HW3 CONVOLUTIONAL NEURAL NETWORKS - IMAGE CLASSIFICATION (GAME OF THRONES DATASET) - ADITYA TORNEKAR

In [40]:

```
#Unzipping the image zip file renamed as GameOfThrones.zip
!unzip GameOfThrones.zip
```

Path changes might be required as per unzipped folder path to reach to the dataset folder which consists of test and train directory

Below path is used as this code was built on Google Colab which has the below directory structure

In [2]:

```
import os
print(os.listdir('/content/dataset'))
```

```
['test', 'train']
```

In [3]:

```
#Importing Libraries
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matri
x,roc_curve,roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator

from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.utils import to_categorical

# CNN libraries
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
import random as rn

import cv2
import numpy as np
from tqdm import tqdm
import os
from random import shuffle
from zipfile import ZipFile
from PIL import Image
```

In [4]:

```
#Defining directories and lists to consume images
X=[]
Z=[]
IMG_SIZE=150
JAIMIE_DIR='/content/dataset/train/Jaimie'
TYRION_DIR='/content/dataset/train/Tyrion'
ARYA_DIR='/content/dataset/train/arya'
CERSIE_DIR='/content/dataset/train/cersie'
DANERYS_DIR='/content/dataset/train/danerys'
JOHN_DIR='/content/dataset/train/john'
NED_DIR='/content/dataset/train/ned stark'
PETER_DIR='/content/dataset/train/peter baelish'
SANSA_DIR='/content/dataset/train/sansa'
```

In [5]:

```
#Defining functions to read directory and assign label accordingly
def assign_label(img,got_type):
    return got_type

def make_train_data(got_type,DIR):
    for img in tqdm(os.listdir(DIR)):
        label=assign_label(img,got_type)
        path = os.path.join(DIR,img)
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

        X.append(np.array(img))
        Z.append(str(label))
```

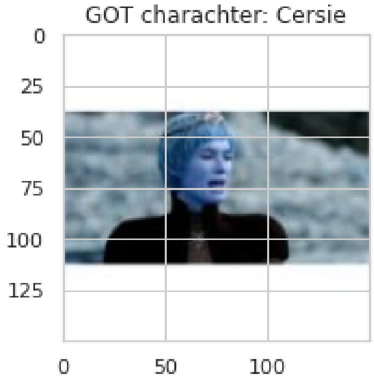
In [6]:

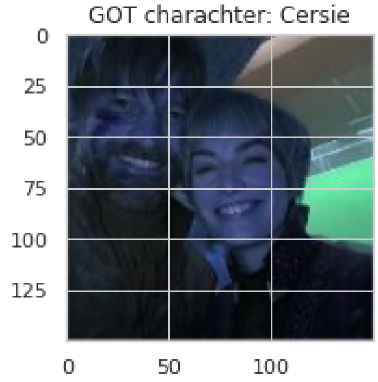
```
#Calling functions to assign labels as per image directories
make_train_data('Jaimie',JAIMIE_DIR)
print(len(X))

make_train_data('Tyrion',TYRION_DIR)
print(len(X))

make_train_data('Arya',ARYA_DIR)
print(len(X))

make_train_data('Cersie',CERSIE_DIR)
print(len(X))

make_train_data('Danerys',DANERYS_DIR)
print(len(X))

make_train_data('John',JOHN_DIR)
print(len(X))

make_train_data('Ned',NED_DIR)
print(len(X))

make_train_data('Peter',PETER_DIR)
print(len(X))

make_train_data('Sansa',SANSA_DIR)
print(len(X))
```

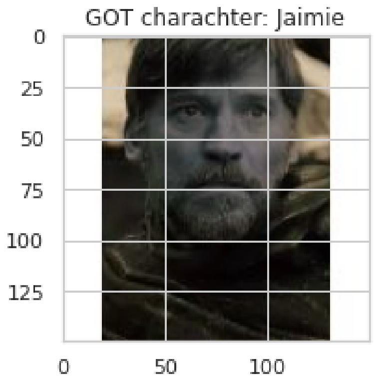
```
100%|██████████| 63/63 [00:00<00:00, 1036.83it/s]
100%|██████████| 63/63 [00:00<00:00, 2000.33it/s]
100%|██████████| 97/97 [00:00<00:00, 2153.42it/s]
100%|██████████| 81/81 [00:00<00:00, 2161.04it/s]
100%|██████████| 70/70 [00:00<00:00, 2044.04it/s]
100%|██████████| 56/56 [00:00<00:00, 2052.62it/s]
  0%|          | 0/73 [00:00<?, ?it/s]

63
126
223
304
374
430

100%|██████████| 73/73 [00:00<00:00, 1929.97it/s]
100%|██████████| 57/57 [00:00<00:00, 2090.73it/s]
100%|██████████| 103/103 [00:00<00:00, 2148.30it/s]

503
560
663
```

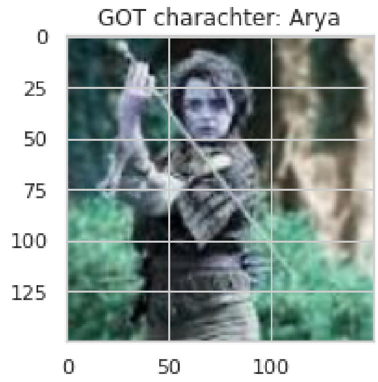
In [7]:

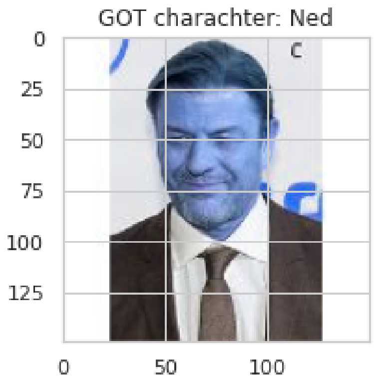
```
#Ploting images to check the data has been read correctly
fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (2):
        l=rn.randint(0,len(Z))
        ax[i,j].imshow(X[l])
        ax[i,j].set_title('GOT charachter: '+Z[l])

plt.tight_layout()
```

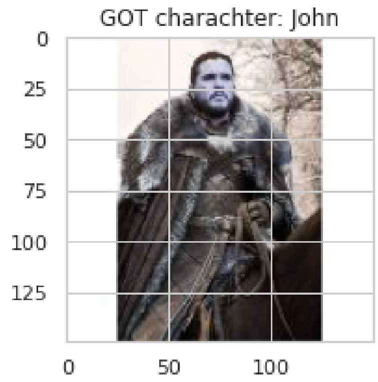

GOT charachter: Cersie

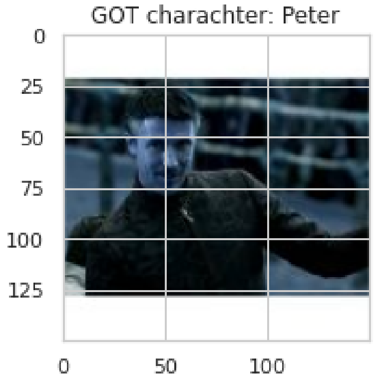

GOT charachter: Cersie

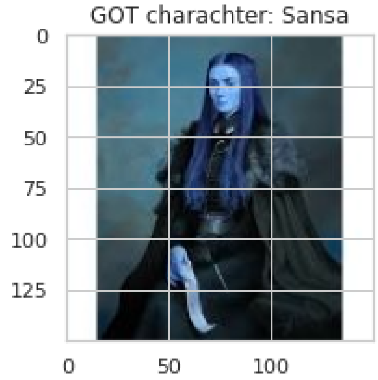

GOT charachter: Jaimie

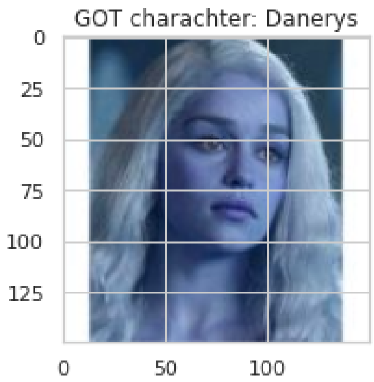

GOT charachter: Arya

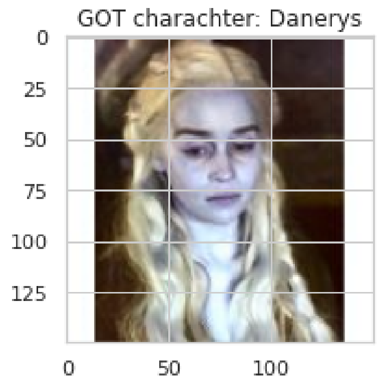

GOT charachter: Ned


GOT charachter: John


GOT charachter: Peter


GOT charachter: Sansa


GOT charachter: Danerys


GOT charachter: Danerys

In [8]:

```
#Assigning numerical values to categorical labels
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,9)
X=np.array(X)
X=X/255
```

In [9]:

```
#Splting train dataset into train and validation
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=15)
```

In [10]:

```
#Building CNN model
np.random.seed(15)
rn.seed(15)
tf.random.set_seed(15)

model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='relu',
input_shape = (150,150,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'
))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 150, kernel_size = (3,3),padding = 'Same',activation ='relu'
))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(500))
model.add(Activation('relu'))

model.add(Dense(9, activation = "softmax"))
```

After testing multiple configurations, the above model configuration further gave decent results after adding dropout and adjusting filters. We can also use Gridsearch for tuning the hyperparameters.

In [11]:

```
#Data generator - Data Augumentation
imagedatagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        vertical_flip=False)


imagedatagen.fit(x_train)
```

The above data augumentation process is performed to reduce the overfitting

In [12]:

```
model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accura
cy'])
```

In [13]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      2432      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 75, 75, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 37, 37, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 96)        55392     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 96)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 18, 18, 96)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 150)       129750    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 150)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 9, 9, 150)         0         
_________________________________________________________________
flatten (Flatten)            (None, 12150)             0         
_________________________________________________________________
dense (Dense)                (None, 500)               6075500   
_________________________________________________________________
activation (Activation)      (None, 500)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 4509      
=================================================================
Total params: 6,286,079
Trainable params: 6,286,079
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

```
batch_size=128
epochs=100
```

In [23]:

```
History = model.fit(imagedatagen.flow(x_train,y_train, batch_size=batch_size), epochs =
epochs, validation_data = (x_test,y_test),verbose = 1, steps_per_epoch=x_train.shape[0]
// batch_size)
```

```
Epoch 1/100
3/3 [==============================] - 15s 5s/step - loss: 1.8575 - accura
cy: 0.3008 - val_loss: 1.9916 - val_accuracy: 0.2892
Epoch 2/100
3/3 [==============================] - 15s 5s/step - loss: 1.9060 - accura
cy: 0.2760 - val_loss: 1.9683 - val_accuracy: 0.3133
Epoch 3/100
3/3 [==============================] - 14s 5s/step - loss: 1.8611 - accura
cy: 0.3415 - val_loss: 1.9446 - val_accuracy: 0.3193
Epoch 4/100
3/3 [==============================] - 14s 5s/step - loss: 1.8501 - accura
cy: 0.3252 - val_loss: 1.9538 - val_accuracy: 0.2892
Epoch 5/100
3/3 [==============================] - 15s 5s/step - loss: 1.8302 - accura
cy: 0.3252 - val_loss: 1.9590 - val_accuracy: 0.2892
Epoch 6/100
3/3 [==============================] - 14s 5s/step - loss: 1.8199 - accura
cy: 0.3279 - val_loss: 2.0214 - val_accuracy: 0.2831
Epoch 7/100
3/3 [==============================] - 15s 5s/step - loss: 1.8708 - accura
cy: 0.3388 - val_loss: 1.9922 - val_accuracy: 0.2711
Epoch 8/100
3/3 [==============================] - 15s 5s/step - loss: 1.8648 - accura
cy: 0.3047 - val_loss: 1.9594 - val_accuracy: 0.2651
Epoch 9/100
3/3 [==============================] - 14s 5s/step - loss: 1.8253 - accura
cy: 0.3388 - val_loss: 1.9580 - val_accuracy: 0.3012
Epoch 10/100
3/3 [==============================] - 15s 5s/step - loss: 1.8254 - accura
cy: 0.3469 - val_loss: 1.9544 - val_accuracy: 0.2952
Epoch 11/100
3/3 [==============================] - 14s 5s/step - loss: 1.7820 - accura
cy: 0.3496 - val_loss: 1.9566 - val_accuracy: 0.3193
Epoch 12/100
3/3 [==============================] - 14s 5s/step - loss: 1.8501 - accura
cy: 0.3117 - val_loss: 1.9051 - val_accuracy: 0.3253
Epoch 13/100
3/3 [==============================] - 15s 5s/step - loss: 1.7573 - accura
cy: 0.3659 - val_loss: 1.8559 - val_accuracy: 0.3072
Epoch 14/100
3/3 [==============================] - 14s 5s/step - loss: 1.7281 - accura
cy: 0.3659 - val_loss: 1.8740 - val_accuracy: 0.3133
Epoch 15/100
3/3 [==============================] - 15s 5s/step - loss: 1.7041 - accura
cy: 0.3672 - val_loss: 1.8795 - val_accuracy: 0.3614
Epoch 16/100
3/3 [==============================] - 14s 5s/step - loss: 1.7119 - accura
cy: 0.3550 - val_loss: 1.8567 - val_accuracy: 0.3434
Epoch 17/100
3/3 [==============================] - 15s 5s/step - loss: 1.7103 - accura
cy: 0.3496 - val_loss: 1.8399 - val_accuracy: 0.3434
Epoch 18/100
3/3 [==============================] - 14s 5s/step - loss: 1.7331 - accura
cy: 0.3577 - val_loss: 1.8188 - val_accuracy: 0.3675
Epoch 19/100
3/3 [==============================] - 14s 5s/step - loss: 1.6744 - accura
cy: 0.3686 - val_loss: 1.8126 - val_accuracy: 0.3855
Epoch 20/100
3/3 [==============================] - 14s 5s/step - loss: 1.6544 - accura
cy: 0.3631 - val_loss: 1.8169 - val_accuracy: 0.3735
Epoch 21/100
```

```
3/3 [==============================] - 15s 5s/step - loss: 1.6160 - accura
cy: 0.3932 - val_loss: 1.8033 - val_accuracy: 0.3494
Epoch 22/100
3/3 [==============================] - 15s 5s/step - loss: 1.6410 - accura
cy: 0.3958 - val_loss: 1.8094 - val_accuracy: 0.3735
Epoch 23/100
3/3 [==============================] - 15s 5s/step - loss: 1.5792 - accura
cy: 0.4255 - val_loss: 1.8446 - val_accuracy: 0.3373
Epoch 24/100
3/3 [==============================] - 15s 5s/step - loss: 1.6596 - accura
cy: 0.4173 - val_loss: 1.8430 - val_accuracy: 0.3434
Epoch 25/100
3/3 [==============================] - 15s 5s/step - loss: 1.7189 - accura
cy: 0.3385 - val_loss: 1.8198 - val_accuracy: 0.3976
Epoch 26/100
3/3 [==============================] - 14s 5s/step - loss: 1.6692 - accura
cy: 0.4119 - val_loss: 1.8830 - val_accuracy: 0.3855
Epoch 27/100
3/3 [==============================] - 15s 5s/step - loss: 1.6754 - accura
cy: 0.3932 - val_loss: 1.8538 - val_accuracy: 0.3735
Epoch 28/100
3/3 [==============================] - 14s 5s/step - loss: 1.6454 - accura
cy: 0.3875 - val_loss: 1.8633 - val_accuracy: 0.3675
Epoch 29/100
3/3 [==============================] - 14s 5s/step - loss: 1.6421 - accura
cy: 0.3902 - val_loss: 1.7951 - val_accuracy: 0.3133
Epoch 30/100
3/3 [==============================] - 15s 5s/step - loss: 1.5983 - accura
cy: 0.4115 - val_loss: 1.8236 - val_accuracy: 0.3133
Epoch 31/100
3/3 [==============================] - 14s 5s/step - loss: 1.5685 - accura
cy: 0.4336 - val_loss: 1.8068 - val_accuracy: 0.3675
Epoch 32/100
3/3 [==============================] - 15s 5s/step - loss: 1.5653 - accura
cy: 0.4089 - val_loss: 1.7755 - val_accuracy: 0.3735
Epoch 33/100
3/3 [==============================] - 15s 5s/step - loss: 1.5527 - accura
cy: 0.3984 - val_loss: 1.7675 - val_accuracy: 0.3494
Epoch 34/100
3/3 [==============================] - 15s 5s/step - loss: 1.5381 - accura
cy: 0.4553 - val_loss: 1.7635 - val_accuracy: 0.3675
Epoch 35/100
3/3 [==============================] - 15s 5s/step - loss: 1.5547 - accura
cy: 0.4245 - val_loss: 1.7846 - val_accuracy: 0.3614
Epoch 36/100
3/3 [==============================] - 15s 5s/step - loss: 1.5504 - accura
cy: 0.4479 - val_loss: 1.7841 - val_accuracy: 0.4096
Epoch 37/100
3/3 [==============================] - 14s 5s/step - loss: 1.5260 - accura
cy: 0.4228 - val_loss: 1.7677 - val_accuracy: 0.3795
Epoch 38/100
3/3 [==============================] - 14s 5s/step - loss: 1.4957 - accura
cy: 0.4661 - val_loss: 1.7197 - val_accuracy: 0.3916
Epoch 39/100
3/3 [==============================] - 14s 5s/step - loss: 1.5756 - accura
cy: 0.4201 - val_loss: 1.7397 - val_accuracy: 0.3976
Epoch 40/100
3/3 [==============================] - 15s 5s/step - loss: 1.4934 - accura
cy: 0.4634 - val_loss: 1.7503 - val_accuracy: 0.3373
Epoch 41/100
3/3 [==============================] - 15s 5s/step - loss: 1.5096 - accura
```

```
cy: 0.4499 - val_loss: 1.7997 - val_accuracy: 0.3614
Epoch 42/100
3/3 [==============================] - 14s 5s/step - loss: 1.4453 - accura
cy: 0.4417 - val_loss: 1.7412 - val_accuracy: 0.3916
Epoch 43/100
3/3 [==============================] - 14s 5s/step - loss: 1.4630 - accura
cy: 0.4472 - val_loss: 1.7506 - val_accuracy: 0.3916
Epoch 44/100
3/3 [==============================] - 14s 5s/step - loss: 1.4343 - accura
cy: 0.5068 - val_loss: 1.8283 - val_accuracy: 0.3735
Epoch 45/100
3/3 [==============================] - 14s 5s/step - loss: 1.4613 - accura
cy: 0.4743 - val_loss: 1.6970 - val_accuracy: 0.4157
Epoch 46/100
3/3 [==============================] - 15s 5s/step - loss: 1.4076 - accura
cy: 0.4714 - val_loss: 1.6865 - val_accuracy: 0.4217
Epoch 47/100
3/3 [==============================] - 15s 5s/step - loss: 1.3879 - accura
cy: 0.4740 - val_loss: 1.7083 - val_accuracy: 0.4096
Epoch 48/100
3/3 [==============================] - 14s 5s/step - loss: 1.3390 - accura
cy: 0.4932 - val_loss: 1.6654 - val_accuracy: 0.4458
Epoch 49/100
3/3 [==============================] - 14s 5s/step - loss: 1.3444 - accura
cy: 0.5149 - val_loss: 1.7135 - val_accuracy: 0.4217
Epoch 50/100
3/3 [==============================] - 14s 5s/step - loss: 1.3445 - accura
cy: 0.5122 - val_loss: 1.6699 - val_accuracy: 0.4398
Epoch 51/100
3/3 [==============================] - 14s 5s/step - loss: 1.3332 - accura
cy: 0.5176 - val_loss: 1.7317 - val_accuracy: 0.4398
Epoch 52/100
3/3 [==============================] - 15s 5s/step - loss: 1.3547 - accura
cy: 0.4986 - val_loss: 1.6769 - val_accuracy: 0.4337
Epoch 53/100
3/3 [==============================] - 14s 5s/step - loss: 1.2830 - accura
cy: 0.5122 - val_loss: 1.7656 - val_accuracy: 0.3976
Epoch 54/100
3/3 [==============================] - 14s 5s/step - loss: 1.3448 - accura
cy: 0.5122 - val_loss: 1.7066 - val_accuracy: 0.4458
Epoch 55/100
3/3 [==============================] - 15s 5s/step - loss: 1.2967 - accura
cy: 0.5312 - val_loss: 1.6792 - val_accuracy: 0.4277
Epoch 56/100
3/3 [==============================] - 14s 5s/step - loss: 1.2750 - accura
cy: 0.5420 - val_loss: 1.7358 - val_accuracy: 0.4096
Epoch 57/100
3/3 [==============================] - 14s 5s/step - loss: 1.2577 - accura
cy: 0.5474 - val_loss: 1.7330 - val_accuracy: 0.4096
Epoch 58/100
3/3 [==============================] - 14s 5s/step - loss: 1.2694 - accura
cy: 0.5285 - val_loss: 1.7195 - val_accuracy: 0.4398
Epoch 59/100
3/3 [==============================] - 14s 5s/step - loss: 1.2619 - accura
cy: 0.5312 - val_loss: 1.8672 - val_accuracy: 0.3916
Epoch 60/100
3/3 [==============================] - 14s 5s/step - loss: 1.2471 - accura
cy: 0.5393 - val_loss: 1.7578 - val_accuracy: 0.4458
Epoch 61/100
3/3 [==============================] - 15s 5s/step - loss: 1.2570 - accura
cy: 0.5807 - val_loss: 1.6948 - val_accuracy: 0.4337
```

```
Epoch 62/100
3/3 [==============================] - 14s 5s/step - loss: 1.2325 - accura
cy: 0.5664 - val_loss: 1.6882 - val_accuracy: 0.4157
Epoch 63/100
3/3 [==============================] - 15s 5s/step - loss: 1.1727 - accura
cy: 0.5859 - val_loss: 1.7021 - val_accuracy: 0.4337
Epoch 64/100
3/3 [==============================] - 14s 5s/step - loss: 1.2462 - accura
cy: 0.5637 - val_loss: 1.7128 - val_accuracy: 0.4337
Epoch 65/100
3/3 [==============================] - 15s 5s/step - loss: 1.2161 - accura
cy: 0.5573 - val_loss: 1.6835 - val_accuracy: 0.4217
Epoch 66/100
3/3 [==============================] - 14s 5s/step - loss: 1.1271 - accura
cy: 0.5989 - val_loss: 1.6351 - val_accuracy: 0.4458
Epoch 67/100
3/3 [==============================] - 14s 5s/step - loss: 1.1656 - accura
cy: 0.5989 - val_loss: 1.7773 - val_accuracy: 0.4337
Epoch 68/100
3/3 [==============================] - 15s 5s/step - loss: 1.1763 - accura
cy: 0.5664 - val_loss: 1.7565 - val_accuracy: 0.4277
Epoch 69/100
3/3 [==============================] - 15s 5s/step - loss: 1.1542 - accura
cy: 0.5703 - val_loss: 1.8224 - val_accuracy: 0.4157
Epoch 70/100
3/3 [==============================] - 14s 5s/step - loss: 1.1015 - accura
cy: 0.6125 - val_loss: 1.6450 - val_accuracy: 0.4578
Epoch 71/100
3/3 [==============================] - 14s 5s/step - loss: 1.1886 - accura
cy: 0.5447 - val_loss: 1.7720 - val_accuracy: 0.4217
Epoch 72/100
3/3 [==============================] - 14s 5s/step - loss: 1.1306 - accura
cy: 0.5989 - val_loss: 1.7363 - val_accuracy: 0.4518
Epoch 73/100
3/3 [==============================] - 14s 5s/step - loss: 1.1393 - accura
cy: 0.5583 - val_loss: 1.6339 - val_accuracy: 0.4217
Epoch 74/100
3/3 [==============================] - 14s 5s/step - loss: 1.1186 - accura
cy: 0.6043 - val_loss: 1.6497 - val_accuracy: 0.4639
Epoch 75/100
3/3 [==============================] - 15s 5s/step - loss: 1.1950 - accura
cy: 0.5755 - val_loss: 1.6169 - val_accuracy: 0.5000
Epoch 76/100
3/3 [==============================] - 14s 5s/step - loss: 1.0754 - accura
cy: 0.6179 - val_loss: 1.6336 - val_accuracy: 0.4639
Epoch 77/100
3/3 [==============================] - 15s 5s/step - loss: 1.0828 - accura
cy: 0.6179 - val_loss: 1.6625 - val_accuracy: 0.4819
Epoch 78/100
3/3 [==============================] - 15s 5s/step - loss: 1.1247 - accura
cy: 0.6094 - val_loss: 1.6567 - val_accuracy: 0.5241
Epoch 79/100
3/3 [==============================] - 14s 5s/step - loss: 1.0646 - accura
cy: 0.5962 - val_loss: 1.5890 - val_accuracy: 0.5120
Epoch 80/100
3/3 [==============================] - 15s 5s/step - loss: 1.1026 - accura
cy: 0.6094 - val_loss: 1.5857 - val_accuracy: 0.5000
Epoch 81/100
3/3 [==============================] - 14s 5s/step - loss: 0.9869 - accura
cy: 0.6477 - val_loss: 1.6138 - val_accuracy: 0.5120
Epoch 82/100
```

```
3/3 [==============================] - 14s 5s/step - loss: 0.9696 - accura
cy: 0.6450 - val_loss: 1.7382 - val_accuracy: 0.5060
Epoch 83/100
3/3 [==============================] - 14s 5s/step - loss: 1.0379 - accura
cy: 0.6287 - val_loss: 1.6544 - val_accuracy: 0.4880
Epoch 84/100
3/3 [==============================] - 14s 5s/step - loss: 1.0451 - accura
cy: 0.6504 - val_loss: 1.6451 - val_accuracy: 0.5120
Epoch 85/100
3/3 [==============================] - 15s 5s/step - loss: 1.0576 - accura
cy: 0.6179 - val_loss: 1.6919 - val_accuracy: 0.4819
Epoch 86/100
3/3 [==============================] - 15s 5s/step - loss: 1.0311 - accura
cy: 0.6260 - val_loss: 1.6100 - val_accuracy: 0.5361
Epoch 87/100
3/3 [==============================] - 15s 5s/step - loss: 1.0526 - accura
cy: 0.6146 - val_loss: 1.6039 - val_accuracy: 0.5120
Epoch 88/100
3/3 [==============================] - 15s 5s/step - loss: 0.9896 - accura
cy: 0.6504 - val_loss: 1.7573 - val_accuracy: 0.4639
Epoch 89/100
3/3 [==============================] - 14s 5s/step - loss: 0.9403 - accura
cy: 0.6558 - val_loss: 1.6617 - val_accuracy: 0.5241
Epoch 90/100
3/3 [==============================] - 14s 5s/step - loss: 1.0899 - accura
cy: 0.5908 - val_loss: 1.7484 - val_accuracy: 0.4880
Epoch 91/100
3/3 [==============================] - 14s 5s/step - loss: 0.9651 - accura
cy: 0.6721 - val_loss: 1.6497 - val_accuracy: 0.5181
Epoch 92/100
3/3 [==============================] - 15s 5s/step - loss: 0.9766 - accura
cy: 0.6510 - val_loss: 1.7180 - val_accuracy: 0.5361
Epoch 93/100
3/3 [==============================] - 15s 5s/step - loss: 0.9824 - accura
cy: 0.6585 - val_loss: 1.6721 - val_accuracy: 0.5181
Epoch 94/100
3/3 [==============================] - 15s 5s/step - loss: 0.9761 - accura
cy: 0.6531 - val_loss: 1.6843 - val_accuracy: 0.5000
Epoch 95/100
3/3 [==============================] - 15s 5s/step - loss: 0.8948 - accura
cy: 0.6721 - val_loss: 1.6821 - val_accuracy: 0.5181
Epoch 96/100
3/3 [==============================] - 15s 5s/step - loss: 0.9243 - accura
cy: 0.6927 - val_loss: 1.6473 - val_accuracy: 0.5241
Epoch 97/100
3/3 [==============================] - 14s 5s/step - loss: 0.8671 - accura
cy: 0.6748 - val_loss: 1.7115 - val_accuracy: 0.5361
Epoch 98/100
3/3 [==============================] - 15s 5s/step - loss: 0.9110 - accura
cy: 0.6797 - val_loss: 1.6664 - val_accuracy: 0.5181
Epoch 99/100
3/3 [==============================] - 14s 5s/step - loss: 0.9585 - accura
cy: 0.6504 - val_loss: 1.7606 - val_accuracy: 0.5000
Epoch 100/100
3/3 [==============================] - 14s 5s/step - loss: 0.8776 - accura
cy: 0.6883 - val_loss: 1.7131 - val_accuracy: 0.5181
```

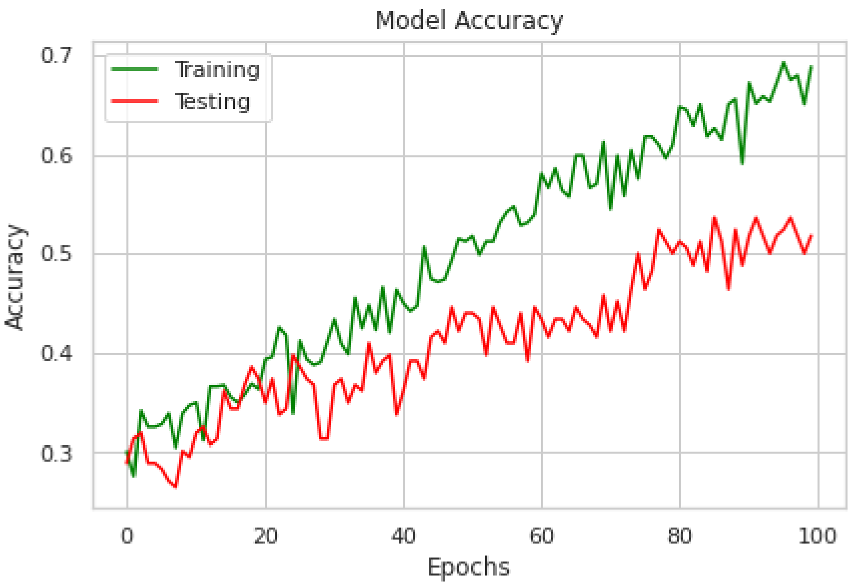
As we can see the validation/test accuracy is almost above 50% and it took 100 epochs and batch size of 128 for decent results.

In [40]:

```
# Plotting Loss
plt.plot(History.history['loss'],color='green')
plt.plot(History.history['val_loss'],'bo',color='red')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Training', 'Testing'],loc='lower left')
plt.show()
```

In [38]:

```
#Plotting accuracy
plt.plot(History.history['accuracy'],color='green')
plt.plot(History.history['val_accuracy'],color='red')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training', 'Testing'])
plt.show()
```



As we can see above the model is performing quite well, as the test accuracy is almost close to the training accuracy over 100 epochs. We can check for other alternate model configurations to further boost the accuracy.

In [26]:

```
# Predictions
pred=model.predict(x_test)
pred_digits=np.argmax(pred,axis=1)
```

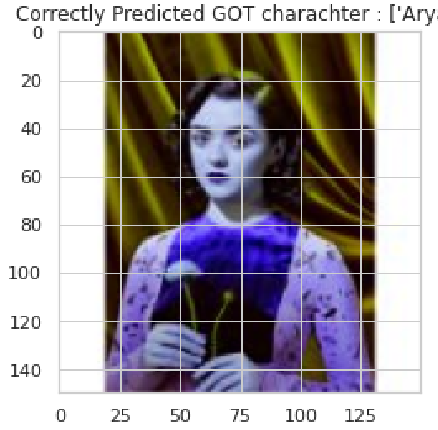
In [27]:

```
# Segregating correctly classified and misclassified images
i=0
prop_class=[]
mis_class=[]

for i in range(len(y_test)):
    if(np.argmax(y_test[i])==pred_digits[i]):
        prop_class.append(i)
    if(len(prop_class)==8):
        break

i=0
for i in range(len(y_test)):
    if(np.argmax(y_test[i])!=pred_digits[i]):
        mis_class.append(i)
    if(len(mis_class)==8):
        break
```

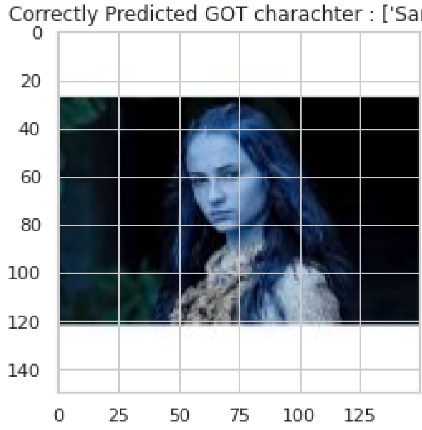
In [28]:

```
#Correctly classified samples
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[prop_class[count]])
        ax[i,j].set_title("Correctly Predicted GOT charachter : "+str(le.inverse_transf
orm([pred_digits[prop_class[count]]])))
        plt.tight_layout()
        count+=1
```

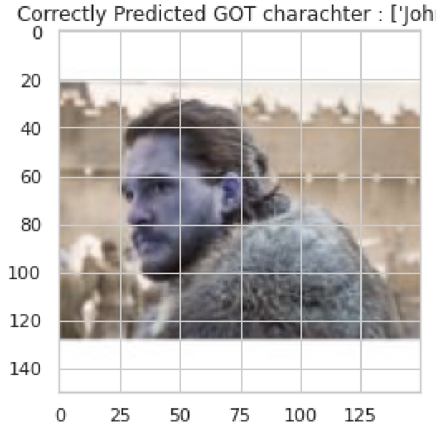
Correctly Predicted GOT charachter : ['Arya']



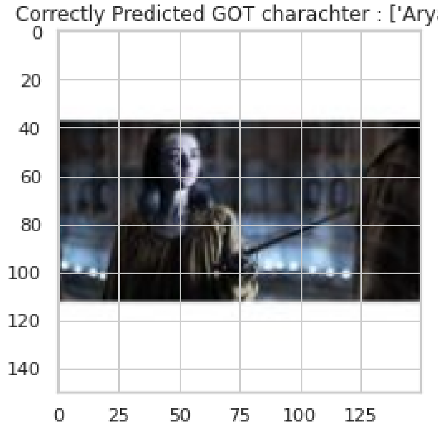
Correctly Predicted GOT charachter : ['Sansa']



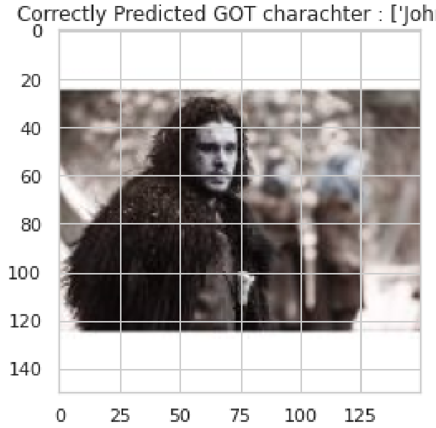
Correctly Predicted GOT charachter : ['John']



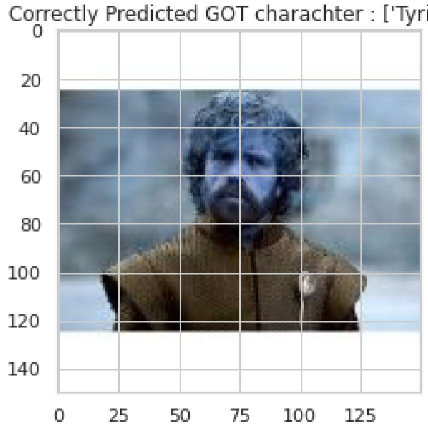
Correctly Predicted GOT charachter : ['Arya']



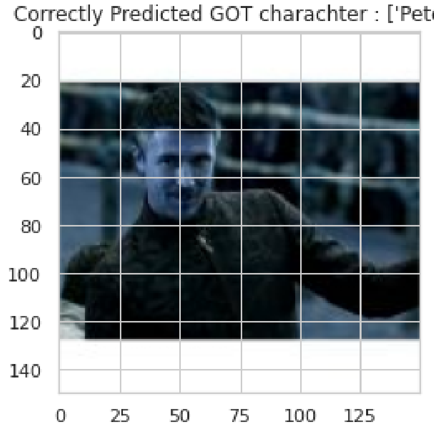
Correctly Predicted GOT charachter : ['John']



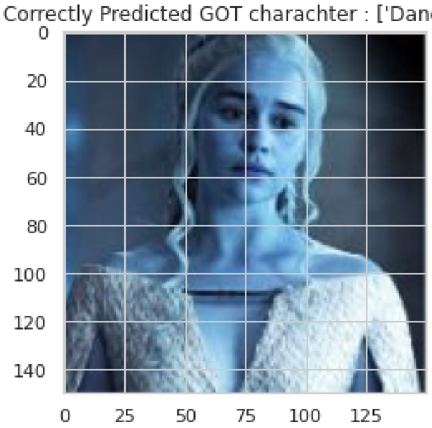
Correctly Predicted GOT charachter : ['Tyrion']



Correctly Predicted GOT charachter : ['Peter']



Correctly Predicted GOT charachter : ['Danerys']

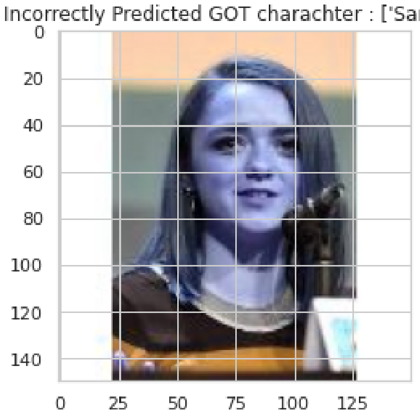


We can see above samples from the set of correctly classified images. More than 50% images were classified into correct labels. Below are some of the misclassified images.We can further check use some decaying learning rate if required for better model performance or bring more data for better training of the model.

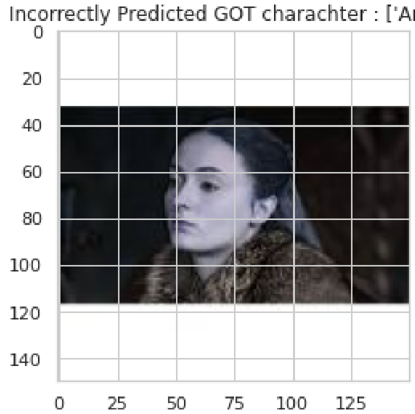
In [29]:

```
#Misclassified samples
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for i in range (4):
    for j in range (2):
        ax[i,j].imshow(x_test[mis_class[count]])
        ax[i,j].set_title("Incorrectly Predicted GOT charachter : "+str(le.inverse_tran
sform([pred_digits[mis_class[count]]])))
        plt.tight_layout()
        count+=1
```

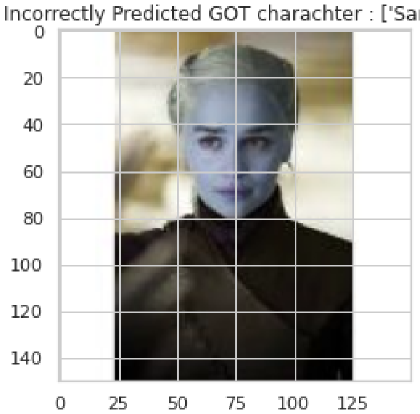
Incorrectly Predicted GOT charachter : ['Sansa']



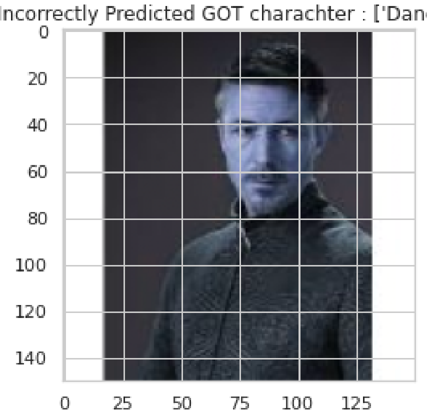
Incorrectly Predicted GOT charachter : ['Arya']



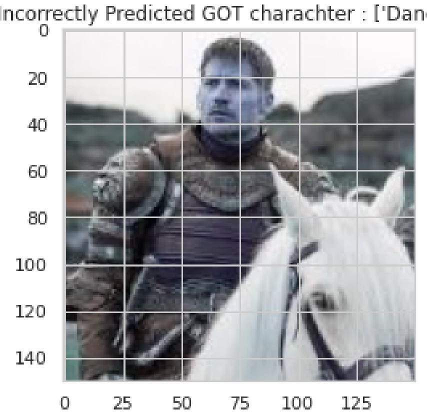
Incorrectly Predicted GOT charachter : ['Sansa']



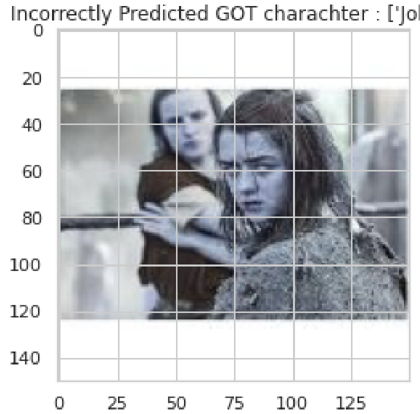
Incorrectly Predicted GOT charachter : ['Danerys']



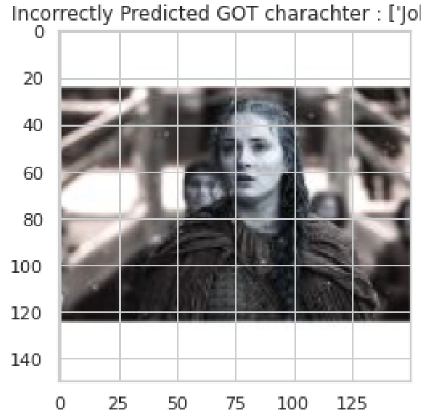
Incorrectly Predicted GOT charachter : ['Danerys']



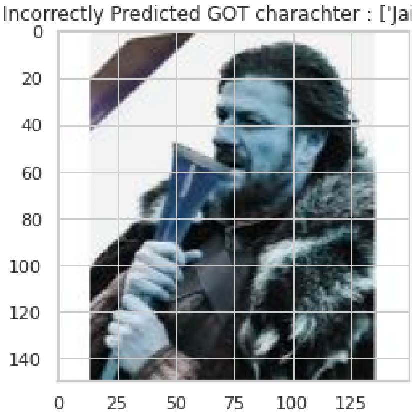
Incorrectly Predicted GOT charachter : ['John']



Incorrectly Predicted GOT charachter : ['John']



Incorrectly Predicted GOT charachter : ['Jaimie']



References:

Data: https://www.kaggle.com/aronighosh/game-of-thrones-character-recognition (https://www.kaggle.com/aronighosh/game-of-thrones-character-recognition)

https://www.tensorflow.org/tutorials/images/cnn (https://www.tensorflow.org/tutorials/images/cnn)

https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/ (https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/)

https://matplotlib.org/3.1.1/index.html (https://matplotlib.org/3.1.1/index.html)

https://keras.io/ (https://keras.io/)

https://www.tensorflow.org/guide/keras/sequential_model (https://www.tensorflow.org/guide/keras/sequential_model)

https://www.kaggle.com/rajmehra03/flower-recognition-cnn-keras (https://www.kaggle.com/rajmehra03/flower-recognition-cnn-keras)