}".format(p[0],p[1],p[2]))
```

```
Precision is [0.7037037  0.73287671 0.72258065 0.83783784]
Recall is [0.19       0.93043478 0.99115044 0.83783784]
f_beta Score is [0.2992126  0.81992337 0.8358209  0.83783784]
```

CNN Model 3

```
In [ ]:
from sklearn.metrics import precision_recall_fscore_support

p=precision_recall_fscore_support(np.array(y_test), y_test_pred_classes_ln4, average=None,labels=list(np.unique(y


print(" Precision is {}\n Recall is {} \n f_beta Score is {}".format(p[0],p[1],p[2]))
```

```
Precision is [0.80769231 0.72435897 0.74834437 0.86956522]
Recall is [0.21       0.9826087  1.         0.81081081]
f_beta Score is [0.33333333 0.83394834 0.85606061 0.83916084]
```

Model 3 (Best) Observation

As we see from the precision for each class, the Pituitary tumor classifier has the highest precision. But here, **we are more concerned about**

the case where a person who has a tumor is wrongly classified as belonging to the non-tumor category (False Negative).

33% of the persons belonging to Glioma tumour and 8.6% belonging to Meningioma tumor are not identified correctly, and the model predicts that they don't have a tumor at all - which shows that our model does not do well in identifying glioma and meningioma tumors. But it is works well for the other scenario, where the model is able to correctly identify those scans that do not not show a tumor.

Weighted F-Score

Model 2

In [ ]:
```
from sklearn.metrics import f1_score

f1_score(np.array(y_test), y_test_pred_classes_ln3, average='weighted')
```

Out[ ]:
```
0.6981597233234159
```

Model 3

In [ ]:
```
from sklearn.metrics import f1_score

f1_score(np.array(y_test), y_test_pred_classes_ln4, average='weighted')
```

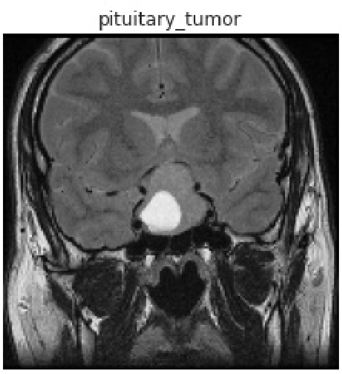
Out[ ]:
```
0.7165923954146127
```

Model 3 with 2 Dense layer and more number of units having better F1 score.

Prediction

Let us predict with best model with is model_ln4

In [ ]:
```
#fn = image.load_img(fp, target_size = (150,150), color_mode='grayscale')
plt.imshow(X_test[1].reshape(150,150), cmap='Greys_r')
i=y_test[1]
i=np.argmax(i)
if(i == 0):
    plt.title("glioma_tumor")
if(i==1):
    plt.title("meningioma_tumor")
if(i==2):
    plt.title("no_tumor")
if(i==3):
    plt.title("pituitary_tumor")

plt.axis('off')
plt.show()
```



In [ ]:
```
res=model_ln4.predict(X_test[1].reshape(1,150,150,1))
```

In [ ]:
```
i=np.argmax(res)


if(i == 0):
```

```
    print("glioma_tumor")
if(i==1):
    print("meningioma_tumor")
if(i==2):
    print("no_tumor")
if(i==3):
    print("pituitary_tumor")
```

```
pituitary_tumor
```

Conclusion

Conclusions

Brain Tumor Classification Using Deep Learning Algorithms

Importance of Automation in MRI Analysis:

Brain tumors, being highly aggressive, require precise diagnostics. Manual MRI image analysis by neurosurgeons is time-consuming and prone to errors. An automated system using deep learning can significantly improve the accuracy and speed of brain tumor classification, aiding in early and effective treatment. Dataset and Preprocessing:

The dataset consists of MRI images categorized into four types: glioma tumor, meningioma tumor, pituitary tumor, and no tumor. The images were preprocessed by converting them to grayscale and resizing them to standard dimensions, reducing complexity and computational load. Imbalanced Dataset:

The dataset is imbalanced, with a higher proportion of tumor images compared to non-tumor images. This imbalance can impact the model's performance, necessitating careful evaluation metrics beyond just accuracy. Model Performance:

Three models were built and evaluated: ANN (Artificial Neural Network): The ANN model showed poor performance with a test accuracy of 70.65%, as it failed to capture the spatial correlations in the images. CNN Model 1: This model included dropout layers but showed a test accuracy of 68.66%, indicating overfitting on the training data. CNN Model 2: This model, with different convolution and dense layers, achieved a better test accuracy of 74.63% and demonstrated good generalization. CNN Model 3: The best performing model with more complex architecture, achieved a test accuracy of 76.37% and a higher F1 score, indicating better balance between precision and recall. Evaluation Metrics:

The confusion matrices and classification reports for CNN Models 2 and 3 revealed that these models performed well in identifying most tumor types but struggled with glioma and meningioma tumors. The weighted F1 score was higher for CNN Model 3, making it the best model among the three. Future Improvements:

Hyperparameter Tuning: Further tuning of the hyperparameters and exploring different architectures could enhance the model's performance. Addressing Imbalance: Techniques such as oversampling the minority class or using class-weighted loss functions can help mitigate the impact of the imbalanced dataset. Filter Visualization: Visualizing the convolutional filters can provide insights into why the model struggles with certain tumor types, guiding further refinements. Overall Conclusion:

CNNs outperform ANNs for image data due to their ability to maintain the spatial structure of images. While the current models show promising results, there is potential for further improvement to achieve more reliable and accurate brain tumor classification. By leveraging advanced deep learning techniques, we can enhance the diagnostic process for brain tumors, ultimately contributing to better patient outcomes through timely and precise treatment interventions.

In [ ]:

```
```