In [1]:

```
import cv2
import pandas
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from sklearn.model_selection import RandomizedSearchCV, KFold, train_test_split
```

In [2]:

```
import os
from PIL import Image
from tensorflow.keras.preprocessing.image import img_to_array
```

In [3]:

```
tumor = [('glioma_tumor', 0),('meningioma_tumor', 1),('no_tumor', 2),('pituitary_tumor',
path1 = "C://Users//M Abhishek Reddy//Desktop//BTD//Training//"
path2 = "C://Users//M Abhishek Reddy//Desktop//BTD//Testing//"
```

In [4]:

```
Path = ['C://Users//M Abhishek Reddy//Desktop//BTD//Training//glioma_tumor//gg (104).jpg
        'C://Users//M Abhishek Reddy//Desktop//BTD//Training//meningioma_tumor\m (10).jp
       'C://Users//M Abhishek Reddy//Desktop//BTD//Training//no_tumor//image (12).jpg',
        'C://Users//M Abhishek Reddy//Desktop//BTD//Training//pituitary_tumor//p (122).j
```

In [5]:

```
def display (img_array) :

    dim = 10

    plt.figure(figsize  = (dim , dim))
    for i, img in enumerate(img_array) :
        plt.subplot(2, 2, i+1)
        plt.imshow(img,'gray')
        plt.title(img.shape)

    plt.show()
```

In [6]:

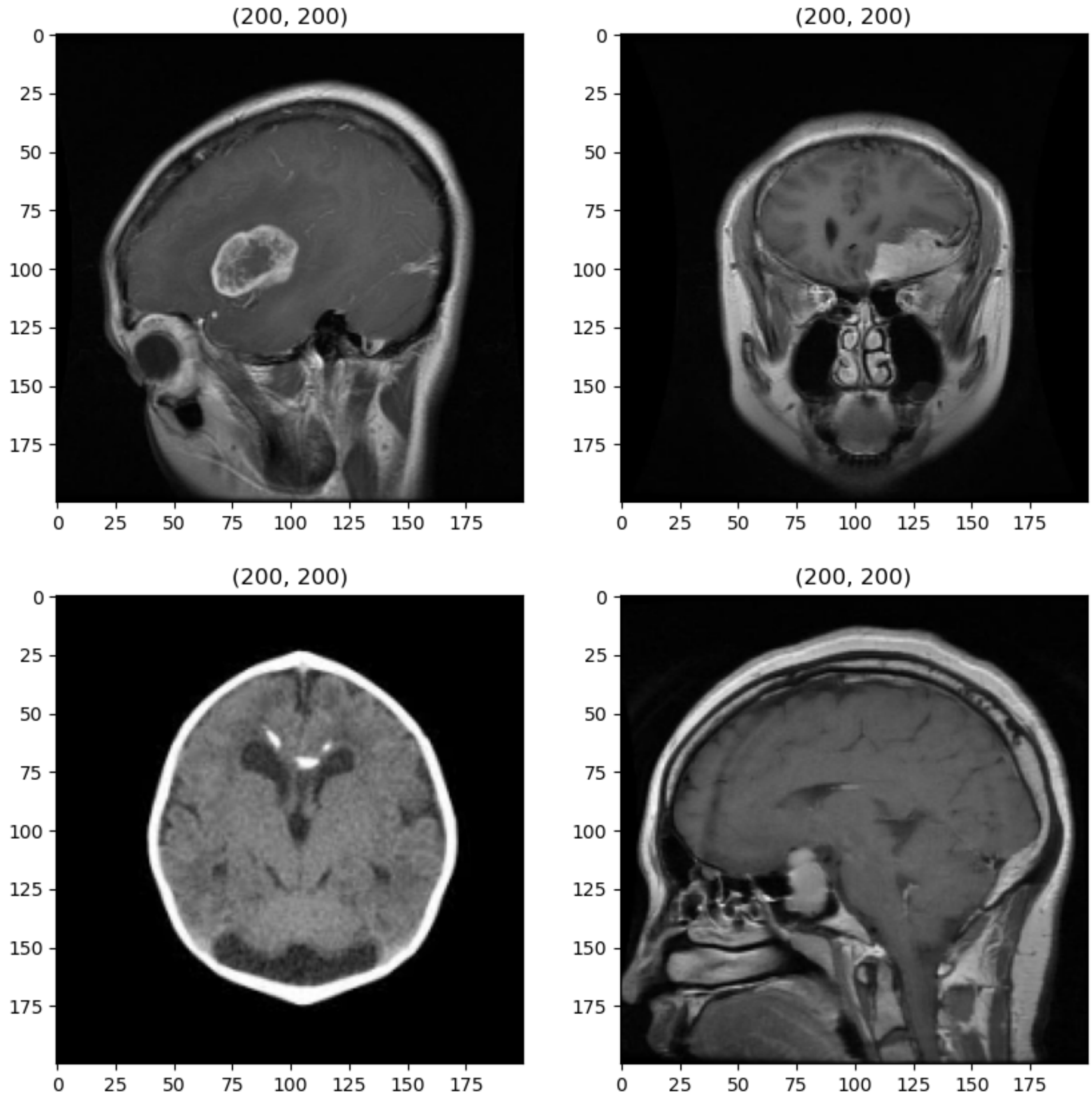
```
def get_array (path) :

    X = []
    y = []

    for typ, val in tumor :
        for image in os.listdir(path+'//'+typ) :
            img = cv2.resize(cv2.imread(path+'//'+typ+'/'+image, cv2.IMREAD_GRAYSCALE),

            # create arrays
            X.append(img_to_array( Image.fromarray(img)))
            y.append(val)
    return X, y
```

In [7]:

```
image_array = []
for i, image_path in enumerate(Path) :
    image_array.append(cv2.resize(cv2.imread(image_path, cv2.IMREAD_GRAYSCALE), (200,200
```

In [8]:

```
display(image_array)
```



In [9]:

```
X_train, y_train = get_array(path1)
```

In [10]:

```
X_test , y_test  = get_array(path2)
```

In [11]:

```
X_train, X_test = np.array(X_train), np.array(X_test)
y_train, y_test = np.array(y_train), np.array(y_test)
```

In [12]:

```
X_train = np.array(X_train)
X_train/= 255.0
```

In [13]:

```
y_train = np.array(keras.utils.to_categorical(y_train))
```

In [14]:

```
X_test = np.array(X_test)
X_test/= 255.0
```

In [15]:

```
y_test = np.array(keras.utils.to_categorical(y_test))
```

In [16]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(2870, 200, 200, 1)
(2870, 4)
```

In [17]:

```
print(X_test.shape)
print(y_test.shape)
```

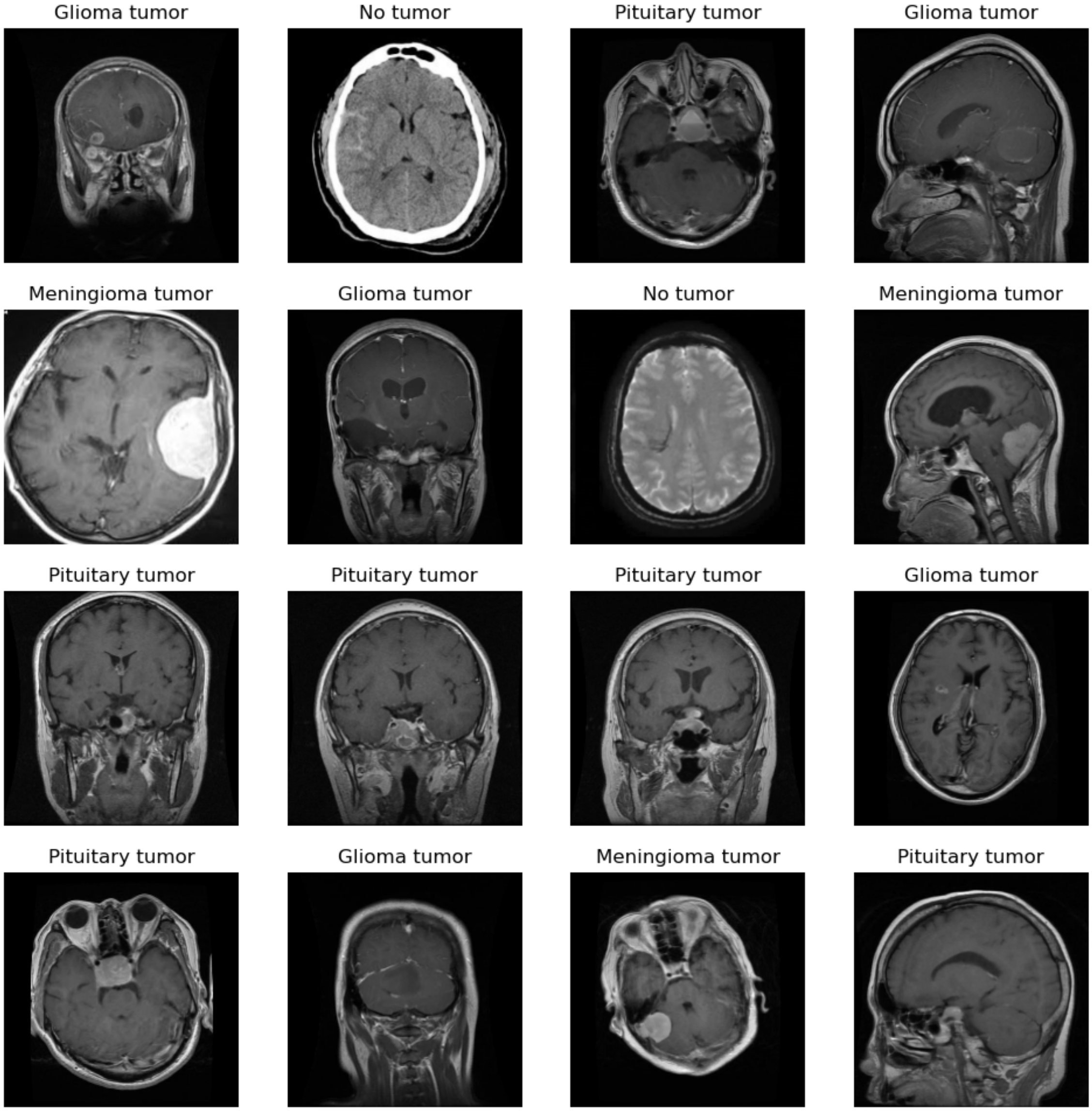
```
(394, 200, 200, 1)
(394, 4)
```

In [18]:

```
info = {0 : 'Glioma tumor', 1 : 'Meningioma tumor', 2 : 'No tumor', 3 : 'Pituitary tumor
```

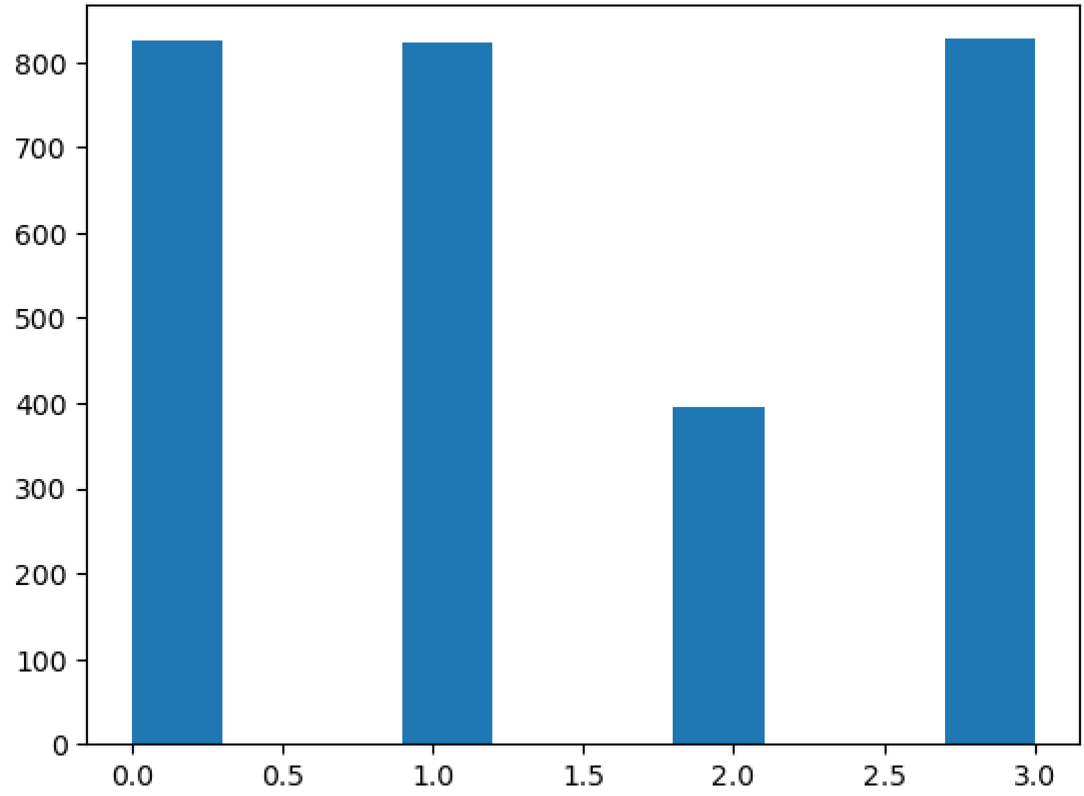
In [19]:

```
# training sample's plots
plt.figure(figsize = (12,12))
for i in range(16) :
    plt.subplot(4, 4, i+1)
    x = np.random.randint(0, 2870)
    plt.imshow(X_train[x], 'gray')
    plt.title(info[np.argmax(y_train[x])])
    plt.axis('off')
plt.show()
```



In [20]:

```
plt.hist(np.argmax(y_train, axis = 1))
plt.show()
```



In [21]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(2870, 200, 200, 1)
(2870, 4)
```

In [22]:

```
X_train = np.reshape(X_train, (2870, 200*200*1))
print(X_train.shape)
print(y_train.shape)
```

```
(2870, 40000)
(2870, 4)
```

In [23]:

```
from imblearn.over_sampling import SMOTE
X_train, y_train = SMOTE(sampling_strategy = 'auto', random_state = 1, k_neighbors = 5).
```

In [24]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(3308, 40000)
(3308, 4)
```

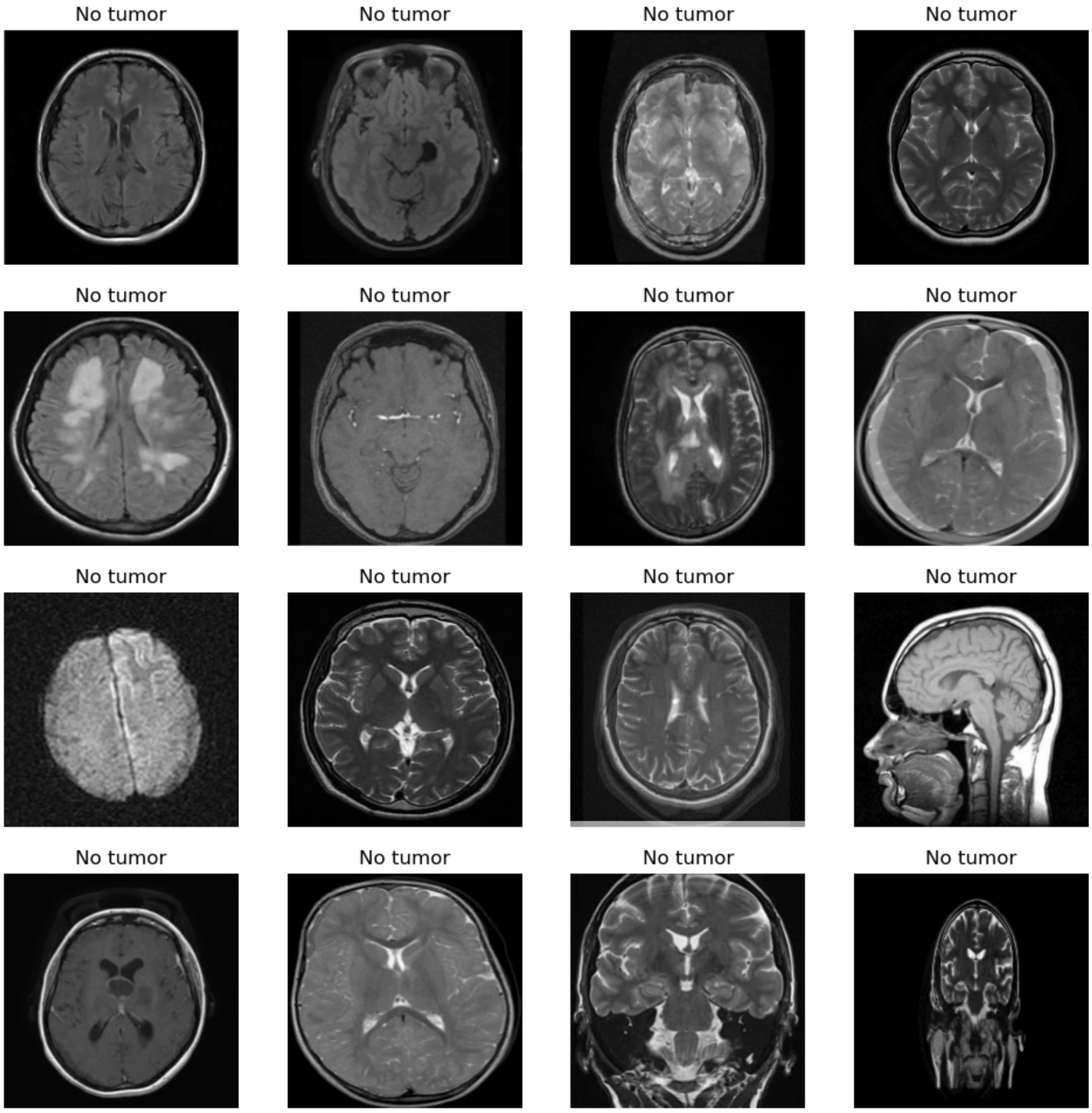
In [25]:

```
X_train = np.reshape(X_train, (3308, 200, 200, 1))
print(X_train.shape)
print(y_train.shape)
```

```
(3308, 200, 200, 1)
(3308, 4)
```

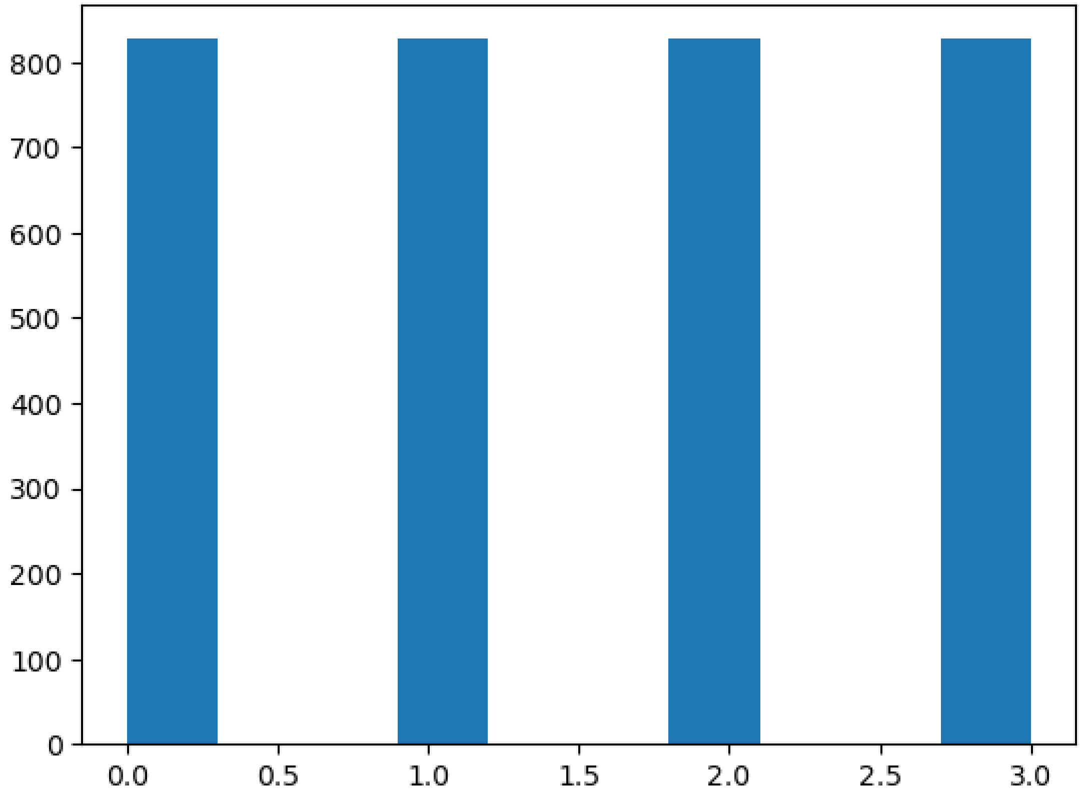
In [26]:

```
plt.figure(figsize = (12,12))
for i in range(16) :
    plt.subplot(4, 4, i+1)
    x = np.random.randint(2870, 3308)
    plt.imshow(X_train[x], 'gray')
    plt.title(info[np.argmax(y_train[x])])
    plt.axis('off')
plt.show()
```



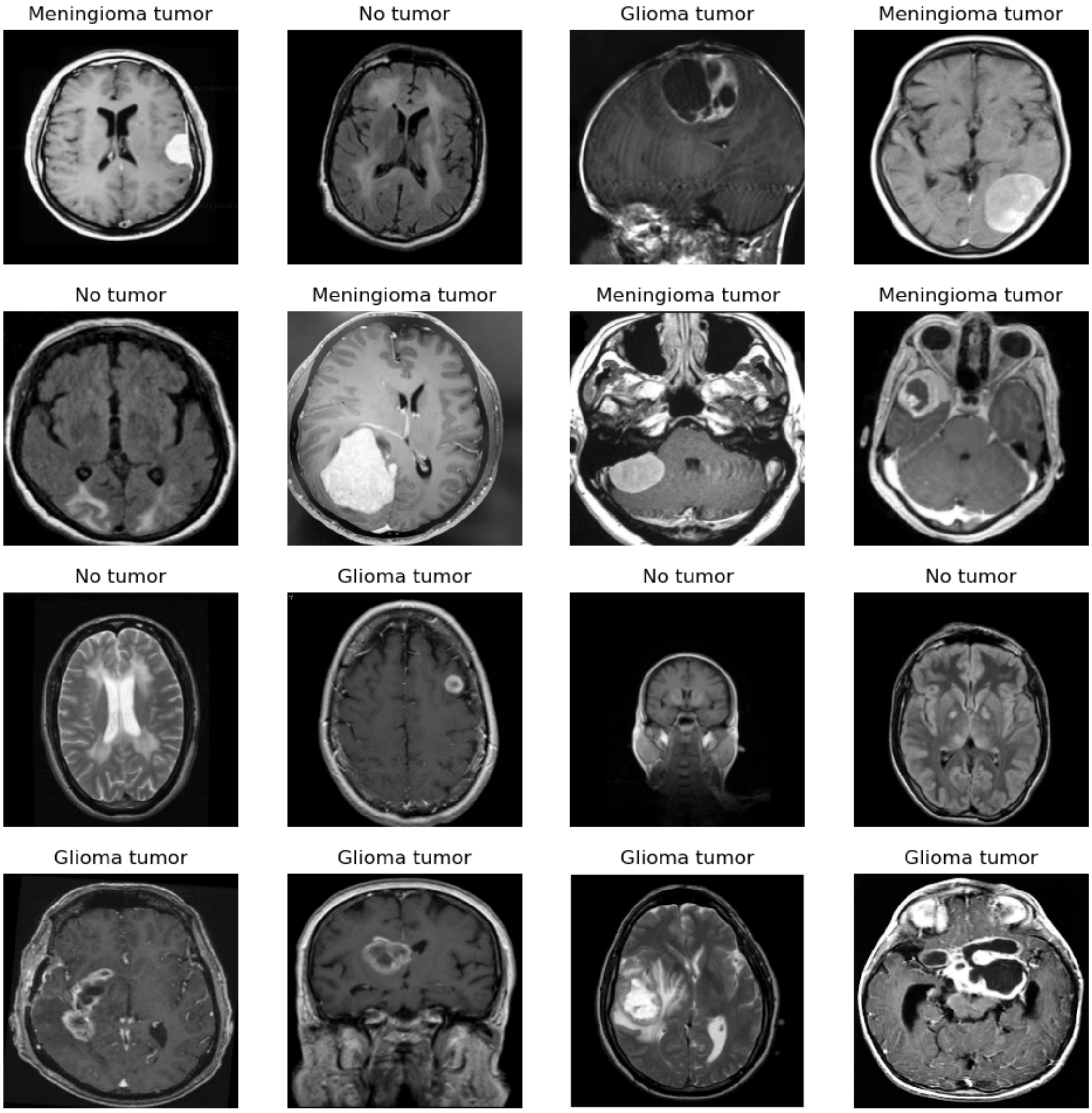
In [27]:

```
plt.hist(np.argmax(y_train, axis = 1))
plt.show()
```



In [28]:

```
# test sample images
plt.figure(figsize = (12,12))
for i in range(16) :
    plt.subplot(4, 4, i+1)
    x = np.random.randint(0,390)
    plt.imshow(X_test[x],'gray')
    plt.title(info[np.argmax(y_test[x])])
    plt.axis('off')
plt.show()
```



In [29]:

```
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.models import Sequential
from keras.layers import BatchNormalization
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Conv2D
```

In [30]:

```
def conv_layer (filterx) :

    model = Sequential()   # working on single layer
    model.add(Conv2D(filterx, (3,3), padding = 'same', activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2), padding = 'valid'))
    model.add(BatchNormalization())

    return model
```

In [31]:

```
def dens_layer (hiddenx) :

    model = Sequential()
    model.add(Dense(hiddenx, activation = 'relu', kernel_regularizer = 'l2'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    return model
```

In [32]:

```
def cnn (filter1, filter2, filter3, hidden1, hidden2) :

    model = Sequential()

    model.add(Input((200,200,1,)))
    model.add(conv_layer(filter1))
    model.add(conv_layer(filter2))
    model.add(conv_layer(filter3))

    model.add(Flatten())
    model.add(dens_layer(hidden1))
    model.add(dens_layer(hidden2))
    model.add(Dense(4, activation = 'softmax'))

    model.compile(loss = 'categorical_crossentropy', optimizer = keras.optimizers.Adam(l

    return model
```

In [33]:

```
print(X_train.shape)
print(y_train.shape)
```

```
(3308, 200, 200, 1)
(3308, 4)
```

In [34]:

```
from keras.preprocessing.image import ImageDataGenerator
gen = ImageDataGenerator(zoom_range = [0.85, 1.0], rotation_range = 3)
# Generate batches of tensor image data with real-time data augmentation.
```

In [35]:

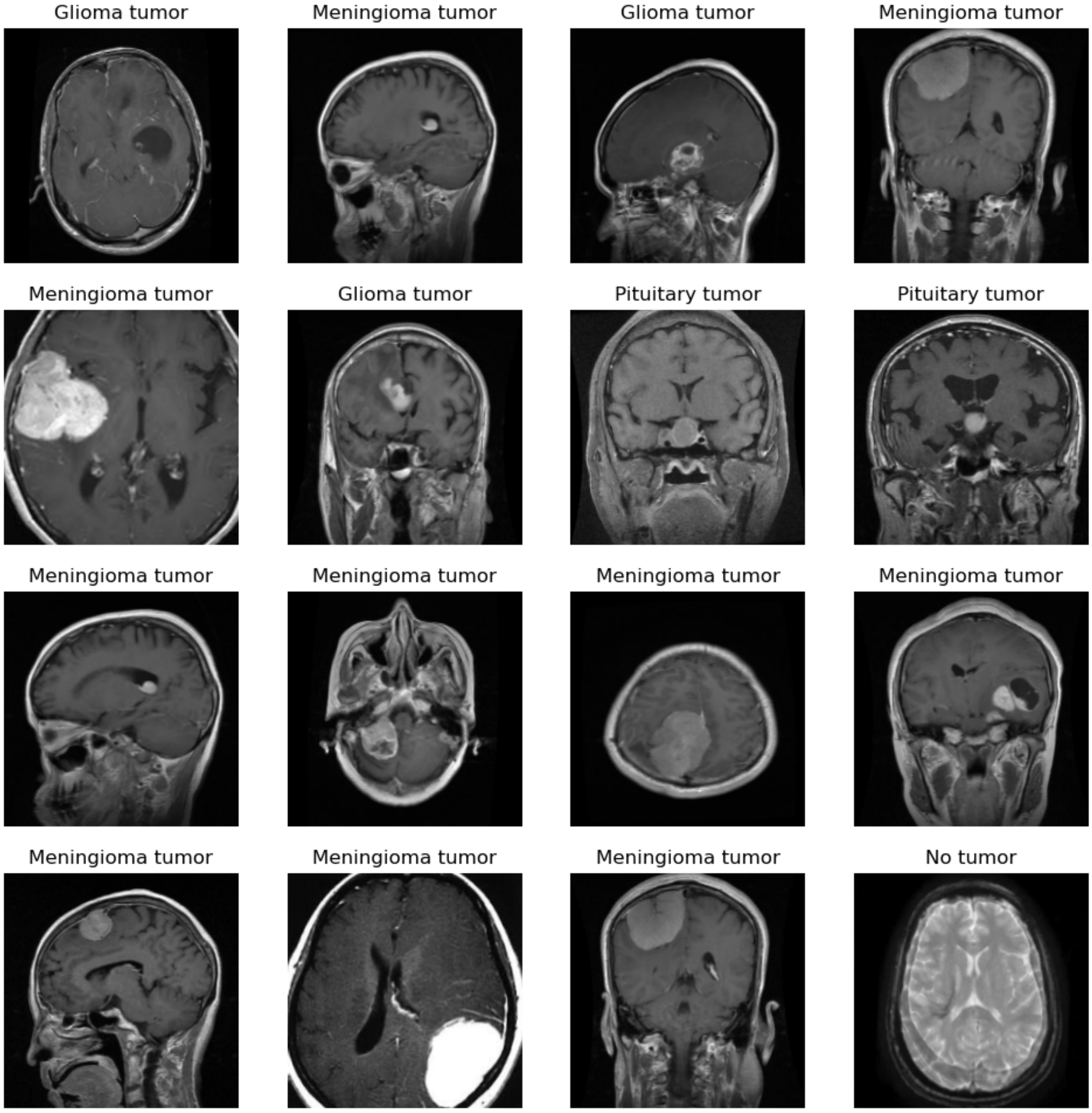
```
gen.fit(X_train)
train_gen = gen.flow(X_train, y_train, batch_size = 32)
```

In [36]:

```
trainX, trainy = train_gen.next()
```

In [37]:

```
plt.figure(figsize = (12,12))
for i in range(16) :
    plt.subplot(4, 4, i+1)
    plt.imshow(trainX[i], 'gray')
    plt.title(info[np.argmax(trainy[i])])
    plt.axis('off')
plt.show()
```



In [38]:

```
from keras.callbacks import ModelCheckpoint
checkp = ModelCheckpoint('./brain_model.h5', monitor = 'val_accuracy', save_best_only =
```

In [39]:

```
model = cnn(128, 64, 32, 128, 64)
```

In [40]:

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential_1 (Sequential)   (None, 100, 100, 128)     1792      
                                                                 
 sequential_2 (Sequential)   (None, 50, 50, 64)        74048     
                                                                 
 sequential_3 (Sequential)   (None, 25, 25, 32)        18592     
                                                                 
 flatten (Flatten)           (None, 20000)             0         
                                                                 
 sequential_4 (Sequential)   (None, 128)               2560640   
                                                                 
 sequential_5 (Sequential)   (None, 64)                8512      
                                                                 
 dense_2 (Dense)             (None, 4)                 260       
                                                                 
=================================================================
Total params: 2,663,844
Trainable params: 2,663,012
Non-trainable params: 832
_________________________________________________________________
None
```

In [41]:

```
history = model.fit(gen.flow(X_train, y_train, batch_size = 32), epochs = 10, validation
```

```
Epoch 1/10
104/104 [==============================] - ETA: 0s - loss: 4.2878 - accura
cy: 0.6569
Epoch 1: val_accuracy improved from -inf to 0.26650, saving model to .\bra
in_model.h5
104/104 [==============================] - 371s 4s/step - loss: 4.2878 - a
ccuracy: 0.6569 - val_loss: 5.7071 - val_accuracy: 0.2665
Epoch 2/10
104/104 [==============================] - ETA: 0s - loss: 3.8657 - accura
cy: 0.7651
Epoch 2: val_accuracy did not improve from 0.26650
104/104 [==============================] - 361s 3s/step - loss: 3.8657 - a
ccuracy: 0.7651 - val_loss: 7.0795 - val_accuracy: 0.2665
Epoch 3/10
104/104 [==============================] - ETA: 0s - loss: 3.5989 - accura
cy: 0.8232
Epoch 3: val_accuracy did not improve from 0.26650
104/104 [==============================] - 408s 4s/step - loss: 3.5989 - a
ccuracy: 0.8232 - val_loss: 7.3545 - val_accuracy: 0.2665
Epoch 4/10
104/104 [==============================] - ETA: 0s - loss: 3.4110 - accura
cy: 0.8449
Epoch 4: val_accuracy improved from 0.26650 to 0.27157, saving model to
.\brain_model.h5
104/104 [==============================] - 388s 4s/step - loss: 3.4110 - a
ccuracy: 0.8449 - val_loss: 6.8136 - val_accuracy: 0.2716
Epoch 5/10
104/104 [==============================] - ETA: 0s - loss: 3.2239 - accura
cy: 0.8664
Epoch 5: val_accuracy improved from 0.27157 to 0.34264, saving model to
.\brain_model.h5
104/104 [==============================] - 413s 4s/step - loss: 3.2239 - a
ccuracy: 0.8664 - val_loss: 5.3930 - val_accuracy: 0.3426
Epoch 6/10
104/104 [==============================] - ETA: 0s - loss: 3.0051 - accura
cy: 0.8993
Epoch 6: val_accuracy improved from 0.34264 to 0.55076, saving model to
.\brain_model.h5
104/104 [==============================] - 405s 4s/step - loss: 3.0051 - a
ccuracy: 0.8993 - val_loss: 4.0670 - val_accuracy: 0.5508
Epoch 7/10
104/104 [==============================] - ETA: 0s - loss: 2.8463 - accura
cy: 0.9087
Epoch 7: val_accuracy did not improve from 0.55076
104/104 [==============================] - 374s 4s/step - loss: 2.8463 - a
ccuracy: 0.9087 - val_loss: 4.0414 - val_accuracy: 0.5406
Epoch 8/10
104/104 [==============================] - ETA: 0s - loss: 2.6608 - accura
cy: 0.9238
Epoch 8: val_accuracy improved from 0.55076 to 0.62437, saving model to
.\brain_model.h5
104/104 [==============================] - 383s 4s/step - loss: 2.6608 - a
ccuracy: 0.9238 - val_loss: 3.8594 - val_accuracy: 0.6244
Epoch 9/10
104/104 [==============================] - ETA: 0s - loss: 2.5262 - accura
cy: 0.9265
Epoch 9: val_accuracy improved from 0.62437 to 0.63706, saving model to
.\brain_model.h5
104/104 [==============================] - 369s 4s/step - loss: 2.5262 - a
ccuracy: 0.9265 - val_loss: 3.6299 - val_accuracy: 0.6371
Epoch 10/10
```

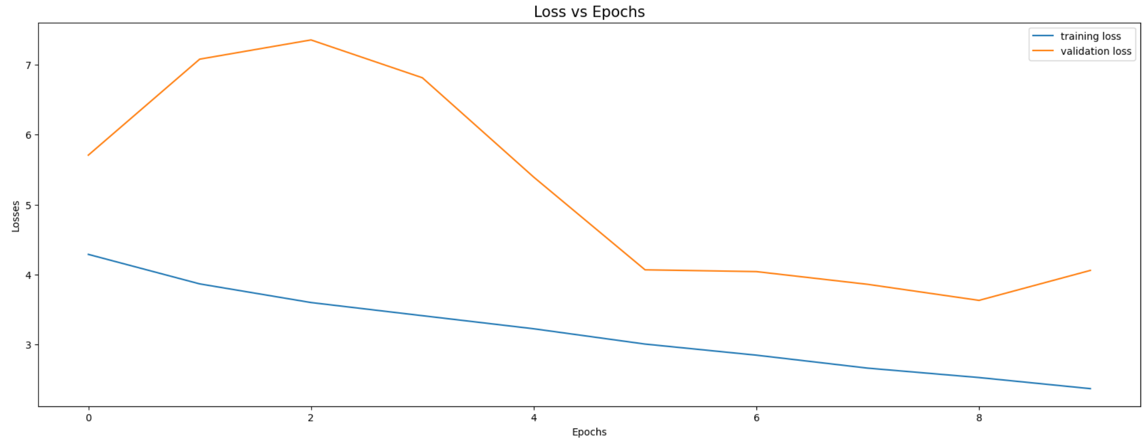
```
104/104 [==============================] - ETA: 0s - loss: 2.3675 - accura
cy: 0.9353
Epoch 10: val_accuracy did not improve from 0.63706
104/104 [==============================] - 363s 3s/step - loss: 2.3675 - a
ccuracy: 0.9353 - val_loss: 4.0587 - val_accuracy: 0.5406
```

In [42]:

```
plt.figure(figsize = (20,7))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training loss', 'validation loss'])
plt.xlabel('Epochs')
plt.ylabel('Losses')
plt.title('Loss vs Epochs', fontsize = 15)
```

Out[42]:

```
Text(0.5, 1.0, 'Loss vs Epochs')
```

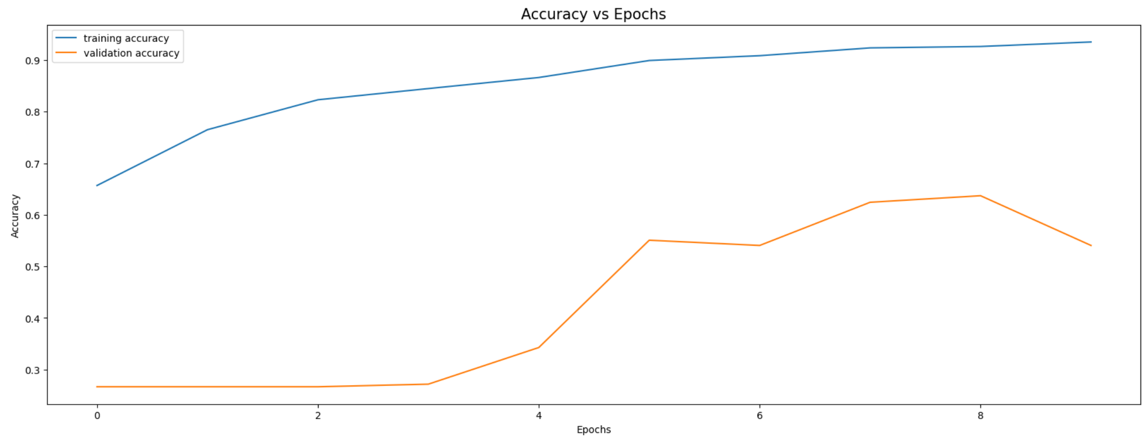


In [43]:

```
plt.figure(figsize = (20,7))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training accuracy', 'validation accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Epochs', fontsize = 15)
```

Out[43]:

```
Text(0.5, 1.0, 'Accuracy vs Epochs')
```



In [44]:

```
pred = model.predict(X_test)
```

```
13/13 [==============================] - 11s 810ms/step
```

In [45]:

```
print(pred[0:3,:])
```

```
[[0.00172452 0.0025041  0.9909849  0.00478642]
 [0.03391148 0.01550729 0.9463521  0.00422918]
 [0.01615475 0.02810627 0.95155895 0.00417996]]
```

In [46]:

```
temp = np.argmax(pred, axis = 1)
pred = np.zeros(pred.shape)
pred[np.arange(pred.shape[0]), temp] = 1
```

In [47]:

```
print(pred[0:3,:])
print(y_test[0:3,:])
```

```
[[0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]]
[[1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]]
```

In [53]:

```
from sklearn.metrics import accuracy_score, classification_report
# print('Accuracy : ' + str(accuracy_score(y_test, pred)))
print(classification_report(y_test, pred, target_names = ['glioma_tumor', 'meningioma_tu
```

```
                  precision    recall  f1-score   support

    glioma_tumor       0.63      0.19      0.29       100
meningioma_tumor       0.72      0.47      0.57       115
        no_tumor       0.41      0.99      0.58       105
 pituitary_tumor       0.95      0.49      0.64        74

       micro avg       0.54      0.54      0.54       394
       macro avg       0.68      0.53      0.52       394
    weighted avg       0.66      0.54      0.52       394
     samples avg       0.54      0.54      0.54       394
```

In [49]:

```
X_test
```

Out[49]:

```
array([[[[0.        ],
         [0.        ],
         [0.        ],
         ...,
         [0.        ],
         [0.        ],
         [0.        ]],

        [[0.        ],
         [0.        ],
         [0.        ],
         ...,
         [0.        ],
         [0.        ],
         [0.        ]],

        [[0.        ],
         [0.        ],
```

In [60]:

```
model.fit(X_train,y_train,batch_size = 5,epochs = 20, validation_data=(X_test, y_test))
```

```
Epoch 1/20
662/662 [==============================] - 399s 603ms/step - loss: 2.3405
- accuracy: 0.7440 - val_loss: 3.7138 - val_accuracy: 0.5051
Epoch 2/20
662/662 [==============================] - 400s 604ms/step - loss: 2.1671
- accuracy: 0.7693 - val_loss: 3.3699 - val_accuracy: 0.6244
Epoch 3/20
662/662 [==============================] - 400s 604ms/step - loss: 2.0471
- accuracy: 0.7693 - val_loss: 3.7208 - val_accuracy: 0.6041
Epoch 4/20
662/662 [==============================] - 403s 609ms/step - loss: 1.8953
- accuracy: 0.7923 - val_loss: 3.2207 - val_accuracy: 0.6396
Epoch 5/20
662/662 [==============================] - 400s 605ms/step - loss: 1.7301
- accuracy: 0.8123 - val_loss: 3.4077 - val_accuracy: 0.6142
Epoch 6/20
662/662 [==============================] - 402s 607ms/step - loss: 1.6145
- accuracy: 0.8274 - val_loss: 3.1036 - val_accuracy: 0.6244
Epoch 7/20
662/662 [==============================] - 446s 673ms/step - loss: 1.5557
- accuracy: 0.8277 - val_loss: 3.1076 - val_accuracy: 0.6117
Epoch 8/20
662/662 [==============================] - 452s 683ms/step - loss: 1.5388
- accuracy: 0.8141 - val_loss: 2.9754 - val_accuracy: 0.6168
Epoch 9/20
662/662 [==============================] - 448s 676ms/step - loss: 1.5088
- accuracy: 0.8289 - val_loss: 3.6059 - val_accuracy: 0.5025
Epoch 10/20
662/662 [==============================] - 455s 688ms/step - loss: 1.4974
- accuracy: 0.8340 - val_loss: 3.2832 - val_accuracy: 0.5711
Epoch 11/20
662/662 [==============================] - 450s 680ms/step - loss: 1.4365
- accuracy: 0.8392 - val_loss: 2.5625 - val_accuracy: 0.6574
Epoch 12/20
662/662 [==============================] - 455s 688ms/step - loss: 1.3731
- accuracy: 0.8570 - val_loss: 3.1880 - val_accuracy: 0.6751
Epoch 13/20
662/662 [==============================] - 450s 679ms/step - loss: 1.3683
- accuracy: 0.8495 - val_loss: 2.5654 - val_accuracy: 0.6472
Epoch 14/20
662/662 [==============================] - 444s 671ms/step - loss: 1.2681
- accuracy: 0.8752 - val_loss: 2.7535 - val_accuracy: 0.6853
Epoch 15/20
662/662 [==============================] - 571s 863ms/step - loss: 1.2269
- accuracy: 0.8703 - val_loss: 3.6338 - val_accuracy: 0.6345
Epoch 16/20
662/662 [==============================] - 701s 1s/step - loss: 1.1777 - a
ccuracy: 0.8872 - val_loss: 2.9320 - val_accuracy: 0.6345
Epoch 17/20
662/662 [==============================] - 683s 1s/step - loss: 1.1470 - a
ccuracy: 0.8915 - val_loss: 3.2240 - val_accuracy: 0.6091
Epoch 18/20
662/662 [==============================] - 415s 627ms/step - loss: 1.0968
- accuracy: 0.8984 - val_loss: 2.0707 - val_accuracy: 0.7183
Epoch 19/20
662/662 [==============================] - 409s 618ms/step - loss: 1.0629
- accuracy: 0.8981 - val_loss: 2.6258 - val_accuracy: 0.6929
Epoch 20/20
662/662 [==============================] - 409s 618ms/step - loss: 1.0179
- accuracy: 0.9048 - val_loss: 2.3284 - val_accuracy: 0.6650
```

Out[60]:

```
<keras.callbacks.History at 0x1c289d034c0>
```

In [61]:

```python
pred = model.predict(X_test)
```

```
13/13 [==============================] - 10s 794ms/step
```

In [62]:

```python
print(pred[0:3,:])
```

```
[[0.01014137 0.03531386 0.23353335 0.7210114 ]
 [0.74444145 0.10824066 0.14584027 0.00147763]
 [0.01636343 0.02251399 0.95623314 0.00488941]]
```

In [63]:

```python
temp = np.argmax(pred, axis = 1)
pred = np.zeros(pred.shape)
pred[np.arange(pred.shape[0]), temp] = 1
```

In [64]:

```python
print(pred[0:3,:])
print(y_test[0:3,:])
```

```
[[0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]]
[[1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]]
```

In [65]:

```python
from sklearn.metrics import accuracy_score, classification_report
print('Accuracy : ' + str(accuracy_score(y_test, pred)))
print(classification_report(y_test, pred, target_names = ['glioma_tumor', 'meningioma_tu
```

```
Accuracy : 0.6649746192893401
                  precision    recall  f1-score   support

    glioma_tumor       0.88      0.22      0.35       100
meningioma_tumor       0.77      0.88      0.82       115
        no_tumor       0.53      1.00      0.69       105
 pituitary_tumor       0.92      0.46      0.61        74

       micro avg       0.66      0.66      0.66       394
       macro avg       0.77      0.64      0.62       394
    weighted avg       0.76      0.66      0.63       394
     samples avg       0.66      0.66      0.66       394
```

In [ ]:

```

```