Exploratory Data Analysis

Important Libraries

1. Pandas

Pandas is a Python programming language for **data manipulation** and **data analysis**. In particular, it offers data structures and operations for manipulating numerical tables and time series.

2. Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

3. Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms

4. Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline #%matplotlib inline will lead to static images of your plot embedded in t
he notebook
```

```
UsageError: unrecognized arguments: #%matplotlib inline will lead to static images of your pl
ot embedded in the notebook
```

Objective

Our Objectives are as follows:-

Data cleansing

Data cleansing or **data cleaning** is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

Data wrangling

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.This may include further **munging**, **data visualization**, data aggregation, training a statistical model, as well as many other potential uses.

Metadata

Our Metadata contains 10015 unique image_id but 7470 unique lesion_id. Most of the cancer images are from 'nv' cancer type which is around 67% , 'mel' cancer images are around 11% and other 5 cancer type (bkl,dcl,df,akiec, vasc) are 22% as whole.

In [2]:
```
data=pd.read_csv("C:/Users/imfal/skincancer/image classification project/HAM10000_metadata.c
sv")
data.sort_values("lesion_id", inplace = True)
```

In [3]:
```
data.head() # original data has matrix 10015x07
```

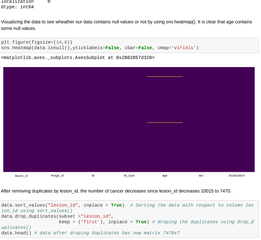
Out[3]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
9197	HAM_0000000	ISIC_0025498	nv	histo	80.0	male	back
9198	HAM_0000000	ISIC_0025340	nv	histo	80.0	male	back
726	HAM_0000001	ISIC_0027959	bkl	histo	70.0	female	face
1661	HAM_0000002	ISIC_0032622	mel	histo	65.0	female	lower extremity
1660	HAM_0000002	ISIC_0033840	mel	histo	65.0	female	lower extremity

By Visualizing data with respect to cancer type dx, which are 7 in numbers. We can clearly see that **nv** group has (around 67%) infected more people in our data.

In [4]:
```
# Looking for which cancer has infected people more.
plt.figure(figsize=(14,6))
sns.set_style('whitegrid')
sns.countplot(x='dx',data=data)
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2826df55e48>
```



isnull()

While making a Data Frame from a csv file, many blank columns are imported as null value into the Data Frame which later creates problems while operating that data frame. Pandas isnull() and notnull() methods are used to check and manage NULL values in a data frame.

In [5]:
```
data.isnull().head() # looking for null values
```

Out[5]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
9197	False	False	False	False	False	False	False
9198	False	False	False	False	False	False	False
726	False	False	False	False	False	False	False
1661	False	False	False	False	False	False	False
1660	False	False	False	False	False	False	False

In [6]:
```
data.isnull().sum() # counting null values column-wise using sum() library with isnull().
                    # And there are 57 null values in column ""age""
```

Out[6]:
```
lesion_id      0
image_id       0
dx             0
dx_type        0
age           57
sex            0
localization   0
dtype: int64
```

Visualizing the data to see wheather our data contains null values or not by using sns.heatmap(). It is clear that age contains some null values.

In [7]:
```
plt.figure(figsize=(14,6))
sns.heatmap(data.isnull(),yticklabels=False, cbar=False, cmap='viridis')
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28628575320>
```


After removing duplicates by lesion_id, the number of cancer decreases since lesion_id decreases 10015 to 7470.

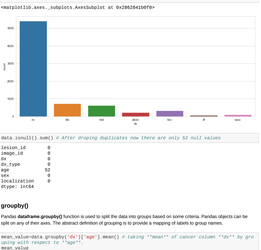
In [8]:
```
data.sort_values("lesion_id", inplace = True) # Sorting the data with respect to column les
ion_id using sort_values()
data.drop_duplicates(subset ='lesion_id',
                     keep = 'first', inplace = True) # dropping the duplicates using drop_d
uplicates()
data.head() # data after droping duplicates has now matrix 7470x7
```

Out[8]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
9197	HAM_0000000	ISIC_0025498	nv	histo	80.0	male	back
726	HAM_0000001	ISIC_0027959	bkl	histo	70.0	female	face
1661	HAM_0000002	ISIC_0032622	mel	histo	65.0	female	lower extremity
3374	HAM_0000003	ISIC_0027880	nv	follow_up	55.0	male	trunk
4918	HAM_0000004	ISIC_0024640	nv	follow_up	40.0	female	back

In [9]:
```
plt.figure(figsize=(14,6))
sns.set_style('whitegrid')
sns.countplot(x='dx',data=data)
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2862841b0f0>
```



In [10]:
```
data.isnull().sum() # After droping duplicates now there are only 52 null values
```

Out[10]:
```
lesion_id      0
image_id       0
dx             0
dx_type        0
age           52
sex            0
localization   0
dtype: int64
```

groupby()

Pandas **dataframe.groupby()** function is used to split the data into groups based on some criteria. Pandas objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.

In [11]:
```
mean_value=data.groupby('dx')['age'].mean() # taking ""mean"" of cancer column ""dx"" by gro
uping with respect to ""age"".
mean_value
```

Out[11]:
```
dx
akiec    60.557018
bcc      66.896624
bkl      64.384158
df       55.000000
mel      61.299389
nv       47.257974
vasc     52.500000
Name: age, dtype: float64
```

In [12]:
```
median_value=data.groupby('dx')['age'].median()
median_value  # taking ""median"" of cancer column ""dx"" by grouping with respect to ""age
```

Out[12]:
```
dx
akiec    67.5
bcc      70.0
bkl      65.0
df       55.0
mel      65.0
nv       45.0
vasc     45.0
Name: age, dtype: float64
```

Filling Null Values

Here it should be notice that filling of null values is taking place in column age with respect to cancer type dx. Meaning, the median of age of akiec group cancer patient will only fill null values of akiec age group and so other.

In [13]:
```
# Filling the null values by using median_value
data["age"] = data.groupby("dx").transform(lambda x: x.fillna(x.median()))
data.head()
```

Out[13]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
9197	HAM_0000000	ISIC_0025498	nv	histo	80.0	male	back
726	HAM_0000001	ISIC_0027959	bkl	histo	70.0	female	face
1661	HAM_0000002	ISIC_0032622	mel	histo	65.0	female	lower extremity
3374	HAM_0000003	ISIC_0027880	nv	follow_up	55.0	male	trunk
4918	HAM_0000004	ISIC_0024640	nv	follow_up	40.0	female	back

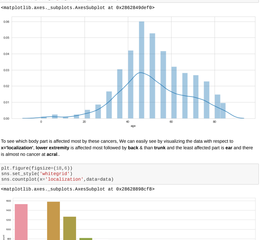
In [14]:
```
data.isnull().sum() # now we can clearly see that there is no null values remaining.
```

Out[14]:
```
lesion_id      0
image_id       0
dx             0
dx_type        0
age            0
sex            0
localization   0
dtype: int64
```

Here,by visualizing distribution plot of age, We can clearly see that the highest number of cancer patient are from the age group (40,60) and highest number of cancer patient are of age 45.

In [15]:
```
plt.figure(figsize=(14,6))
sns.distplot(data['age'],bins=range(0,100,5),kde=True)
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2862849def0>
```



To see which body part is affected most by these cancers, We can easily see by visualizing the data with respect to **x='localization'**, **lower extremity** is affected most followed by **back** & than **trunk** and the least affected part is **ear** and there is almost no cancer as **acral**.

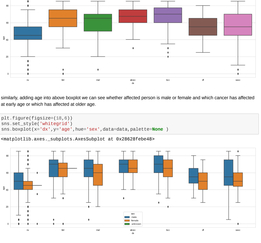
In [16]:
```
plt.figure(figsize=(10,6))
sns.set_style('whitegrid')
sns.countplot(x='localization',data=data)
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28628898cf0>
```

Now, Visualizing the data with respect to **x='localization'** and cancers type **dx**. We can clearly see at **lower extremity nv** cancer has affected most followed by **bkl** & **mel**. Similarly at **back** side **nv** has affected most followed by **mel**.**bkl**. But when we look at **ear** , there we can see this is least affected part of our body and mainly affected by **nv** & **mel** and there is almost no cancer at **acral**.

In [17]:
```
plt.figure(figsize=(10,6))
sns.set_style('whitegrid')
sns.countplot(x='localization',hue='dx',data=data,palette='Reds' )
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2862846abcc0>
```



Now by plotting boxplot with respect to cancers type **x='dx'** with **age**. we can see **nv** has affected people at early age around (40,55), while **vasc** has large age interval of affecting people of age around (45,70) and **bcc** has affected mostly older age people around (60, 77).

In [18]:
```
plt.figure(figsize=(10,6))
sns.set_style('whitegrid')
sns.boxplot(x='dx',y='age',data=data,palette='Reds' )
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28628f0c438>
```


similarly, adding age into above boxplot we can see whether affected person is male or female and which cancer has affected at early age or which has affected at older age.

In [19]:
```
plt.figure(figsize=(10,6))
sns.set_style('whitegrid')
sns.boxplot(x='dx',y='age',hue='sex',data=data,palette='Reds' )
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28628feb4e0>
```

Data Manipulation and Data Wrangling

Data Manipulation is the process of changing data to make it easier to read or more organized.

Data Wrangling is process of transforming and mapping data from one raw data into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics

In [4]:
```
#importing all the important libraies
import cv2
import numpy as np
import pandas as pd
import os
import glob
```

In [5]:
```
# importing metadata from the directory
df = pd.read_csv("HAM10000_metadata.csv")
df.head(5)
```

Out[5]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

In [6]:
```
# to show the categories of lesion
np.unique(df['dx'].tolist())
```

Out[6]:
```
array(['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc'], dtype='<U5')
```

In [7]:
```
def image_to_feature_vector(image, size=(32, 32)):
    # resize the image to a fixed size, then flatten the image into
    # a list of raw pixel intensities
    return cv2.resize(image, size).flatten()
```

In [8]:
```
img_dir = "C:\\Users\\DELL\\3D Objects\\skin-cancer-mnist-ham10000\\HAM10000_images_part_1"
# Enter Directory of all images
data_path = os.path.join(img_dir,'*g')
files = glob.glob(data_path)
data=[]
F1=[]

for f1 in files:
    word_list= f1.split('\\')                          # spliting the path of each file with
"//"
    F1.append(word_list[-1].split('.')[0])             # spliting the image Id (eg.ISIC_0034
320.jpg) of each file with "."
    img = cv2.imread(f1)
    images=image_to_feature_vector(img, size=(32, 32)) # resizeing of an image
    images1=images.tolist()
    data.append(images1)
```

In [9]:
```
# number of images in the data
len(data)
```

Out[9]:
```
10015
```

In [84]:
```
# number of pixels in the data
len(data[0])
```

Out[84]:
```
3072
```

In [91]:
```
a=['image_id']
str1='Pixel_'
for i in range(3072):
   a.append(str1 + str(i))
```

In [66]:
```
# Python3 program to Convert 1D
# list to 2D list
from itertools import islice

def convert(lst, var_lst):
    it = iter(lst)
    F2=[list(islice(it, i)) for i in var_lst]
    return F2

# Driver code
var_lst = [1]*len(F1)
F2=convert(F1, var_lst)
```

In [93]:
```
def merge(lst1, lst2):
    return [a + b for (a, b) in zip(lst1, lst2)]
F3=merge(F2,data)
```

In [94]:
```
df1=pd.DataFrame(F3,columns=a)
df1.head(5)
```

In [114]:
```
# labeling the categories of lesion
def score_to_numeric(x):
    if x=='akiec':
        return 0
    if x=='bcc':
        return 1
    if x=='bkl':
        return 2
    if x=='df':
        return 3
    if x=="mel":
        return 4
    if x=='nv':
        return 5
    if x=='vasc':
        return 6
```

In [116]:
```
#df=df.drop('label',axis=1)
```

In [119]:
```
# merging the metadata with pixels of the images according to the image_ID
df3=pd.merge(df, df1, on='image_id', how='outer')
df3.head(5)
```

Out[119]:

	lesion_id	image_id	dx	dx_type	age	sex	localization	Pixel_0	Pixel_1	Pixel_2	...	Pixel_3062	Pixel_3063	P
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	187	148	190	...	178	154	
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	25	14	23	...	91	43	
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	146	133	186	...	167	143	
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	27	16	31	...	77	22	
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	134	110	153	...	219	179	

5 rows × 3079 columns

In [121]:
```
df3['label']=df['dx'].apply(score_to_numeric)
df3.head(5)
```

Out[121]:

	lesion_id	image_id	dx	dx_type	age	sex	localization	Pixel_0	Pixel_1	Pixel_2	...	Pixel_3063	Pixel_3064	P
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	187	148	190	...	154	132	
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	25	14	23	...	43	26	
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	146	133	186	...	143	128	
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	27	16	31	...	22	16	
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	134	110	153	...	179	161	

5 rows × 3080 columns

In [ ]:
```
# saving the dataframe
df3.to_csv('df3_final.csv', header=True, index=False)
```

K - Nearest Neigbhor

Note : All analysis now onwards has been done on Google colab for fast processing with aid of Graphic Processing Unit(GPU).

In [0]:
```
from google.colab import drive
drive.mount('/content/drive') # Note all analysis now onwards has been done on Google colab for fast processing
                              # with aid of Graphic Processing Unit(GPU).
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
import numpy as np
import pandas as pd
import random
random.seed(3)
```

In [0]:
```
df3 = pd.read_csv("/content/drive/My Drive/image classification project/df3_final.csv",header=None)  #importing data from directory
df3.head(5)
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	187	148	190	191	153	194
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	25	14	23	66	40	56
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	146	133	186	157	145	198
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	27	16	31	70	55	86
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	134	110	153	171	142	188

5 rows × 3080 columns

In [0]:
```
df3.tail(5)
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10
10010	HAM_0002867	ISIC_0033084	akiec	histo	40.0	male	abdomen	196	177	192	195
10011	HAM_0002867	ISIC_0033550	akiec	histo	40.0	male	abdomen	3	7	5	14
10012	HAM_0002867	ISIC_0033536	akiec	histo	40.0	male	abdomen	130	125	132	137
10013	HAM_0000239	ISIC_0032854	akiec	histo	80.0	male	face	145	123	159	152
10014	HAM_0003521	ISIC_0032258	mel	histo	70.0	female	back	119	140	173	130

5 rows × 3080 columns

In [0]:
```
X = df3.iloc[:,7:3079].values
y = df3.iloc[:, 3079].values
```

In [0]:
```
from sklearn.model_selection import train_test_split
```

In [0]:
```
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.30)
```

In [0]:
```
Xtr
```

Out[0]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [161, 154, 189, ..., 149, 143, 175],
       [152, 147, 220, ..., 133, 139, 209],
       ...,
       [198, 175, 219, ..., 168, 158, 209],
       [223, 193, 234, ..., 194, 169, 207],
       [172, 162, 183, ..., 173, 164, 185]])
```

In [0]:
```
Xte
```

Out[0]:
```
array([[162, 162, 241, ..., 148, 153, 219],
       [157, 138, 229, ..., 136, 138, 212],
       [201, 181, 205, ..., 144, 144, 174],
       ...,
       [151, 137, 215, ..., 143, 127, 202],
       [169, 137, 224, ..., 163, 160, 224],
       [161, 171, 238, ..., 154, 168, 226]])
```

In [0]:
```
ytr
```

Out[0]:
```
array([5, 4, 5, ..., 2, 3, 5])
```

In [0]:
```
yte
```

Out[0]:
```
array([5, 5, 1, ..., 5, 5, 5])
```

In [0]:
```
Xtr_rows = Xtr.reshape(Xtr.shape[0], 32 * 32 * 3)
Xte_rows = Xte.reshape(Xte.shape[0], 32 * 32 * 3)
```

In [0]:
```
Xtr_rows
```

Out[0]:
```
array([[  0,   0,   0, ...,   0,   0,   0],
       [161, 154, 189, ..., 149, 143, 175],
       [152, 147, 220, ..., 133, 139, 209],
       ...,
       [198, 175, 219, ..., 168, 158, 209],
       [223, 193, 234, ..., 194, 169, 207],
       [172, 162, 183, ..., 173, 164, 185]])
```

In [0]:
```
class NearestNeighbor(object):
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

In [0]:
```
nn = NearestNeighbor() # create a Nearest Neighbor classifier class
nn.train(Xtr_rows, ytr) # train the classifier on the training images and labels
Yte_predict = nn.predict(Xte_rows) # predict labels on the test images
# and now print the classification accuracy, which is the average number
# of examples that are correctly predicted (i.e. label matches)
print('accuracy: %f' % ( np.mean(Yte_predict == yte) ))
```

```
accuracy: 0.673877
```

In [0]:
```
Accuracy=np.mean(Yte_predict == yte)
Accuracy
```

Out[0]:
```
0.6738768718801996
```

In [0]:
```
# Python script for confusion matrix creation.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

results = confusion_matrix(yte, Yte_predict)

print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(yte, Yte_predict))
print ('Report : ',classification_report(yte, Yte_predict))
```

```
Confusion Matrix :
[[  22   11   18    4    2   45    3]
 [   5   46   21    6    6   55    7]
 [   6   22   98    8   23  160    6]
 [   0    7    7    4    0   16    1]
 [   7    7   51    3   61  190    4]
 [   9   27   85    6   84 1789   37]
 [   0    2    3    1    2   23    5]]
Accuracy Score : 0.6738768718801996
Report :               precision    recall  f1-score   support

           0       0.45      0.21      0.29       105
           1       0.38      0.32      0.34       146
           2       0.35      0.30      0.32       323
           3       0.12      0.11      0.12        35
           4       0.34      0.19      0.24       323
           5       0.79      0.88      0.83      2037
           6       0.08      0.14      0.10        36

    accuracy                           0.67      3005
   macro avg       0.36      0.31      0.32      3005
weighted avg       0.64      0.67      0.65      3005
```

In [0]:
```
class NearestNeighbor1(object):
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of s
ize N """
    # the nearest neighbor classifier simply remembers all the trainin
g data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict lab
el for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L2 distance (sqaured sum of absolute value differenc
es)
      distances = np.sqrt(np.sum(np.square(self.Xtr - X[i,:]), axis =
1))
      min_index = np.argmin(distances) # get the index with smallest d
istance
      Ypred[i] = self.ytr[min_index] # predict the label of the neares
t example

    return Ypred
```

In [0]:
```
nn1 = NearestNeighbor1() # create a Nearest Neighbor classifier class
nn1.train(Xtr_rows, ytr) # train the classifier on the training images
and labels
Yte_predict1 = nn1.predict(Xte_rows) # predict labels on the test imag
es
# and now print the classification accuracy, which is the average numb
er
# of examples that are correctly predicted (i.e. label matches)
print('accuracy: %f' % ( np.mean(Yte_predict1 == yte) ))
```

```
accuracy: 0.684193
```

In [0]:
```
Accuracy1=np.mean(Yte_predict1 == yte)
Accuracy1
```

Out[0]:
```
0.6841930116472545
```

In [0]:
```
results1 = confusion_matrix(yte, Yte_predict1)

print ('Confusion Matrix :')
print(results1)

print ('Accuracy Score :',accuracy_score(yte, Yte_predict1))
print ('Report : ',classification_report(yte, Yte_predict1))
```

```
Confusion Matrix :
[[  21   13   19    5    4   40    3]
 [   7   36   28    6    2   59    8]
 [   5   20  118   11   22  144    3]
 [   2    3    5    6    0   19    0]
 [   8    7   46    2   69  189    2]
 [   5   18  102    6   77 1802   27]
 [   0    1    6    0    3   22    4]]
Accuracy Score : 0.6841930116472545
Report :               precision    recall  f1-score   support

           0       0.44      0.20      0.27       105
           1       0.37      0.25      0.30       146
           2       0.36      0.37      0.36       323
           3       0.17      0.17      0.17        35
           4       0.39      0.21      0.28       323
           5       0.79      0.88      0.84      2037
           6       0.09      0.11      0.10        36

    accuracy                           0.68      3005
   macro avg       0.37      0.31      0.33      3005
weighted avg       0.65      0.68      0.66      3005
```

Support Vector Machine(SVM)

In [0]:
```
# importing important libraries
import numpy as np
import pandas as pd
import random
random.seed(3)
```

In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au
th?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog
leusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&r
esponse_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fa
uth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20
https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20ht
tps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
df3 = pd.read_csv("/content/drive/My Drive/image classification projec
t/df3_final.csv",header=None) #importing data from directory
df3.head(5)
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	187	148	190	191	153	194
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	25	14	23	66	40	56
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	146	133	186	157	145	198
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	27	16	31	70	55	86
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	134	110	153	171	142	188

5 rows × 3080 columns

In [0]:
```
X = df3.iloc[:,7:3079].values
y = df3.iloc[:, 3079].values
```

In [0]:
```
from sklearn.model_selection import train_test_split
```

In [0]:
```
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.30)
#split data into training and testing group
```

In [0]:
```
from sklearn import svm
```

Linear kernel

In [0]:
```
classifier_linear = svm.SVC(kernel='linear')
#fit to the trainin data
classifier_linear.fit(Xtr,ytr)
```

Out[0]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef
0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kerne
l='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
    tol=0.001, verbose=False)
```

In [0]:
```
y_pred = classifier_linear.predict(Xte)
```

In [0]:
```
print('accuracy: %f' % ( np.mean(y_pred == yte) ))
# Accuracy of the model
```

```
accuracy: 0.613311
```

In [0]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [0]:
```
print(confusion_matrix(yte,y_pred))
```

```
[[  19   26   14    6    5   23    1]
 [  16   72   18    3   14   28    3]
 [   8   18  105    5   36  139    0]
 [   1    9    3    5    2   11    1]
 [  10   15   46    7   96  140    1]
 [  37   43  207    5  148 1610    8]
 [   0    6    2    1    7    6   19]]
```

In [0]:
```
print(classification_report(yte,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.21      0.20      0.21        94
           1       0.38      0.47      0.42       154
           2       0.27      0.34      0.30       311
           3       0.16      0.16      0.16        32
           4       0.31      0.30      0.31       315
           5       0.82      0.78      0.80      2058
           6       0.58      0.46      0.51        41

    accuracy                           0.64      3005
   macro avg       0.39      0.39      0.39      3005
weighted avg       0.66      0.64      0.65      3005
```

Polynomial Kernel

In [0]:
```
classifier_poly1 = svm.SVC(kernel='poly',degree=2)
#fit to the trainin data
classifier_poly1.fit(Xtr,ytr)
```

Out[0]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef
0=0.0,
    decision_function_shape='ovr', degree=2, gamma='scale', kerne
l='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
    tol=0.001, verbose=False)
```

In [0]:
```
y_pred_poly1 = classifier_poly1.predict(Xte)
```

In [0]:
```
print('accuracy: %f' % ( np.mean(y_pred_poly1 == yte) ))
print(confusion_matrix(yte,y_pred_poly1))
print(classification_report(yte,y_pred_poly1))
```

```
accuracy: 0.731780
[[  15   25   12    2    8   32    0]
 [  11   72   14    1    5   48    3]
 [   4   12  100    1   22  172    0]
 [   2    9    1    3    3   14    0]
 [   4    9   50    2   71  178    1]
 [   8   18   57    2   48 1920    5]
 [   1    6    0    0    5   11   18]]
              precision    recall  f1-score   support

           0       0.33      0.16      0.22        94
           1       0.48      0.47      0.47       154
           2       0.43      0.32      0.37       311
           3       0.27      0.09      0.14        32
           4       0.44      0.23      0.30       315
           5       0.81      0.93      0.87      2058
           6       0.67      0.44      0.53        41

    accuracy                           0.73      3005
   macro avg       0.49      0.38      0.41      3005
weighted avg       0.69      0.73      0.70      3005
```

Radial basis function kernel

In [0]:
```
classifier_rbf = svm.SVC(kernel='rbf')
#fit to the trainin data
classifier_rbf.fit(Xtr,ytr)
```

Out[0]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef
0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kerne
l='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
    tol=0.001, verbose=False)
```

In [0]:
```
y_pred_rbf = classifier_rbf.predict(Xte)
```

In [0]:
```
print('accuracy: %f' % ( np.mean(y_pred_rbf== yte) ))
print(confusion_matrix(yte,y_pred_rbf))
print(classification_report(yte,y_pred_rbf))
```

```
accuracy: 0.725125
[[   4   21   16    0    2   51    0]
 [   3   48   13    0    2   88    0]
 [   0    7  102    0   11  191    0]
 [   2    7    3    0    1   19    0]
 [   1    2   34    0   23  255    0]
 [   2    8   32    0   14 2002    0]
 [   0    5    0    0    0   36    0]]
              precision    recall  f1-score   support

           0       0.33      0.04      0.08        94
           1       0.49      0.31      0.38       154
           2       0.51      0.33      0.40       311
           3       0.00      0.00      0.00        32
           4       0.43      0.07      0.13       315
           5       0.76      0.97      0.85      2058
           6       0.00      0.00      0.00        41

    accuracy                           0.73      3005
   macro avg       0.36      0.25      0.26      3005
weighted avg       0.65      0.73      0.66      3005


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classificati
on.py:1272: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Sigmoid Kernel

In [0]:
```
classifier_sigmoid = svm.SVC(kernel='sigmoid')
#fit to the trainin data
classifier_sigmoid.fit(Xtr,ytr)
```

Out[0]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef
0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kerne
l='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
    tol=0.001, verbose=False)
```

In [0]:
```
y_pred_sigmoid = classifier_sigmoid.predict(Xte)
```

In [0]:
```
print('accuracy: %f' % ( np.mean(y_pred_sigmoid== yte) ))
print(confusion_matrix(yte,y_pred_sigmoid))
print(classification_report(yte,y_pred_sigmoid))
```

```
accuracy: 0.684859
[[   0    0    0    0    0   94    0]
 [   0    0    0    0    0  154    0]
 [   0    0    0    0    0  311    0]
 [   0    0    0    0    0   32    0]
 [   0    0    0    0    0  315    0]
 [   0    0    0    0    0 2058    0]
 [   0    0    0    0    0   41    0]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        94
           1       0.00      0.00      0.00       154
           2       0.00      0.00      0.00       311
           3       0.00      0.00      0.00        32
           4       0.00      0.00      0.00       315
           5       0.68      1.00      0.81      2058
           6       0.00      0.00      0.00        41

    accuracy                           0.68      3005
   macro avg       0.10      0.14      0.12      3005
weighted avg       0.47      0.68      0.56      3005


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classificati
on.py:1272: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Artificial Neural Network

In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au
th?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog
leusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&r
esponse_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fa
uth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20
https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20ht
tps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
#Installing important libraries
import numpy as np
import pandas as pd
```

In [0]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LeakyReLU,PReLU,ELU,Activation
from keras.layers import Dropout
```

```
ERROR: Could not find a version that satisfies the requirement Label
Powerset (from versions: none)
ERROR: No matching distribution found for LabelPowerset
```

In [0]:
```
!pip install --upgrade LabelPowerset
```

```
ERROR: Could not find a version that satisfies the requirement Label
Powerset (from versions: none)
ERROR: No matching distribution found for LabelPowerset
```

In [0]:
```
df3 = pd.read_csv("/content/drive/My Drive/image classification projec
t/df3_final.csv",header=None)
X = df3.iloc[:,7:3079].values
y = df3.iloc[:, 3079].values
#Normalizing the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
np.random.seed(3)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
e=0.30,random_state = 0)
```

Single-Layer Neural Network

In [0]:
```
# Initialising the Single Layer Neural Network
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(100 ,input_dim = X_train.shape[1]))
classifier.add(Activation('relu'))
classifier.add(Dropout(0.3))
classifier.add(Dense(100))
classifier.add(Activation('relu'))
classifier.add(Dropout(0.3))
classifier.add(Dense(7, activation = 'softmax'))
classifier.compile(optimizer = 'adam', loss = 'sparse_categorical_cros
sentropy', metrics = ['accuracy'])

model_history=classifier.fit(X_train, y_train,validation_split=0.20, b
atch_size = 128, nb_epoch = 50)
print(model_history.history.keys())
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: Use
rWarning: The `nb_epoch` argument in `fit` has been renamed `epoch
s`.
  del sys.path[0]

Train on 5608 samples, validate on 1402 samples
Epoch 1/50
5608/5608 [==============================] - 1s 167us/step - loss:
1.3444 - acc: 0.6004 - val_loss: 0.8975 - val_acc: 0.6726
Epoch 2/50
5608/5608 [==============================] - 0s 45us/step - loss:
1.0582 - acc: 0.6473 - val_loss: 0.8563 - val_acc: 0.6997
Epoch 3/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.9583 - acc: 0.6699 - val_loss: 0.8360 - val_acc: 0.6961
Epoch 4/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.9109 - acc: 0.6764 - val_loss: 0.8190 - val_acc: 0.7097
Epoch 5/50
```

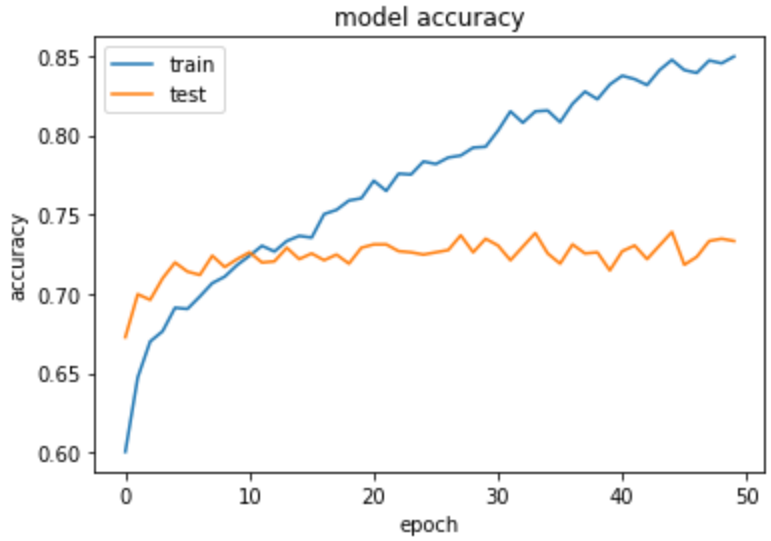
```
5608/5608 [==============================] - 0s 43us/step - loss:
0.8775 - acc: 0.6912 - val_loss: 0.7930 - val_acc: 0.7197
Epoch 6/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.8485 - acc: 0.6904 - val_loss: 0.8105 - val_acc: 0.7140
Epoch 7/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.8221 - acc: 0.6983 - val_loss: 0.7783 - val_acc: 0.7118
Epoch 8/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.8055 - acc: 0.7067 - val_loss: 0.7801 - val_acc: 0.7240
Epoch 9/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.7848 - acc: 0.7108 - val_loss: 0.7943 - val_acc: 0.7168
Epoch 10/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.7844 - acc: 0.7181 - val_loss: 0.7831 - val_acc: 0.7218
Epoch 11/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.7470 - acc: 0.7240 - val_loss: 0.7678 - val_acc: 0.7261
Epoch 12/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.7408 - acc: 0.7302 - val_loss: 0.7675 - val_acc: 0.7197
Epoch 13/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.7431 - acc: 0.7266 - val_loss: 0.7569 - val_acc: 0.7204
Epoch 14/50
5608/5608 [==============================] - 0s 41us/step - loss:
0.7146 - acc: 0.7332 - val_loss: 0.7672 - val_acc: 0.7290
Epoch 15/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.7054 - acc: 0.7364 - val_loss: 0.7506 - val_acc: 0.7218
Epoch 16/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.7056 - acc: 0.7354 - val_loss: 0.7667 - val_acc: 0.7254
Epoch 17/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.6806 - acc: 0.7502 - val_loss: 0.7576 - val_acc: 0.7211
Epoch 18/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.6646 - acc: 0.7529 - val_loss: 0.7375 - val_acc: 0.7247
Epoch 19/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.6542 - acc: 0.7587 - val_loss: 0.7559 - val_acc: 0.7190
Epoch 20/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.6469 - acc: 0.7602 - val_loss: 0.7516 - val_acc: 0.7290
Epoch 21/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.6350 - acc: 0.7712 - val_loss: 0.7577 - val_acc: 0.7311
Epoch 22/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.6280 - acc: 0.7648 - val_loss: 0.7558 - val_acc: 0.7311
Epoch 23/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.6215 - acc: 0.7757 - val_loss: 0.7427 - val_acc: 0.7268
```

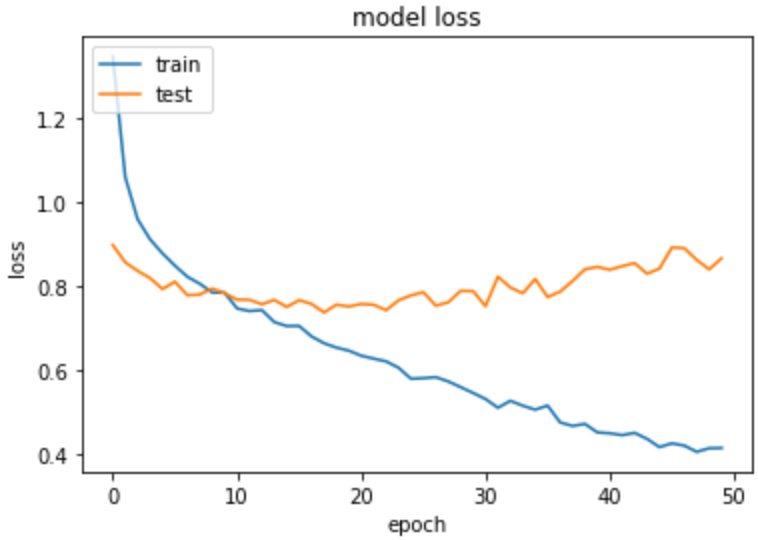
```
Epoch 24/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.6065 - acc: 0.7751 - val_loss: 0.7663 - val_acc: 0.7261
Epoch 25/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.5803 - acc: 0.7833 - val_loss: 0.7780 - val_acc: 0.7247
Epoch 26/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.5817 - acc: 0.7817 - val_loss: 0.7856 - val_acc: 0.7261
Epoch 27/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.5841 - acc: 0.7858 - val_loss: 0.7535 - val_acc: 0.7275
Epoch 28/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.5740 - acc: 0.7871 - val_loss: 0.7615 - val_acc: 0.7368
Epoch 29/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.5605 - acc: 0.7921 - val_loss: 0.7888 - val_acc: 0.7261
Epoch 30/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.5466 - acc: 0.7926 - val_loss: 0.7882 - val_acc: 0.7347
Epoch 31/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.5323 - acc: 0.8028 - val_loss: 0.7524 - val_acc: 0.7304
Epoch 32/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.5115 - acc: 0.8149 - val_loss: 0.8223 - val_acc: 0.7211
Epoch 33/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.5281 - acc: 0.8078 - val_loss: 0.7967 - val_acc: 0.7297
Epoch 34/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.5165 - acc: 0.8149 - val_loss: 0.7830 - val_acc: 0.7382
Epoch 35/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.5071 - acc: 0.8154 - val_loss: 0.8169 - val_acc: 0.7254
Epoch 36/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.5168 - acc: 0.8081 - val_loss: 0.7741 - val_acc: 0.7190
Epoch 37/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.4769 - acc: 0.8197 - val_loss: 0.7867 - val_acc: 0.7311
Epoch 38/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.4685 - acc: 0.8274 - val_loss: 0.8117 - val_acc: 0.7254
Epoch 39/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.4733 - acc: 0.8226 - val_loss: 0.8399 - val_acc: 0.7261
Epoch 40/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.4532 - acc: 0.8318 - val_loss: 0.8455 - val_acc: 0.7147
Epoch 41/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.4510 - acc: 0.8374 - val_loss: 0.8388 - val_acc: 0.7268
Epoch 42/50
5608/5608 [==============================] - 0s 43us/step - loss:
```

```
0.4466 - acc: 0.8352 - val_loss: 0.8472 - val_acc: 0.7304
Epoch 43/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.4521 - acc: 0.8315 - val_loss: 0.8546 - val_acc: 0.7218
Epoch 44/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.4379 - acc: 0.8409 - val_loss: 0.8294 - val_acc: 0.7304
Epoch 45/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.4183 - acc: 0.8474 - val_loss: 0.8414 - val_acc: 0.7389
Epoch 46/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.4272 - acc: 0.8409 - val_loss: 0.8914 - val_acc: 0.7183
Epoch 47/50
5608/5608 [==============================] - 0s 44us/step - loss:
0.4219 - acc: 0.8392 - val_loss: 0.8899 - val_acc: 0.7233
Epoch 48/50
5608/5608 [==============================] - 0s 43us/step - loss:
0.4071 - acc: 0.8470 - val_loss: 0.8621 - val_acc: 0.7332
Epoch 49/50
5608/5608 [==============================] - 0s 42us/step - loss:
0.4156 - acc: 0.8452 - val_loss: 0.8397 - val_acc: 0.7347
Epoch 50/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.4161 - acc: 0.8495 - val_loss: 0.8658 - val_acc: 0.7332
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

In [0]:

```
import matplotlib.pyplot as plt
# summarize history for accuracy
plt.plot(model_history.history['acc'])
plt.plot(model_history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [0]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
pred = list()
for i in range(len(y_pred)):
    pred.append(np.argmax(y_pred[i]))
#Converting one hot encoded test label to label
test = list()
for i in range(len(y_test)):
    test.append(y_test[i])
```

In [0]:
```
from sklearn.metrics import accuracy_score
a = accuracy_score(pred,test)
print('Accuracy is:', a*100)
```

```
Accuracy is: 72.31281198003327
```

In [0]:
```
# Making the Confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(test, pred)
cm
```

Out[0]:
```
array([[  25,   16,   19,    1,    4,   27,    1],
       [  15,   74,   16,    2,    6,   40,   11],
       [  12,   11,  134,    1,   45,  110,    2],
       [   3,   21,   10,    1,    1,    8,    0],
       [   7,    4,   49,    0,  100,  175,    3],
       [   8,   23,   97,    0,  104, 1777,    3],
       [   0,    5,    2,    0,    3,    9,   20]])
```

In [0]:
```
print(classification_report(test, pred))
```

```
              precision    recall  f1-score   support

           0       0.36      0.27      0.31        93
           1       0.48      0.45      0.47       164
           2       0.41      0.43      0.42       315
           3       0.20      0.02      0.04        44
           4       0.38      0.30      0.33       338
           5       0.83      0.88      0.85      2012
           6       0.50      0.51      0.51        39

    accuracy                           0.71      3005
   macro avg       0.45      0.41      0.42      3005
weighted avg       0.69      0.71      0.70      3005
```

Fully Connected Neural Network

In [0]:
```
classifier1 = Sequential()

# Adding the input layer and the second hidden layer
classifier1.add(Dense(100 ,input_dim = X_train.shape[1]))
classifier1.add(Activation('relu'))
classifier1.add(Dense(100))
classifier1.add(Activation('relu'))
classifier1.add(Dropout(0.3))
classifier1.add(Dense(100))
classifier1.add(Activation('relu'))
classifier1.add(Dropout(0.3))
classifier1.add(Dense(7, activation = 'softmax'))
classifier1.compile(optimizer = 'adam', loss = 'sparse_categorical_cro
ssentropy', metrics = ['accuracy'])
model1_history=classifier1.fit(X_train, y_train,validation_split=0.20,
batch_size = 128, nb_epoch = 50)
print(model1_history.history.keys())
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: Use
rWarning: The `nb_epoch` argument in `fit` has been renamed `epoch
s`.


Train on 5608 samples, validate on 1402 samples
Epoch 1/50
5608/5608 [==============================] - 1s 212us/step - loss:
1.1630 - acc: 0.6279 - val_loss: 0.9004 - val_acc: 0.6862
Epoch 2/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.9512 - acc: 0.6730 - val_loss: 0.8552 - val_acc: 0.6983
Epoch 3/50
5608/5608 [==============================] - 0s 49us/step - loss:
0.9025 - acc: 0.6808 - val_loss: 0.8331 - val_acc: 0.7033
Epoch 4/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.8773 - acc: 0.6912 - val_loss: 0.8554 - val_acc: 0.7104
Epoch 5/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.8596 - acc: 0.6945 - val_loss: 0.8139 - val_acc: 0.7147
Epoch 6/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.8096 - acc: 0.7115 - val_loss: 0.8241 - val_acc: 0.7118
Epoch 7/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.7965 - acc: 0.7143 - val_loss: 0.7855 - val_acc: 0.7147
Epoch 8/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.7701 - acc: 0.7238 - val_loss: 0.7971 - val_acc: 0.7175
Epoch 9/50
5608/5608 [==============================] - 0s 49us/step - loss:
0.7568 - acc: 0.7236 - val_loss: 0.7867 - val_acc: 0.7261
Epoch 10/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.7219 - acc: 0.7400 - val_loss: 0.8049 - val_acc: 0.7147
Epoch 11/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.6939 - acc: 0.7495 - val_loss: 0.7993 - val_acc: 0.7247
Epoch 12/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.6773 - acc: 0.7532 - val_loss: 0.7899 - val_acc: 0.7183
Epoch 13/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.6751 - acc: 0.7611 - val_loss: 0.8167 - val_acc: 0.7118
Epoch 14/50
5608/5608 [==============================] - 0s 49us/step - loss:
0.6372 - acc: 0.7636 - val_loss: 0.7902 - val_acc: 0.7268
Epoch 15/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.6407 - acc: 0.7625 - val_loss: 0.8039 - val_acc: 0.7204
Epoch 16/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.6274 - acc: 0.7668 - val_loss: 0.8306 - val_acc: 0.7111
Epoch 17/50
5608/5608 [==============================] - 0s 48us/step - loss:
```

```
0.5858 - acc: 0.7899 - val_loss: 0.8382 - val_acc: 0.7168
Epoch 18/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.5652 - acc: 0.7924 - val_loss: 0.8097 - val_acc: 0.7168
Epoch 19/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.5535 - acc: 0.8001 - val_loss: 0.8275 - val_acc: 0.7240
Epoch 20/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.5512 - acc: 0.7958 - val_loss: 0.8485 - val_acc: 0.7290
Epoch 21/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.5017 - acc: 0.8113 - val_loss: 0.8243 - val_acc: 0.7225
Epoch 22/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.4964 - acc: 0.8222 - val_loss: 0.8731 - val_acc: 0.7268
Epoch 23/50
5608/5608 [==============================] - 0s 50us/step - loss:
0.4744 - acc: 0.8211 - val_loss: 0.9204 - val_acc: 0.7211
Epoch 24/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.4675 - acc: 0.8306 - val_loss: 0.9205 - val_acc: 0.7261
Epoch 25/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.4396 - acc: 0.8395 - val_loss: 0.9316 - val_acc: 0.7161
Epoch 26/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.4135 - acc: 0.8524 - val_loss: 0.8918 - val_acc: 0.7297
Epoch 27/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.4371 - acc: 0.8388 - val_loss: 0.9403 - val_acc: 0.7240
Epoch 28/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.4546 - acc: 0.8340 - val_loss: 0.9244 - val_acc: 0.7211
Epoch 29/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.4093 - acc: 0.8493 - val_loss: 1.0173 - val_acc: 0.7154
Epoch 30/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.3857 - acc: 0.8611 - val_loss: 1.0086 - val_acc: 0.7354
Epoch 31/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.3581 - acc: 0.8693 - val_loss: 1.0383 - val_acc: 0.7168
Epoch 32/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.3595 - acc: 0.8682 - val_loss: 1.0154 - val_acc: 0.7261
Epoch 33/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.3407 - acc: 0.8759 - val_loss: 1.0217 - val_acc: 0.7211
Epoch 34/50
5608/5608 [==============================] - 0s 49us/step - loss:
0.3409 - acc: 0.8762 - val_loss: 1.0240 - val_acc: 0.7147
Epoch 35/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.2923 - acc: 0.8905 - val_loss: 1.0890 - val_acc: 0.7290
Epoch 36/50
```

```
5608/5608 [==============================] - 0s 46us/step - loss:
0.2942 - acc: 0.8907 - val_loss: 1.0665 - val_acc: 0.7389
Epoch 37/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.2921 - acc: 0.8900 - val_loss: 1.2216 - val_acc: 0.7233
Epoch 38/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.2932 - acc: 0.8928 - val_loss: 1.2760 - val_acc: 0.7161
Epoch 39/50
5608/5608 [==============================] - 0s 46us/step - loss:
0.2946 - acc: 0.8976 - val_loss: 1.1550 - val_acc: 0.7325
Epoch 40/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.2634 - acc: 0.9050 - val_loss: 1.1881 - val_acc: 0.7190
Epoch 41/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.2666 - acc: 0.9030 - val_loss: 1.1821 - val_acc: 0.7240
Epoch 42/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.2504 - acc: 0.9105 - val_loss: 1.2085 - val_acc: 0.7389
Epoch 43/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.2702 - acc: 0.9092 - val_loss: 1.2827 - val_acc: 0.7282
Epoch 44/50
5608/5608 [==============================] - 0s 45us/step - loss:
0.2384 - acc: 0.9142 - val_loss: 1.2328 - val_acc: 0.7368
Epoch 45/50
5608/5608 [==============================] - 0s 49us/step - loss:
0.2182 - acc: 0.9237 - val_loss: 1.3069 - val_acc: 0.7354
Epoch 46/50
5608/5608 [==============================] - 0s 49us/step - loss:
0.2110 - acc: 0.9256 - val_loss: 1.4059 - val_acc: 0.7068
Epoch 47/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.2374 - acc: 0.9187 - val_loss: 1.4289 - val_acc: 0.7168
Epoch 48/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.2184 - acc: 0.9206 - val_loss: 1.2973 - val_acc: 0.7061
Epoch 49/50
5608/5608 [==============================] - 0s 47us/step - loss:
0.2198 - acc: 0.9228 - val_loss: 1.3335 - val_acc: 0.7133
Epoch 50/50
5608/5608 [==============================] - 0s 48us/step - loss:
0.2158 - acc: 0.9228 - val_loss: 1.4246 - val_acc: 0.7211
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

In [0]:
```
# Predicting the Test set results
y_pred1 = classifier1.predict(X_test)
pred1 = list()
for i in range(len(y_pred1)):
    pred1.append(np.argmax(y_pred1[i]))
#Converting one hot encoded test label to label
test1 = list()
for i in range(len(y_test)):
    test1.append(y_test[i])
```

In [0]:
```
a1= accuracy_score(pred1,test1)
print('Accuracy is:', a1*100)
```

```
Accuracy is: 71.71381031613977
```

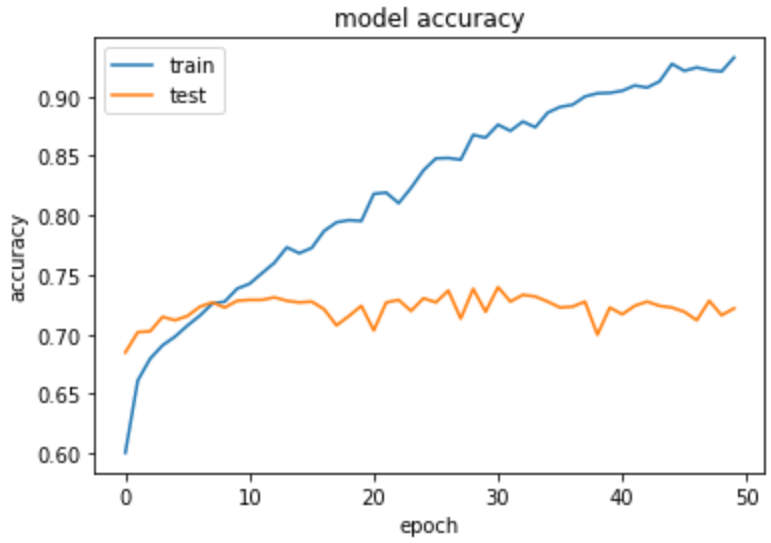
In [0]:
```
# Making the Confusion Matrix
cm1= confusion_matrix(test1, pred1)
cm1
```

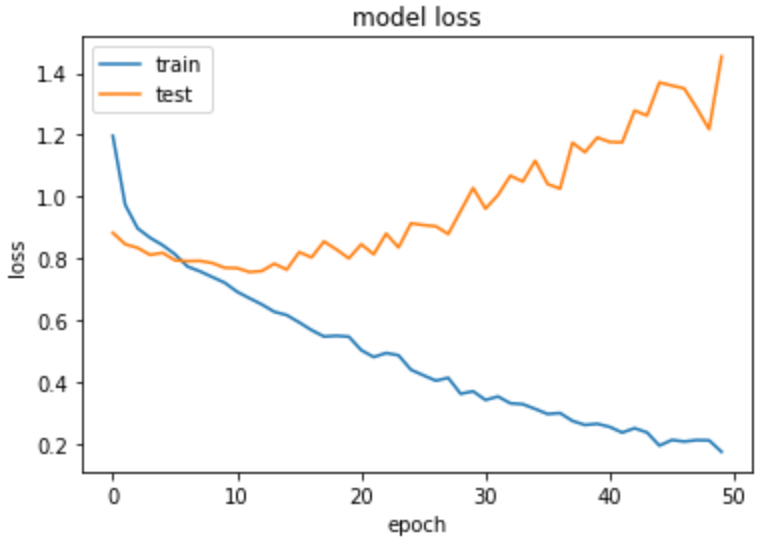
Out[0]:
```
array([[  28,   27,    6,    0,    6,   25,    1],
       [   7,   91,    8,    3,    3,   43,    9],
       [   9,   24,  106,    2,   37,  136,    1],
       [   2,   15,    7,    2,    1,   17,    0],
       [  10,   10,   31,    0,  100,  185,    2],
       [   5,   34,   63,    2,   88, 1814,    6],
       [   1,    5,    2,    1,    3,   13,   14]])
```

In [0]:
```
print(classification_report(test1, pred1))
```

```
              precision    recall  f1-score   support

           0       0.45      0.30      0.36        93
           1       0.44      0.55      0.49       164
           2       0.48      0.34      0.39       315
           3       0.20      0.05      0.07        44
           4       0.42      0.30      0.35       338
           5       0.81      0.90      0.85      2012
           6       0.42      0.36      0.39        39

    accuracy                           0.72      3005
   macro avg       0.46      0.40      0.42      3005
weighted avg       0.69      0.72      0.70      3005
```

In [0]:
```
# summarize history for accuracy
plt.plot(model1_history.history['acc'])
plt.plot(model1_history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss

plt.plot(model1_history.history['loss'])
plt.plot(model1_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





In [0]:
```
classifier2 = Sequential()


# Adding the input layer and the three hidden layer
classifier2.add(Dense(100 ,input_dim = X_train.shape[1]))
classifier2.add(Activation('relu'))
classifier2.add(Dropout(0.3))
classifier2.add(Dense(100))
classifier2.add(Activation('relu'))
classifier2.add(Dropout(0.3))
classifier2.add(Dense(100))
classifier2.add(Activation('relu'))
classifier2.add(Dropout(0.3))
classifier2.add(Dense(100))
classifier2.add(Activation('relu'))
classifier2.add(Dropout(0.3))
classifier2.add(Dense(7, activation = 'softmax'))
classifier2.compile(optimizer = 'adam', loss = 'sparse_categorical_cro
ssentropy', metrics = ['accuracy'])
model2_history=classifier2.fit(X_train, y_train,validation_split=0.33,
batch_size = 128, nb_epoch = 50)
print(model2_history.history.keys())
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:18: Use
rWarning: The `nb_epoch` argument in `fit` has been renamed `epoch
s`.

Train on 4696 samples, validate on 2314 samples
Epoch 1/50
4696/4696 [==============================] - 2s 329us/step - loss:
1.3385 - acc: 0.5841 - val_loss: 0.9820 - val_acc: 0.6707
Epoch 2/50
4696/4696 [==============================] - 0s 57us/step - loss:
1.0601 - acc: 0.6469 - val_loss: 0.9583 - val_acc: 0.6560
Epoch 3/50
4696/4696 [==============================] - 0s 59us/step - loss:
1.0166 - acc: 0.6508 - val_loss: 0.9232 - val_acc: 0.6733
Epoch 4/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.9859 - acc: 0.6625 - val_loss: 0.9037 - val_acc: 0.6819
Epoch 5/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.9776 - acc: 0.6597 - val_loss: 0.9157 - val_acc: 0.6876
Epoch 6/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.9449 - acc: 0.6650 - val_loss: 0.8776 - val_acc: 0.6884
Epoch 7/50
4696/4696 [==============================] - 0s 55us/step - loss:
0.9327 - acc: 0.6740 - val_loss: 0.9063 - val_acc: 0.6850
Epoch 8/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.9164 - acc: 0.6714 - val_loss: 0.8981 - val_acc: 0.7061
Epoch 9/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.8910 - acc: 0.6812 - val_loss: 0.8618 - val_acc: 0.6901
Epoch 10/50
4696/4696 [==============================] - 0s 57us/step - loss:
```

4696/4696 [==============================] - 0s 57us/step - loss:
0.8883 - acc: 0.6823 - val_loss: 0.8466 - val_acc: 0.7083
Epoch 11/50

4696/4696 [==============================] - 0s 55us/step - loss:
0.8823 - acc: 0.6868 - val_loss: 0.8463 - val_acc: 0.7005
Epoch 12/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.8546 - acc: 0.6899 - val_loss: 0.8458 - val_acc: 0.7048
Epoch 13/50
4696/4696 [==============================] - 0s 55us/step - loss:
0.8516 - acc: 0.6910 - val_loss: 0.8341 - val_acc: 0.7122
Epoch 14/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.8454 - acc: 0.6966 - val_loss: 0.8443 - val_acc: 0.7174
Epoch 15/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.8274 - acc: 0.6919 - val_loss: 0.8106 - val_acc: 0.7135
Epoch 16/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.8185 - acc: 0.7002 - val_loss: 0.8222 - val_acc: 0.7105
Epoch 17/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.8134 - acc: 0.7068 - val_loss: 0.8177 - val_acc: 0.7118
Epoch 18/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.8159 - acc: 0.7023 - val_loss: 0.8062 - val_acc: 0.7118
Epoch 19/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.8049 - acc: 0.7098 - val_loss: 0.8318 - val_acc: 0.7113
Epoch 20/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.8008 - acc: 0.7051 - val_loss: 0.8264 - val_acc: 0.7152
Epoch 21/50
4696/4696 [==============================] - 0s 55us/step - loss:
0.7879 - acc: 0.7119 - val_loss: 0.8167 - val_acc: 0.7131
Epoch 22/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.7685 - acc: 0.7198 - val_loss: 0.8097 - val_acc: 0.7092
Epoch 23/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.7778 - acc: 0.7189 - val_loss: 0.7976 - val_acc: 0.7191
Epoch 24/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.7594 - acc: 0.7221 - val_loss: 0.8054 - val_acc: 0.7165
Epoch 25/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.7496 - acc: 0.7215 - val_loss: 0.7903 - val_acc: 0.7169
Epoch 26/50
4696/4696 [==============================] - 0s 69us/step - loss:
0.7434 - acc: 0.7270 - val_loss: 0.7868 - val_acc: 0.7178
Epoch 27/50
4696/4696 [==============================] - 0s 64us/step - loss:
0.7591 - acc: 0.7230 - val_loss: 0.8039 - val_acc: 0.7277
Epoch 28/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.7341 - acc: 0.7302 - val_loss: 0.7983 - val_acc: 0.7226

```
Epoch 29/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.7276 - acc: 0.7353 - val_loss: 0.7919 - val_acc: 0.7217
Epoch 30/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.7098 - acc: 0.7306 - val_loss: 0.7986 - val_acc: 0.7226
Epoch 31/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.7146 - acc: 0.7359 - val_loss: 0.8032 - val_acc: 0.7187
Epoch 32/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.7026 - acc: 0.7389 - val_loss: 0.8112 - val_acc: 0.7148
Epoch 33/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.6875 - acc: 0.7474 - val_loss: 0.7853 - val_acc: 0.7247
Epoch 34/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.6951 - acc: 0.7428 - val_loss: 0.7923 - val_acc: 0.7213
Epoch 35/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.6787 - acc: 0.7487 - val_loss: 0.7741 - val_acc: 0.7234
Epoch 36/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.6862 - acc: 0.7464 - val_loss: 0.8087 - val_acc: 0.7200
Epoch 37/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.6852 - acc: 0.7513 - val_loss: 0.7935 - val_acc: 0.7200
Epoch 38/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.6520 - acc: 0.7598 - val_loss: 0.7808 - val_acc: 0.7273
Epoch 39/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.6480 - acc: 0.7611 - val_loss: 0.7902 - val_acc: 0.7247
Epoch 40/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.6511 - acc: 0.7587 - val_loss: 0.8000 - val_acc: 0.7213
Epoch 41/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.6397 - acc: 0.7660 - val_loss: 0.7894 - val_acc: 0.7282
Epoch 42/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.6499 - acc: 0.7666 - val_loss: 0.7727 - val_acc: 0.7282
Epoch 43/50
4696/4696 [==============================] - 0s 55us/step - loss:
0.6362 - acc: 0.7570 - val_loss: 0.7949 - val_acc: 0.7303
Epoch 44/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.6210 - acc: 0.7666 - val_loss: 0.8119 - val_acc: 0.7174
Epoch 45/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.6259 - acc: 0.7609 - val_loss: 0.7901 - val_acc: 0.7269
Epoch 46/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.6105 - acc: 0.7798 - val_loss: 0.7837 - val_acc: 0.7217
Epoch 47/50
4696/4696 [==============================] - 0s 57us/step - loss:
```

```
0.6164 - acc: 0.7777 - val_loss: 0.8037 - val_acc: 0.7264
Epoch 48/50
4696/4696 [==============================] - 0s 56us/step - loss:
0.5919 - acc: 0.7866 - val_loss: 0.8070 - val_acc: 0.7269
Epoch 49/50
4696/4696 [==============================] - 0s 57us/step - loss:
0.5926 - acc: 0.7792 - val_loss: 0.8001 - val_acc: 0.7221
Epoch 50/50
4696/4696 [==============================] - 0s 55us/step - loss:
0.5769 - acc: 0.7868 - val_loss: 0.8034 - val_acc: 0.7303
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

In [0]:
```
# Predicting the Test set results
y_pred2 = classifier2.predict(X_test)
pred2 = list()
for i in range(len(y_pred2)):
    pred2.append(np.argmax(y_pred2[i]))
#Converting one hot encoded test label to label
test2 = list()
for i in range(len(y_test)):
    test2.append(y_test[i])
```

In [0]:
```
a2 = accuracy_score(pred2,test2)
print('Accuracy is:', a2*100)
```

```
Accuracy is: 71.48086522462562
```

In [0]:
```
# Making the Confusion Matrix
cm2= confusion_matrix(test2, pred2)
cm2
```

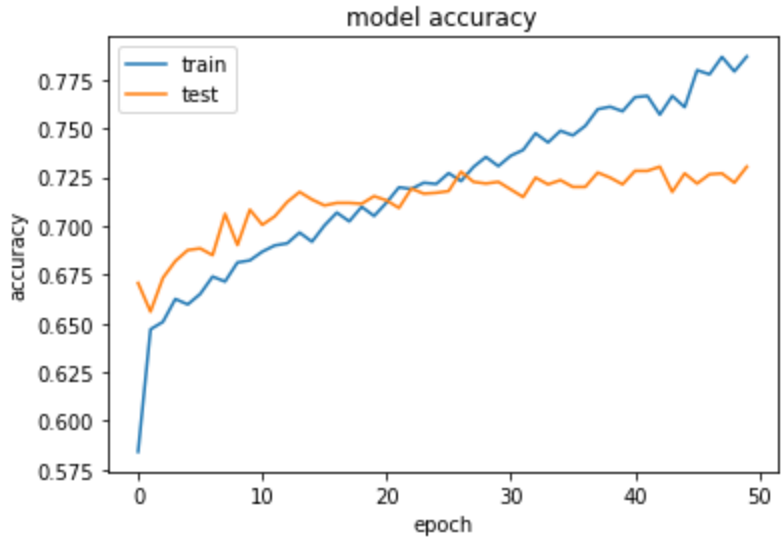
Out[0]:
```
array([[  21,   27,   20,    0,    1,   24,    0],
       [  11,   95,   22,    0,    0,   31,    5],
       [   6,   27,  131,    0,    9,  141,    1],
       [   3,   18,   13,    0,    0,   10,    0],
       [   5,   12,   66,    0,   28,  227,    0],
       [   3,   40,   84,    0,   14, 1869,    2],
       [   2,    8,    1,    0,    0,   24,    4]])
```

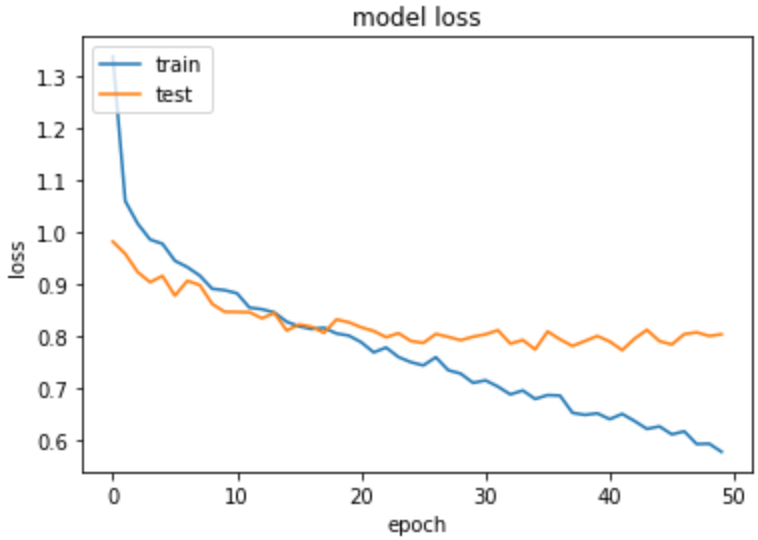
In [0]:
```
print(classification_report(test2, pred2))
```

```
              precision    recall  f1-score   support

           0       0.41      0.23      0.29        93
           1       0.42      0.58      0.49       164
           2       0.39      0.42      0.40       315
           3       0.00      0.00      0.00        44
           4       0.54      0.08      0.14       338
           5       0.80      0.93      0.86      2012
           6       0.33      0.10      0.16        39

    accuracy                           0.71      3005
   macro avg       0.41      0.33      0.33      3005
weighted avg       0.68      0.71      0.67      3005
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classificati
on.py:1272: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [0]:
```
# summarize history for accuracy
plt.plot(model2_history.history['acc'])
plt.plot(model2_history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(model2_history.history['loss'])
plt.plot(model2_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
accuracy
epoch
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
0
10
20
30
40
50


model loss
train
test
loss
epoch
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0
10
20
30
40
50

In [0]:
```
classifier3 = Sequential()

# Adding the input layer and the three hidden layer
classifier3.add(Dense(100 ,input_dim = X_train.shape[1]))
classifier3.add(Activation('relu'))
classifier3.add(Dropout(0.3))
classifier3.add(Dense(100))
classifier3.add(Activation('relu'))
classifier3.add(Dropout(0.3))
classifier3.add(Dense(100))
classifier3.add(Activation('relu'))
classifier3.add(Dropout(0.3))
classifier3.add(Dense(100))
classifier3.add(Activation('relu'))
classifier3.add(Dropout(0.3))
classifier3.add(Dense(100))
classifier3.add(Activation('relu'))
classifier3.add(Dropout(0.3))
classifier3.add(Dense(7, activation = 'softmax'))
classifier3.compile(optimizer = 'adam', loss = 'sparse_categorical_cro
ssentropy', metrics = ['accuracy'])

model3_history=classifier3.fit(X_train, y_train,validation_split=0.33,
batch_size = 128, nb_epoch = 50)
print(model3_history.history.keys())
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:21: Use
rWarning: The `nb_epoch` argument in `fit` has been renamed `epoch
s`.

Train on 4696 samples, validate on 2314 samples
Epoch 1/50
4696/4696 [==============================] - 2s 361us/step - loss:
1.2259 - acc: 0.6152 - val_loss: 0.9811 - val_acc: 0.6716
Epoch 2/50
4696/4696 [==============================] - 0s 60us/step - loss:
1.0467 - acc: 0.6491 - val_loss: 0.9677 - val_acc: 0.6711
Epoch 3/50
4696/4696 [==============================] - 0s 61us/step - loss:
1.0026 - acc: 0.6593 - val_loss: 0.9631 - val_acc: 0.6724
Epoch 4/50
4696/4696 [==============================] - 0s 65us/step - loss:
0.9913 - acc: 0.6589 - val_loss: 0.9669 - val_acc: 0.6780
Epoch 5/50
4696/4696 [==============================] - 0s 77us/step - loss:
0.9756 - acc: 0.6603 - val_loss: 0.9404 - val_acc: 0.6733
Epoch 6/50
4696/4696 [==============================] - 0s 66us/step - loss:
0.9533 - acc: 0.6650 - val_loss: 0.9263 - val_acc: 0.6832
Epoch 7/50
4696/4696 [==============================] - 0s 66us/step - loss:
0.9365 - acc: 0.6657 - val_loss: 0.9501 - val_acc: 0.6880
Epoch 8/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.9331 - acc: 0.6727 - val_loss: 0.9345 - val_acc: 0.6871
Epoch 9/50
4696/4696 [==============================] - 0s 60us/step - loss:
```

```
0.9145 - acc: 0.6697 - val_loss: 0.9224 - val_acc: 0.6940
Epoch 10/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.9056 - acc: 0.6655 - val_loss: 0.8952 - val_acc: 0.6923
Epoch 11/50
4696/4696 [==============================] - 0s 64us/step - loss:
0.8991 - acc: 0.6738 - val_loss: 0.9307 - val_acc: 0.6979
Epoch 12/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.8787 - acc: 0.6789 - val_loss: 0.8843 - val_acc: 0.7018
Epoch 13/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.8945 - acc: 0.6808 - val_loss: 0.9241 - val_acc: 0.7014
Epoch 14/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.8591 - acc: 0.6851 - val_loss: 0.8768 - val_acc: 0.7087
Epoch 15/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.8569 - acc: 0.6844 - val_loss: 0.8635 - val_acc: 0.7113
Epoch 16/50
4696/4696 [==============================] - 0s 59us/step - loss:

0.8432 - acc: 0.6910 - val_loss: 0.8621 - val_acc: 0.7057
Epoch 17/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.8348 - acc: 0.6934 - val_loss: 0.8614 - val_acc: 0.7057
Epoch 18/50
4696/4696 [==============================] - 0s 66us/step - loss:
0.8446 - acc: 0.6966 - val_loss: 0.9033 - val_acc: 0.7031
Epoch 19/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.8247 - acc: 0.6995 - val_loss: 0.8392 - val_acc: 0.7061
Epoch 20/50
4696/4696 [==============================] - 0s 64us/step - loss:
0.8260 - acc: 0.7070 - val_loss: 0.8547 - val_acc: 0.7074
Epoch 21/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.8266 - acc: 0.6993 - val_loss: 0.8589 - val_acc: 0.7113
Epoch 22/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.8052 - acc: 0.7072 - val_loss: 0.8743 - val_acc: 0.7139
Epoch 23/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.8017 - acc: 0.7012 - val_loss: 0.8605 - val_acc: 0.7070
Epoch 24/50
4696/4696 [==============================] - 0s 65us/step - loss:
0.7987 - acc: 0.7036 - val_loss: 0.8622 - val_acc: 0.7165
Epoch 25/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.7890 - acc: 0.7053 - val_loss: 0.8424 - val_acc: 0.7204
Epoch 26/50
4696/4696 [==============================] - 0s 62us/step - loss:
0.7732 - acc: 0.7117 - val_loss: 0.8353 - val_acc: 0.7165
Epoch 27/50
4696/4696 [==============================] - 0s 62us/step - loss:
0.7843 - acc: 0.7091 - val_loss: 0.8334 - val_acc: 0.7195
Epoch 28/50
```

```
Epoch 28/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.7707 - acc: 0.7117 - val_loss: 0.8137 - val_acc: 0.7178
Epoch 29/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.7661 - acc: 0.7164 - val_loss: 0.8303 - val_acc: 0.7096
Epoch 30/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.7691 - acc: 0.7161 - val_loss: 0.8355 - val_acc: 0.7204
Epoch 31/50
4696/4696 [==============================] - 0s 65us/step - loss:
0.7475 - acc: 0.7238 - val_loss: 0.8031 - val_acc: 0.7260
Epoch 32/50
4696/4696 [==============================] - 0s 59us/step - loss:
0.7540 - acc: 0.7198 - val_loss: 0.8506 - val_acc: 0.7048
Epoch 33/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.7327 - acc: 0.7306 - val_loss: 0.8310 - val_acc: 0.7165
Epoch 34/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.7343 - acc: 0.7219 - val_loss: 0.8367 - val_acc: 0.7122
Epoch 35/50
4696/4696 [==============================] - 0s 63us/step - loss:
0.7312 - acc: 0.7334 - val_loss: 0.8246 - val_acc: 0.7234
Epoch 36/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.7191 - acc: 0.7347 - val_loss: 0.8186 - val_acc: 0.7325
Epoch 37/50
4696/4696 [==============================] - 0s 62us/step - loss:
0.7208 - acc: 0.7325 - val_loss: 0.8244 - val_acc: 0.7178
Epoch 38/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.7154 - acc: 0.7379 - val_loss: 0.8075 - val_acc: 0.7208
Epoch 39/50
4696/4696 [==============================] - 0s 63us/step - loss:
0.6991 - acc: 0.7428 - val_loss: 0.8306 - val_acc: 0.7169
Epoch 40/50
4696/4696 [==============================] - 0s 62us/step - loss:
0.7108 - acc: 0.7381 - val_loss: 0.8422 - val_acc: 0.7299
Epoch 41/50
4696/4696 [==============================] - 0s 63us/step - loss:
0.7110 - acc: 0.7323 - val_loss: 0.8205 - val_acc: 0.7122
Epoch 42/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.6999 - acc: 0.7319 - val_loss: 0.8262 - val_acc: 0.7221
Epoch 43/50
4696/4696 [==============================] - 0s 62us/step - loss:
0.6671 - acc: 0.7543 - val_loss: 0.7978 - val_acc: 0.7204
Epoch 44/50
4696/4696 [==============================] - 0s 63us/step - loss:
0.6718 - acc: 0.7453 - val_loss: 0.8344 - val_acc: 0.7208
Epoch 45/50
4696/4696 [==============================] - 0s 60us/step - loss:
0.6717 - acc: 0.7477 - val_loss: 0.7988 - val_acc: 0.7273
Epoch 46/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.6596 - acc: 0.7543 - val_loss: 0.8037 - val_acc: 0.7290
```

```
0.6596 - acc: 0.7545 - val_loss: 0.8057 - val_acc: 0.7290
Epoch 47/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.6663 - acc: 0.7523 - val_loss: 0.8173 - val_acc: 0.7234
Epoch 48/50
4696/4696 [==============================] - 0s 64us/step - loss:
0.6519 - acc: 0.7474 - val_loss: 0.7846 - val_acc: 0.7252
Epoch 49/50
4696/4696 [==============================] - 0s 58us/step - loss:
0.6497 - acc: 0.7566 - val_loss: 0.8213 - val_acc: 0.7161
Epoch 50/50
4696/4696 [==============================] - 0s 61us/step - loss:
0.6418 - acc: 0.7621 - val_loss: 0.7987 - val_acc: 0.7165
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

In [0]:
```
# Predicting the Test set results
y_pred3 = classifier2.predict(X_test)
pred3 = list()
for i in range(len(y_pred3)):
    pred3.append(np.argmax(y_pred3[i]))
#Converting one hot encoded test label to label
test3 = list()
for i in range(len(y_test)):
    test3.append(y_test[i])
```

In [0]:
```
a3 = accuracy_score(pred3,test3)
print('Accuracy is:', a3*100)
```

```
Accuracy is: 71.48086522462562
```

In [0]:
```
# Making the Confusion Matrix
cm3= confusion_matrix(test3, pred3)
cm3
```

Out[0]:
```
array([[  21,   27,   20,    0,    1,   24,    0],
       [  11,   95,   22,    0,    0,   31,    5],
       [   6,   27,  131,    0,    9,  141,    1],
       [   3,   18,   13,    0,    0,   10,    0],
       [   5,   12,   66,    0,   28,  227,    0],
       [   3,   40,   84,    0,   14, 1869,    2],
       [   2,    8,    1,    0,    0,   24,    4]])
```

In [0]:
```
print(classification_report(test3, pred3))
```

```
              precision    recall  f1-score   support

           0       0.41      0.23      0.29        93
           1       0.42      0.58      0.49       164
           2       0.39      0.42      0.40       315
           3       0.00      0.00      0.00        44
           4       0.54      0.08      0.14       338
           5       0.80      0.93      0.86      2012
           6       0.33      0.10      0.16        39

    accuracy                           0.71      3005
   macro avg       0.41      0.33      0.33      3005
weighted avg       0.68      0.71      0.67      3005


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classificati
on.py:1272: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [0]:

Convolutional Neural Network

In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au
th?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog
leusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&r
esponse_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fa
uth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20
https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20ht
tps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
import pandas as pd
import numpy as np
```

In [0]:
```
import matplotlib.pyplot as plt
```

In [0]:
```
df3 = pd.read_csv("/content/drive/My Drive/image classification projec
t/df3_final.csv",header=None) #importing data from directory
X = df3.iloc[:,7:3079].values
y = df3.iloc[:, 3079].values
np.random.seed(3)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
e=0.20,random_state = 0)
```

In [0]:
```
X_train=X_train/255.0
X_test=X_test/255.0
X_train=X_train.reshape((X_train.shape[0]),32,32,3)                #res
izeing of the data
X_test=X_test.reshape((X_test.shape[0]),32,32,3)
```

In [0]:
```
X_test.shape
```

Out[0]:
```
(2003, 32, 32, 3)
```

In [0]:
```
#importing all important libraries
import keras
from keras.utils.np_utils import to_categorical # used for converting labels to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
import itertools
from keras.layers.normalization import BatchNormalization
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

from keras.optimizers import Adam
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade (https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info (https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

In [ ]:
```
# Encode categorical features as a one-hot numeric array
```

In [0]:
```
from keras.utils import to_categorical
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
```

In [0]:
```
input_shape = (32,32, 3)
num_classes = 7

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding = 'Same',input_shape=input_shape))

model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))



model.add(Conv2D(128, (3, 3), activation='relu',padding = 'Same'))

model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_28 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 128)       36992
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 8, 8, 128)         0
_________________________________________________________________
dropout_29 (Dropout)         (None, 8, 8, 128)         0
_________________________________________________________________
flatten_10 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_19 (Dense)             (None, 256)               2097408
_________________________________________________________________
dropout_30 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_20 (Dense)             (None, 7)                 1799
=================================================================
Total params: 2,137,095
Trainable params: 2,137,095
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
model.compile(optimizer = 'adam' , loss = "categorical_crossentropy",
metrics=["accuracy"])

epochs = 50
batch_size = 10
history = model.fit(X_train,y_train_one_hot, batch_size=batch_size,
                              epochs = epochs, validation_split=0.1)
```

```
Train on 7210 samples, validate on 802 samples
Epoch 1/50
7210/7210 [==============================] - 5s 672us/step - loss:
0.9964 - acc: 0.6660 - val_loss: 0.9458 - val_acc: 0.6820
Epoch 2/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.9078 - acc: 0.6730 - val_loss: 0.8989 - val_acc: 0.6858
Epoch 3/50
7210/7210 [==============================] - 4s 512us/step - loss:
0.8776 - acc: 0.6843 - val_loss: 0.7844 - val_acc: 0.7045
Epoch 4/50
7210/7210 [==============================] - 4s 507us/step - loss:
0.8367 - acc: 0.6946 - val_loss: 0.7701 - val_acc: 0.7195
Epoch 5/50
7210/7210 [==============================] - 4s 510us/step - loss:
0.7954 - acc: 0.7026 - val_loss: 0.7554 - val_acc: 0.7232
Epoch 6/50
7210/7210 [==============================] - 4s 511us/step - loss:
```

```
0.7691 - acc: 0.7100 - val_loss: 0.7213 - val_acc: 0.7257
Epoch 7/50
7210/7210 [==============================] - 4s 508us/step - loss:
0.7519 - acc: 0.7189 - val_loss: 0.7262 - val_acc: 0.7382
Epoch 8/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.7389 - acc: 0.7225 - val_loss: 0.7263 - val_acc: 0.7406
Epoch 9/50
7210/7210 [==============================] - 4s 504us/step - loss:
0.7153 - acc: 0.7287 - val_loss: 0.6772 - val_acc: 0.7431
Epoch 10/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.6989 - acc: 0.7408 - val_loss: 0.6917 - val_acc: 0.7531
Epoch 11/50
7210/7210 [==============================] - 4s 507us/step - loss:
0.6945 - acc: 0.7398 - val_loss: 0.7364 - val_acc: 0.7544
Epoch 12/50
7210/7210 [==============================] - 4s 504us/step - loss:
0.6765 - acc: 0.7481 - val_loss: 0.6676 - val_acc: 0.7581
Epoch 13/50
7210/7210 [==============================] - 4s 511us/step - loss:
0.6621 - acc: 0.7452 - val_loss: 0.6709 - val_acc: 0.7681
Epoch 14/50
7210/7210 [==============================] - 4s 507us/step - loss:
0.6554 - acc: 0.7552 - val_loss: 0.7361 - val_acc: 0.7382
Epoch 15/50
7210/7210 [==============================] - 4s 502us/step - loss:
0.6437 - acc: 0.7614 - val_loss: 0.6756 - val_acc: 0.7594
Epoch 16/50
7210/7210 [==============================] - 4s 512us/step - loss:
0.6292 - acc: 0.7580 - val_loss: 0.6855 - val_acc: 0.7494
Epoch 17/50
7210/7210 [==============================] - 4s 515us/step - loss:
0.6176 - acc: 0.7739 - val_loss: 0.6757 - val_acc: 0.7469
Epoch 18/50
7210/7210 [==============================] - 4s 508us/step - loss:
0.5986 - acc: 0.7731 - val_loss: 0.7106 - val_acc: 0.7631
Epoch 19/50
7210/7210 [==============================] - 4s 508us/step - loss:
0.6025 - acc: 0.7786 - val_loss: 0.6677 - val_acc: 0.7718
Epoch 20/50
7210/7210 [==============================] - 4s 502us/step - loss:
0.5911 - acc: 0.7796 - val_loss: 0.6985 - val_acc: 0.7606
Epoch 21/50
7210/7210 [==============================] - 4s 512us/step - loss:
0.5772 - acc: 0.7813 - val_loss: 0.6592 - val_acc: 0.7569
Epoch 22/50
7210/7210 [==============================] - 4s 526us/step - loss:
0.5590 - acc: 0.7836 - val_loss: 0.6514 - val_acc: 0.7631
Epoch 23/50
7210/7210 [==============================] - 4s 511us/step - loss:
0.5650 - acc: 0.7904 - val_loss: 0.6657 - val_acc: 0.7643
Epoch 24/50
7210/7210 [==============================] - 4s 508us/step - loss:
0.5339 - acc: 0.7974 - val_loss: 0.6758 - val_acc: 0.7756
Epoch 25/50
```

```
7210/7210 [==============================] - 4s 513us/step - loss:
0.5333 - acc: 0.7939 - val_loss: 0.6799 - val_acc: 0.7830
Epoch 26/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.5249 - acc: 0.7960 - val_loss: 0.6740 - val_acc: 0.7618
Epoch 27/50
7210/7210 [==============================] - 4s 502us/step - loss:
0.5195 - acc: 0.8039 - val_loss: 0.7030 - val_acc: 0.7656
Epoch 28/50
7210/7210 [==============================] - 4s 512us/step - loss:
0.5059 - acc: 0.8097 - val_loss: 0.6850 - val_acc: 0.7544
Epoch 29/50
7210/7210 [==============================] - 4s 500us/step - loss:
0.5024 - acc: 0.8024 - val_loss: 0.6987 - val_acc: 0.7581
Epoch 30/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.4862 - acc: 0.8144 - val_loss: 0.7131 - val_acc: 0.7668
Epoch 31/50
7210/7210 [==============================] - 4s 505us/step - loss:
0.4817 - acc: 0.8178 - val_loss: 0.7249 - val_acc: 0.7519
Epoch 32/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.4760 - acc: 0.8179 - val_loss: 0.6907 - val_acc: 0.7743
Epoch 33/50
7210/7210 [==============================] - 4s 514us/step - loss:
0.4528 - acc: 0.8252 - val_loss: 0.6877 - val_acc: 0.7706
Epoch 34/50
7210/7210 [==============================] - 4s 509us/step - loss:
0.4597 - acc: 0.8252 - val_loss: 0.7114 - val_acc: 0.7656
Epoch 35/50
7210/7210 [==============================] - 4s 521us/step - loss:
0.4485 - acc: 0.8302 - val_loss: 0.6720 - val_acc: 0.7718
Epoch 36/50
7210/7210 [==============================] - 4s 520us/step - loss:
0.4392 - acc: 0.8318 - val_loss: 0.7784 - val_acc: 0.7581
Epoch 37/50
7210/7210 [==============================] - 4s 502us/step - loss:
0.4457 - acc: 0.8344 - val_loss: 0.6899 - val_acc: 0.7656
Epoch 38/50
7210/7210 [==============================] - 4s 504us/step - loss:
0.4266 - acc: 0.8395 - val_loss: 0.7121 - val_acc: 0.7656
Epoch 39/50
7210/7210 [==============================] - 4s 511us/step - loss:
0.4067 - acc: 0.8442 - val_loss: 0.8055 - val_acc: 0.7406
Epoch 40/50
7210/7210 [==============================] - 4s 506us/step - loss:
0.4166 - acc: 0.8405 - val_loss: 0.7133 - val_acc: 0.7618
Epoch 41/50
7210/7210 [==============================] - 4s 501us/step - loss:
0.4139 - acc: 0.8413 - val_loss: 0.7072 - val_acc: 0.7706
Epoch 42/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.4124 - acc: 0.8402 - val_loss: 0.7207 - val_acc: 0.7731
Epoch 43/50
7210/7210 [==============================] - 4s 510us/step - loss:
0.4011 - acc: 0.8469 - val_loss: 0.7575 - val_acc: 0.7656
```

```
Epoch 44/50
7210/7210 [==============================] - 4s 510us/step - loss:
0.3987 - acc: 0.8479 - val_loss: 0.7013 - val_acc: 0.7656
Epoch 45/50
7210/7210 [==============================] - 4s 503us/step - loss:
0.3824 - acc: 0.8542 - val_loss: 0.6898 - val_acc: 0.7631
Epoch 46/50
7210/7210 [==============================] - 4s 502us/step - loss:
0.3890 - acc: 0.8533 - val_loss: 0.8221 - val_acc: 0.7494
Epoch 47/50
7210/7210 [==============================] - 4s 507us/step - loss:
0.3726 - acc: 0.8578 - val_loss: 0.7394 - val_acc: 0.7643
Epoch 48/50
7210/7210 [==============================] - 4s 509us/step - loss:
0.3767 - acc: 0.8556 - val_loss: 0.7697 - val_acc: 0.7643
Epoch 49/50
7210/7210 [==============================] - 4s 504us/step - loss:
0.3677 - acc: 0.8578 - val_loss: 0.7900 - val_acc: 0.7693
Epoch 50/50
7210/7210 [==============================] - 4s 504us/step - loss:
0.3576 - acc: 0.8662 - val_loss: 0.7340 - val_acc: 0.7656
```

In [0]:
```
model.evaluate(X_test, y_test_one_hot)[1]
```

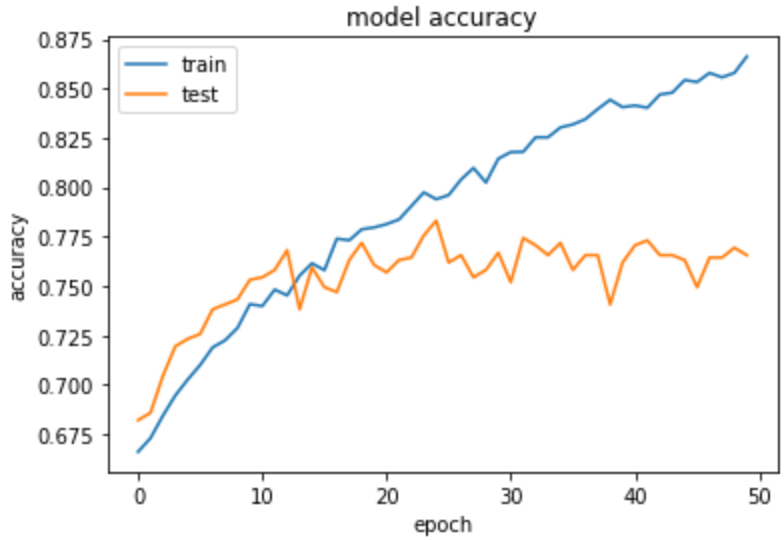
```
2003/2003 [==============================] - 0s 73us/step
```

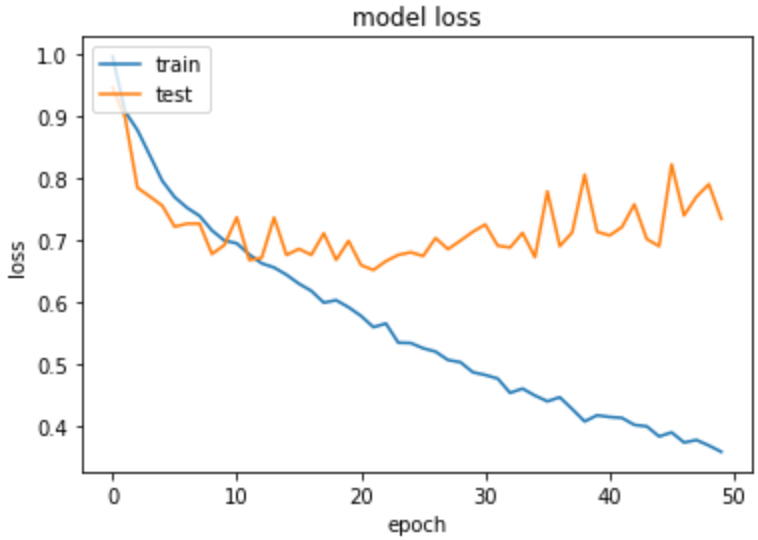
Out[0]:
```
0.7553669494268481
```

In [0]:
```
model.save_weights("model.h5") #savinf Model's weight into model.h5
```

Plots of Training and validation accuracy and loss

In [0]:
```
# Summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





Confusion Matrix and Classification reports

In [0]:
```
#Prediction of Testing image
preds=np.round(model.predict(X_test),0)
```

In [0]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

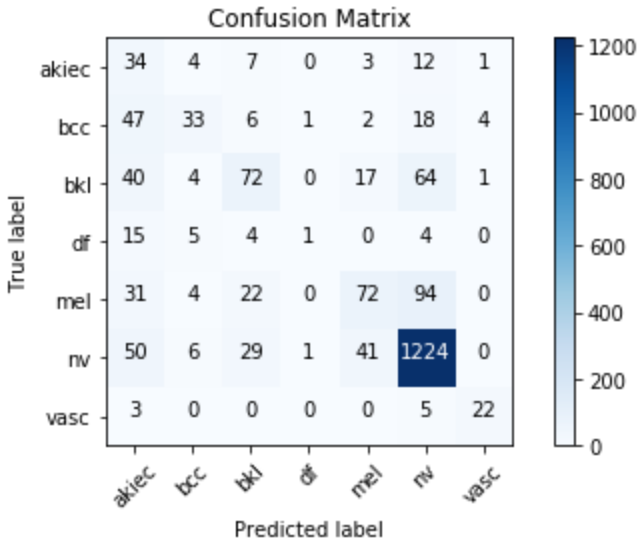
In [0]:
```
cm = confusion_matrix(y_test, preds.argmax(axis=1))
```

In [0]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shap
e[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [0]:
```
cm_plot_labels = ['akiec', 'bcc', 'bkl', 'df', 'mel','nv', 'vasc']
```

In [0]:
```
plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[  34    4    7    0    3   12    1]
 [  47   33    6    1    2   18    4]
 [  40    4   72    0   17   64    1]
 [  15    5    4    1    0    4    0]
 [  31    4   22    0   72   94    0]
 [  50    6   29    1   41 1224    0]
 [   3    0    0    0    0    5   22]]
```



In [0]:
```
# Generate a classification report
report = classification_report(y_test_one_hot, preds, target_names=c
m_plot_labels)
print(report)
```

```
              precision    recall  f1-score   support

       akiec       0.52      0.18      0.27        61
         bcc       0.59      0.30      0.40       111
         bkl       0.51      0.36      0.43       198
          df       0.33      0.03      0.06        29
         mel       0.53      0.32      0.40       223
          nv       0.86      0.91      0.88      1351
        vasc       0.79      0.73      0.76        30

   micro avg       0.80      0.72      0.75      2003
   macro avg       0.59      0.41      0.46      2003
weighted avg       0.76      0.72      0.72      2003
 samples avg       0.72      0.72      0.72      2003


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classificati
on.py:1272: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in samples with no predicted labels. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [0]:

```
#Saving model
from google.colab import files
files.download('model.h5')
```

Mobilenet Convolutional Architecture

In [1]:
```
#Importing all neccessary librarys
import shutil
import numpy as np
import pandas as pd
from random import random

# Image operations and plotting
from PIL import Image
from skimage.io import imread, imshow
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid")
%matplotlib inline

# File, path and directory operations
import os
import os.path
import shutil
from glob import glob

# We are only using Keras for data augmentation
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator, img_to_arra
y, load_img

# Model building
from fastai.vision import *
from fastai.callbacks.hooks import *
import torchvision

# Data preparation
from sklearn.model_selection import train_test_split

# For reproducability
from numpy.random import seed
seed(108)

# For aesthetics
import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
```

In [2]:
```
print(os.listdir("../input"))
```

```
['skin-cancer-mnist-ham10000']
```

Loading HAM10000_metadata and splitting into train , validation and test

In [3]:
```
print(os.listdir("../input/skin-cancer-mnist-ham10000"))
```

```
['hmnist_28_28_RGB.csv', 'ham10000_images_part_1', 'HAM10000_image
s_part_2', 'hmnist_28_28_L.csv', 'HAM10000_images_part_1', 'HAM1000
0_metadata.csv', 'hmnist_8_8_RGB.csv', 'hmnist_8_8_L.csv', 'ham1000
0_images_part_2']
```

In [4]:
```
# Create a new directory
base = "base"
os.mkdir(base)
```

In [5]:
```
#[CREATE FOLDERS INSIDE THE BASE DIRECTORY]

# now we create 7 folders inside 'base':

# train
    # nv
    # mel
    # bkl
    # bcc
    # akiec
    # vasc
    # df

# valid
    # nv
    # mel
    # bkl
    # bcc
    # akiec
    # vasc
    # df

# create a path to 'base' to which we will join the names of the new f
olders
# train
train = os.path.join(base, 'train')
os.mkdir(train)

# valid
valid = os.path.join(base, 'valid')
os.mkdir(valid)


# [CREATE FOLDERS INSIDE THE TRAIN, VALIDATION AND TEST FOLDERS]
# Inside each folder we create seperate folders for each class

# create new folders inside train
nv = os.path.join(train, 'nv')
os.mkdir(nv)
mel = os.path.join(train, 'mel')
```

```
os.mkdir(mel)
bkl = os.path.join(train, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(train, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(train, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(train, 'vasc')
os.mkdir(vasc)
df = os.path.join(train, 'df')
os.mkdir(df)




# test
test = os.path.join(base, 'test')
os.mkdir(test)

nv = os.path.join(test, 'nv')
os.mkdir(nv)
mel = os.path.join(test, 'mel')
os.mkdir(mel)
bkl = os.path.join(test, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(test, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(test, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(test, 'vasc')
os.mkdir(vasc)
df = os.path.join(test, 'df')
os.mkdir(df)
```

In [6]:
```
# create new folders inside valid
nv = os.path.join(valid, 'nv')
os.mkdir(nv)
mel = os.path.join(valid, 'mel')
os.mkdir(mel)
bkl = os.path.join(valid, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(valid, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(valid, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(valid, 'vasc')
os.mkdir(vasc)
df = os.path.join(valid, 'df')
os.mkdir(df)
```

In [7]:
```
import pandas as pd
df = pd.read_csv("../input/skin-cancer-mnist-ham10000/HAM10000_metadata.csv")
```

In [8]:
```
from numpy.random import seed
seed(101)
df2=df.iloc[:,1:3]
msk = np.random.rand(len(df2)) < 0.85
train1_df2 = df2[msk]
test_df2 = df2[~msk]
msk1 = np.random.rand(len(train1_df2)) < 0.85
train_df2 = train1_df2[msk1]
validation_df2 = train1_df2[~msk1]
```

In [9]:
```
train_df2['dx'].value_counts()
```

Out[9]:
```
nv       4783
mel       846
bkl       799
bcc       378
akiec     237
vasc      101
df         80
Name: dx, dtype: int64
```

In [10]:
```
validation_df2['dx'].value_counts()
```

Out[10]:
```
nv       864
mel      125
bkl      119
bcc       70
akiec     42
df        18
vasc      17
Name: dx, dtype: int64
```

In [11]:
```
test_df2['dx'].value_counts()
```

Out[11]:
```
nv       1058
bkl       181
mel       142
bcc        66
akiec      48
vasc       24
df         17
Name: dx, dtype: int64
```

In [12]:
```
# Set the image_id as the index in df_data
df.set_index('image_id', inplace=True)
```

In [13]:
```
# Get a list of images in each of the two folders
folder_1 = os.listdir('../input/skin-cancer-mnist-ham10000/HAM10000_images_part_1')
folder_2 = os.listdir('../input/skin-cancer-mnist-ham10000/HAM10000_images_part_2')
```

In [14]:
```
# Get a list of train , val and test images

train_df2_list = list(train_df2['image_id'])
validation_df2_list = list(validation_df2['image_id'])
test_df2_list = list(test_df2['image_id'])
```

In [15]:
```
# Transfer the train images

for image in train_df2_list:

    fname = image + '.jpg'
    label = df.loc[image,'dx']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_1', fname)
        # destination path to image
        dst = os.path.join(train, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_2', fname)
        # destination path to image
        dst = os.path.join(train, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

In [16]:
```
# Transfer the val images

for image in validation_df2_list:

    fname = image + '.jpg'
    label = df.loc[image,'dx']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_1', fname)
        # destination path to image
        dst = os.path.join(valid, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_2', fname)
        # destination path to image
        dst = os.path.join(valid, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

In [17]:
```
for image in test_df2_list:

    fname = image + '.jpg'
    label = df.loc[image,'dx']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_1', fname)
        # destination path to image
        dst = os.path.join(test, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_2', fname)
        # destination path to image
        dst = os.path.join(test, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

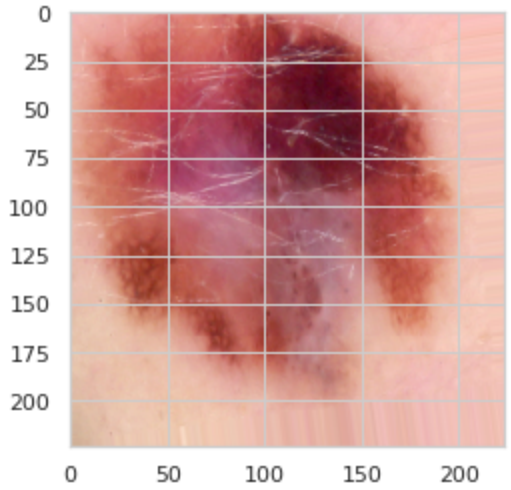
In [18]:
```
# check how many train images we have in each folder
print("............................")
print("Train folder")
print("............................")
print(len(os.listdir('base/train/nv')))
print(len(os.listdir('base/train/mel')))

print(len(os.listdir('base/train/bkl')))
print(len(os.listdir('base/train/bcc')))
print(len(os.listdir('base/train/akiec')))
print(len(os.listdir('base/train/vasc')))
print(len(os.listdir('base/train/df')))
print("............................")
# check how many train images we have in each folder
print("validation folder")
print("............................")
print(len(os.listdir('base/valid/nv')))
print(len(os.listdir('base/valid/mel')))
print(len(os.listdir('base/valid/bkl')))
print(len(os.listdir('base/valid/bcc')))
print(len(os.listdir('base/valid/akiec')))
print(len(os.listdir('base/valid/vasc')))
print(len(os.listdir('base/valid/df')))
print("............................")
# check how many train images we have in each folder
print("Test folder")
print("............................")
print(len(os.listdir('base/test/nv')))
print(len(os.listdir('base/test/mel')))
print(len(os.listdir('base/test/bkl')))
print(len(os.listdir('base/test/bcc')))
print(len(os.listdir('base/test/akiec')))
print(len(os.listdir('base/test/vasc')))
print(len(os.listdir('base/test/df')))
```

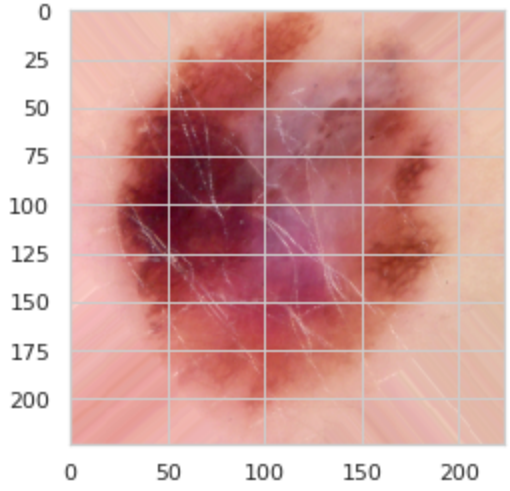
```
.............................
Train folder
.............................
4783
846
799
378
237
101
80
.............................
validation folder
.............................
864
125
119
70
42
17
18
.............................
Test folder
.............................
1058
142
181
66
48

24
17
```

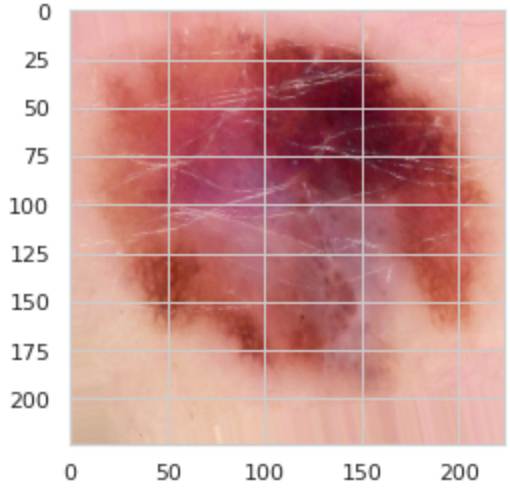
Balancing the unbalanced classes using Data Augmentation technique

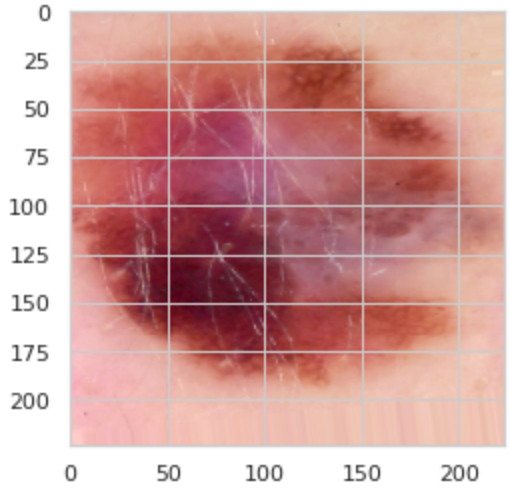
In [19]:

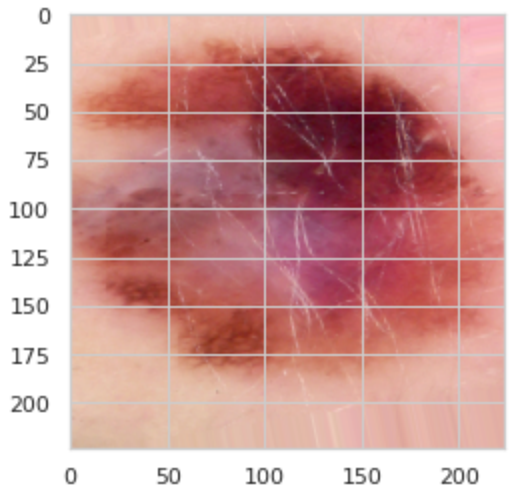
```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from tensorflow.keras.preprocessing.image import array_to_img, img_t
o_array, load_img
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=180,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest')
img_path = load_img('../input/skin-cancer-mnist-ham10000/HAM10000_imag
es_part_2/ISIC_0029316.jpg',target_size=(224, 224))
 # this is a PIL image
x = img_to_array(img_path)  # Numpy array with shape (224, 224, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 224, 224,
3)

# The .flow() command below generates batches of randomly transformed
images
# It will loop indefinitely, so we need to `break` the loop at some po
int!
i = 0
for batch in datagen.flow(x, batch_size=1):
  plt.figure(i)
  imgplot = plt.imshow(array_to_img(batch[0]))
  i += 1
  if i % 5 == 0:
    break
```




0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200
0
50
100
150
200



In [20]:
```
# note that we are not augmenting class 'nv'
class_list = ['mel','bkl','bcc','akiec','vasc','df']

for item in class_list:

    # We are creating temporary directories here because we delete the
se directories later
    # create a base
    aug = 'aug'
    os.mkdir(aug)
    # create a dir within the base to store images of the same class
    img_dir = os.path.join(aug, 'img_dir')
    os.mkdir(img_dir)

    # Choose a class
    img_class = item

    # list all images in that directory
    img_list = os.listdir('base/train/' + img_class)

    # Copy images from the class train dir to the img_dir e.g. class
'mel'
    for fname in img_list:
            # source path to image
            src = os.path.join('base/train/' + img_class, fname)
            # destination path to image
            dst = os.path.join(img_dir, fname)
            # copy the image from the source to the destination
            shutil.copyfile(src, dst)


    # point to a dir containing the images and not to the images thems
elves
    path = aug
    save_path = 'base/train/' + img_class

    # Create a data generator
    datagen = ImageDataGenerator(
        rotation_range=180,
```

```
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        vertical_flip=True,
        #brightness_range=(0.9,1.1),
        fill_mode='nearest')

    batch_size = 50

    aug_datagen = datagen.flow_from_directory(path,
                                              save_to_dir=save_path,
                                              save_format='jpg',
                                                      target_size=(224,2
24),
                                                      batch_size=batch_s
ize)



    # Generate the augmented images and add them to the training folde
rs

    ###########

    num_aug_images_wanted = 5000 # total number of images we want to h
ave in each class

    ###########

    num_files = len(os.listdir(img_dir))
    num_batches = int(np.ceil((num_aug_images_wanted-num_files)/batc
h_size))

    # run the generator and create about 6000 augmented images
    for i in range(0,num_batches):

        imgs, labels = next(aug_datagen)

    # delete temporary directory with the raw image files
    shutil.rmtree('aug')
```

```
Found 846 images belonging to 1 classes.
Found 799 images belonging to 1 classes.
Found 378 images belonging to 1 classes.
Found 237 images belonging to 1 classes.
Found 101 images belonging to 1 classes.
Found 80 images belonging to 1 classes.
```

In [21]:
```
# check how many train images we have in each folder
print("...............................")
print("Train folder")
print("...............................")
print(len(os.listdir('base/train/nv')))
print(len(os.listdir('base/train/mel')))
print(len(os.listdir('base/train/bkl')))
print(len(os.listdir('base/train/bcc')))
print(len(os.listdir('base/train/akiec')))
print(len(os.listdir('base/train/vasc')))
print(len(os.listdir('base/train/df')))
print("...............................")
# check how many train images we have in each folder
print("validation folder")
print("...............................")
print(len(os.listdir('base/valid/nv')))
print(len(os.listdir('base/valid/mel')))
print(len(os.listdir('base/valid/bkl')))
print(len(os.listdir('base/valid/bcc')))
print(len(os.listdir('base/valid/akiec')))
print(len(os.listdir('base/valid/vasc')))
print(len(os.listdir('base/valid/df')))
print("...............................")
# check how many train images we have in each folder
print("Test folder")
print("...............................")
print(len(os.listdir('base/test/nv')))
print(len(os.listdir('base/test/mel')))
print(len(os.listdir('base/test/bkl')))
print(len(os.listdir('base/test/bcc')))
print(len(os.listdir('base/test/akiec')))
print(len(os.listdir('base/test/vasc')))
print(len(os.listdir('base/test/df')))
```

```
.............................
Train folder
.............................
4783
5030
5044
4786
4790
3433
4050
.............................
validation folder
.............................
864
125
119
70
42
17
18
.............................
Test folder
.............................
1058
142
181
66
48
24

17
```

Using MobileNet Architecture

In [22]:
```
# Define transformations for data augmentation
tfms = get_transforms(do_flip=True,
                      max_rotate=10,
                      max_zoom=1.1,
                      max_warp=0.2)

# Build dataset by applying transforms to the data from our directory
data = (ImageList.from_folder(base)
        .split_by_folder()
        .label_from_folder()
        .add_test_folder('test')
        .transform(tfms, size=224)
        .databunch()
        .normalize(imagenet_stats))
```

In [23]:
```
wd=1e-2

mobilenet_split = lambda m: (m[0][0][10], m[1])
arch  = torchvision.models.mobilenet_v2
learn = cnn_learner(data, arch, cut=-1, split_on=mobilenet_split, wd=w
d, metrics=[accuracy])
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353
104.pth" to /root/.cache/torch/checkpoints/mobilenet_v2-b0353104.pth
```

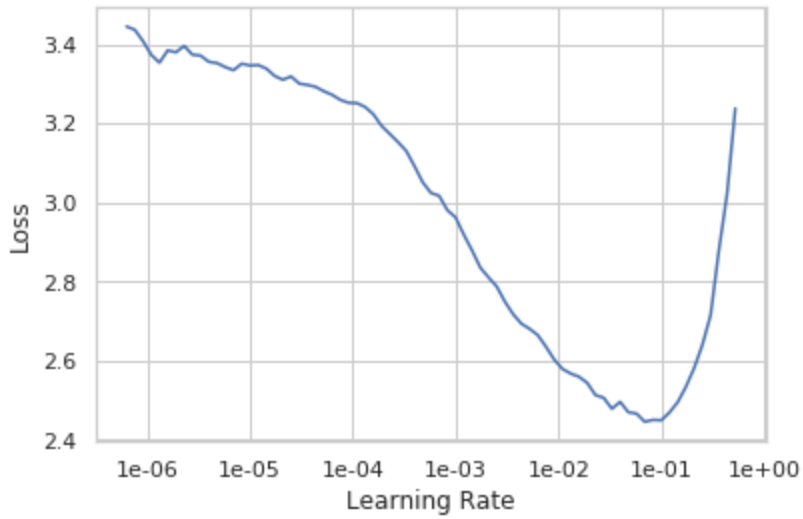
In [24]:
```
learn.lr_find();
learn.recorder.plot();
```

0.00% [0/1 00:00<00:00]

epoch	train_loss	valid_loss	accuracy	time

17.87% [89/498 01:07<05:12 8.3212]

```
LR Finder is complete, type {learner_name}.recorder.plot() to see th
e graph.
```



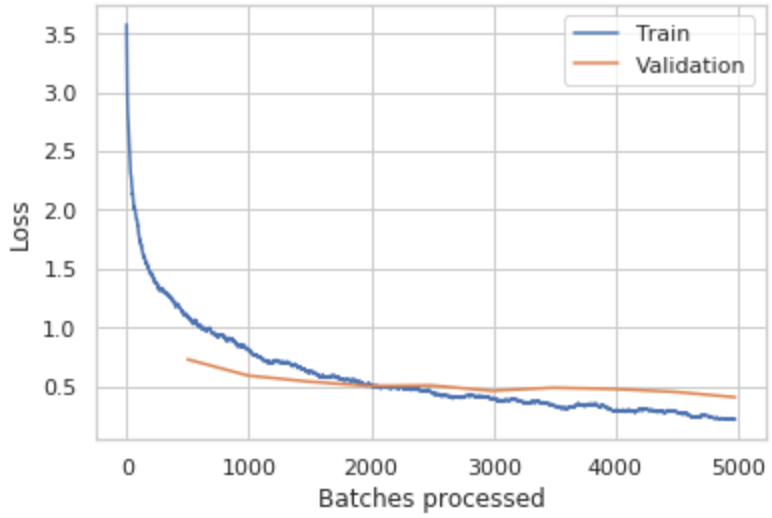
In [25]:
```
# Set our learning rate to the value where learning is fastest and los
s
# is still decreasing.
lr=3e-3

# This function uses our input lr as an anchor and sweeps through a ra
nge
# in order to search out the best local minima.
learn.fit_one_cycle(10, slice(lr), pct_start=0.9)
```

epoch	train_loss	valid_loss	accuracy	time
0	1.088360	0.730347	0.778486	06:41
1	0.807788	0.592182	0.807171	06:48
2	0.627882	0.541020	0.811155	06:53
3	0.516480	0.503540	0.812749	06:59
4	0.453764	0.509392	0.816733	06:58
5	0.394898	0.462734	0.831076	06:59
6	0.341926	0.489190	0.829482	07:09
7	0.293263	0.478801	0.837450	06:50
8	0.295792	0.454103	0.837450	06:40
9	0.221239	0.408573	0.861355	06:44

In [26]:
```
# Save the current state of our model
learn.save('mobile_v1_stage-1')
```

Plot of training and validation loss

In [27]:
```
learn.recorder.plot_losses()
```



In [28]:
```
# Exctract predictions and losses to evaluate model

preds,y,losses = learn.get_preds(with_loss=True)
interp = ClassificationInterpretation(learn, preds, y, losses)
```

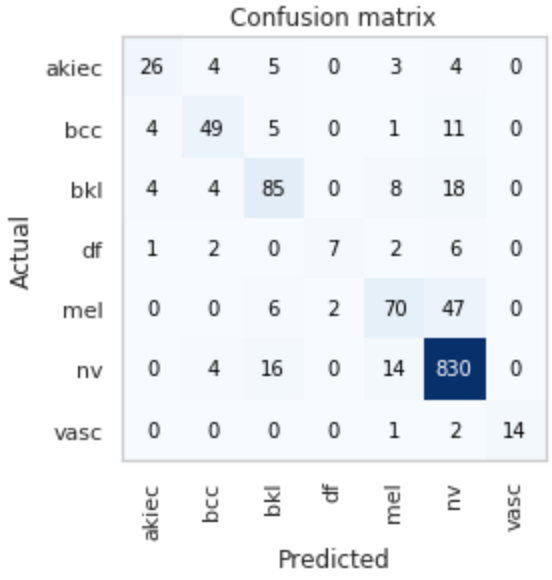
In [29]:
```
def top_k_spread(preds, y, spread):
  for i in range(spread):
    print(f"Top {i+1} accuracy: {top_k_accuracy(preds, y, i+1)}")
```

In [30]:
```
# Top-1 accuracy of 86% is quite near the best models from the open co
mpetition
top_k_spread(preds, y, 5)
```

```
Top 1 accuracy: 0.8613545894622803
Top 2 accuracy: 0.9505975842475891
Top 3 accuracy: 0.9832669496536255
Top 4 accuracy: 0.9920318722724915
Top 5 accuracy: 0.9968127608299255
```

Confusion Matrix Report

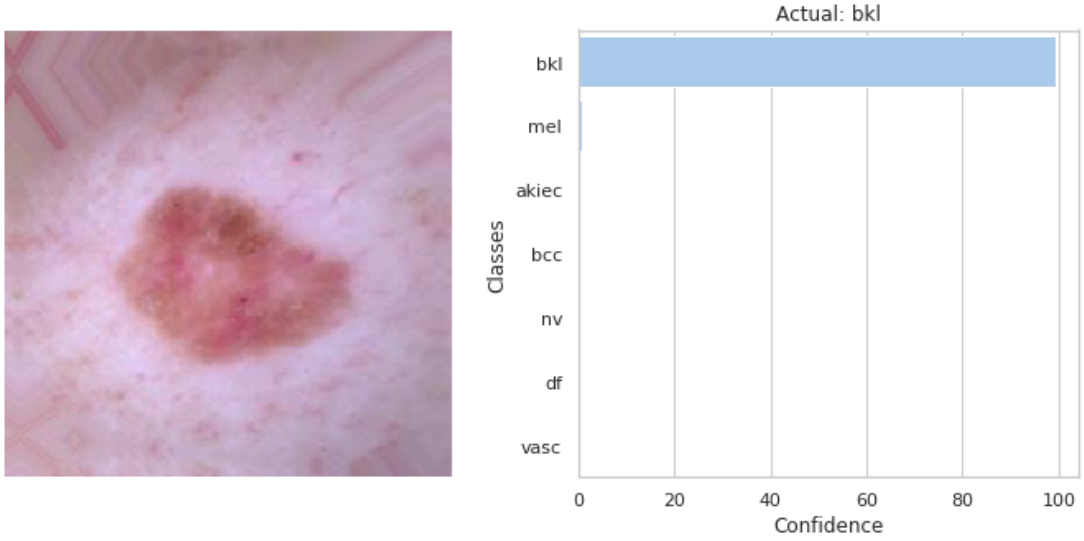
In [31]:
```
interp.plot_confusion_matrix()
```



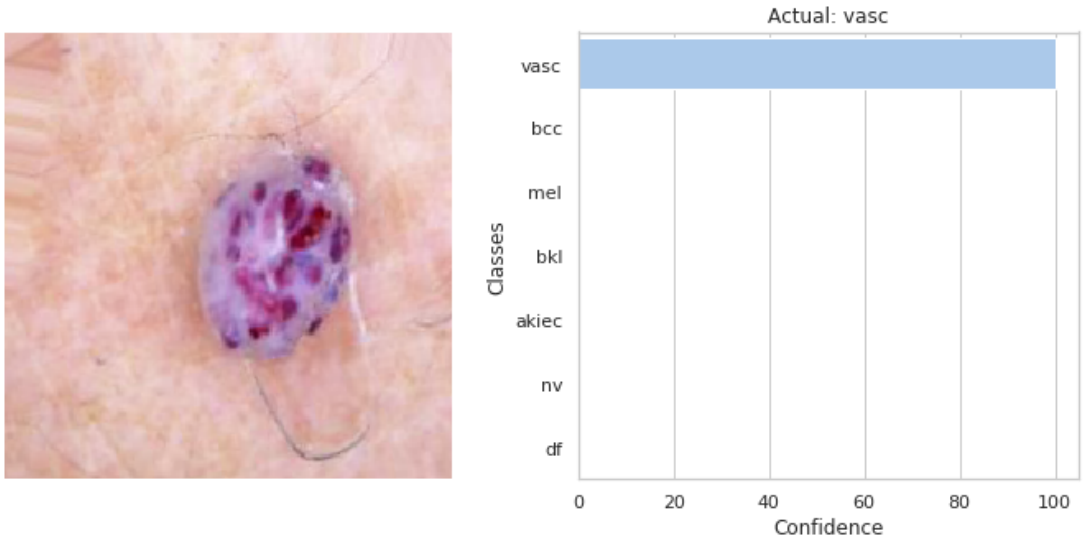
In [32]:
```
# This function creates a display of the models prediction and confide
nce levels
def plot_prediction(learner, index):
  data = learner.data.train_ds[index][0]
  pred = learner.predict(data)
  classes = learner.data.classes

  prediction = pd.DataFrame(to_np(pred[2]*100), columns=['Confidenc
e'])
  prediction['Classes'] = classes
  prediction = prediction.sort_values(by='Confidence', ascending=Fals
e)

  fig = plt.figure(figsize=(12, 5))
  ax1 = fig.add_subplot(121)
  show_image(data, figsize=(5, 5), ax=ax1)
  ax2 = fig.add_subplot(122)
  sns.set_color_codes("pastel")
  sns.barplot(x='Confidence', y='Classes', data=prediction,
              label="Total", color="b")
  ax2.set_title(f'Actual: {learn.data.train_ds[index][1]}')
```

Predictions of some random images

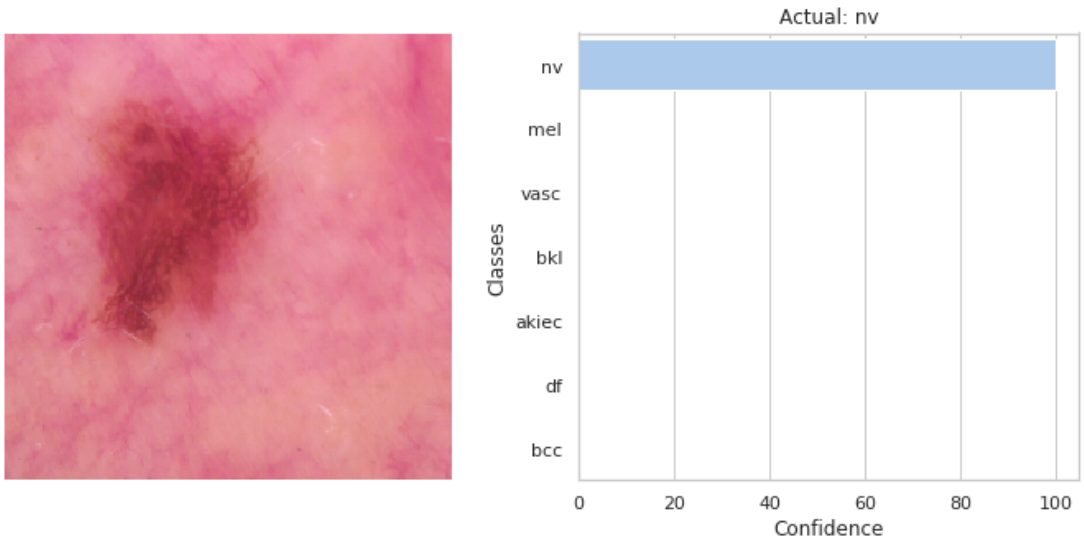
In [33]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



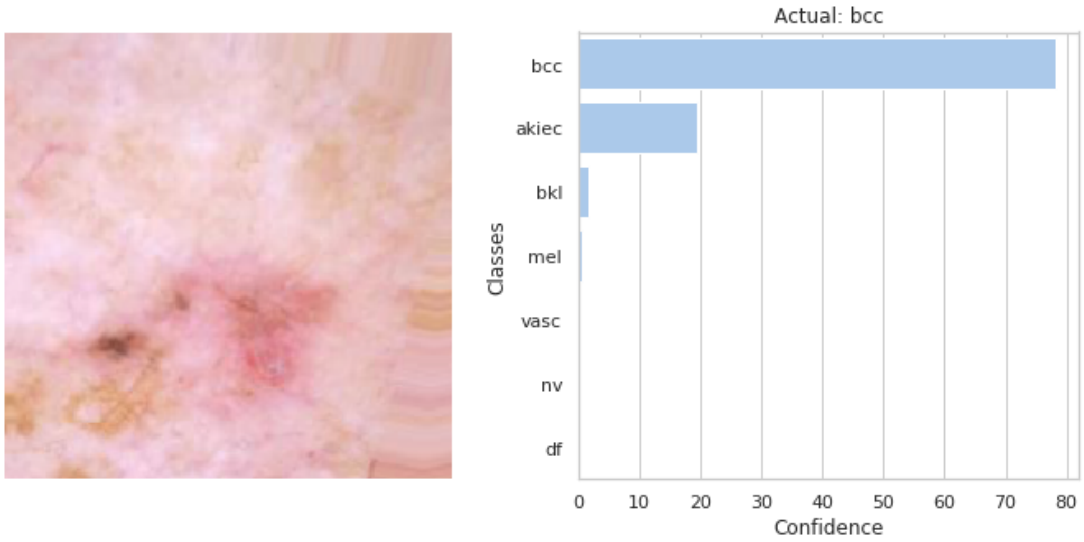
In [34]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



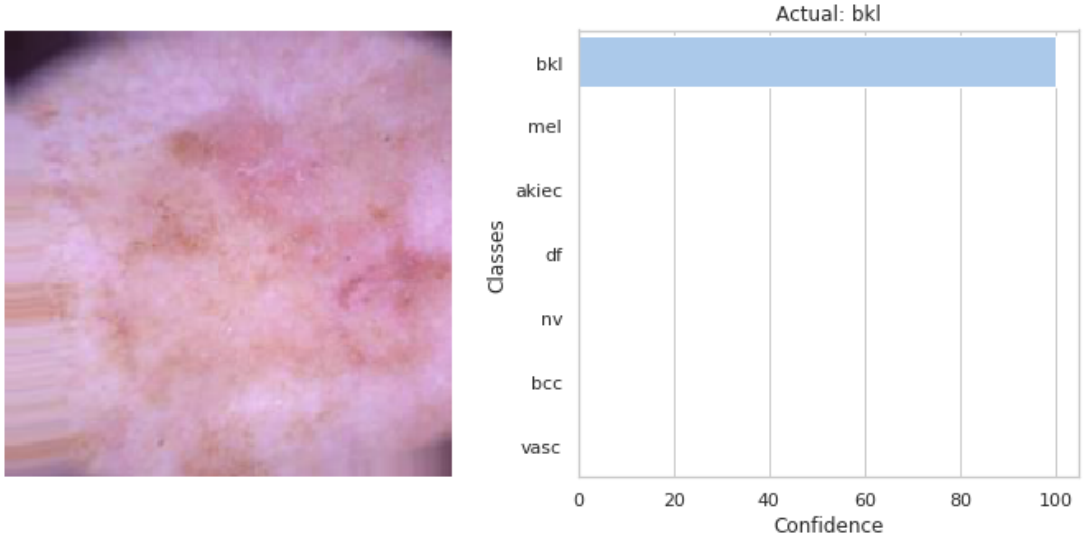
In [35]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



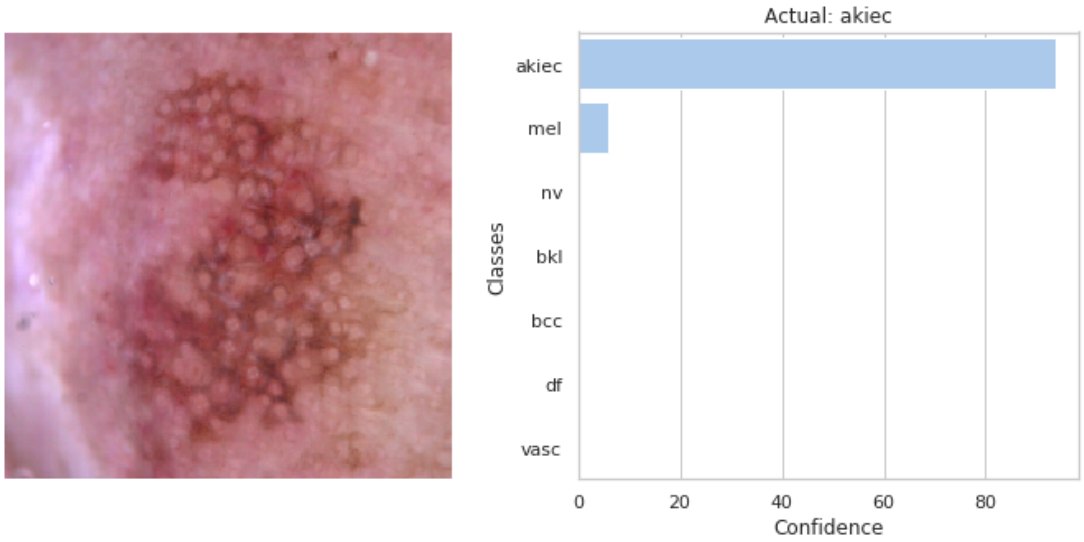
In [36]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



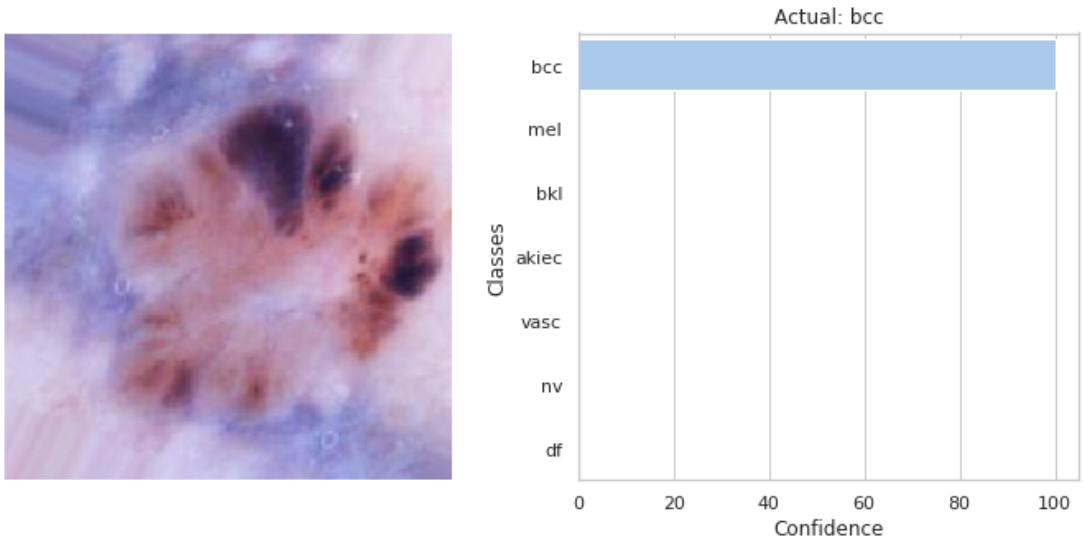
In [37]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



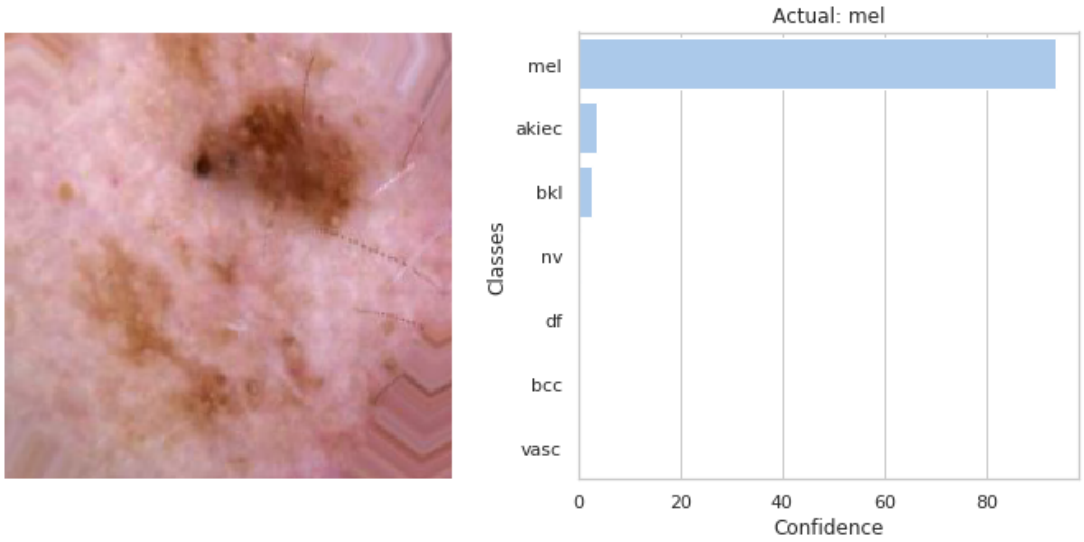
In [38]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



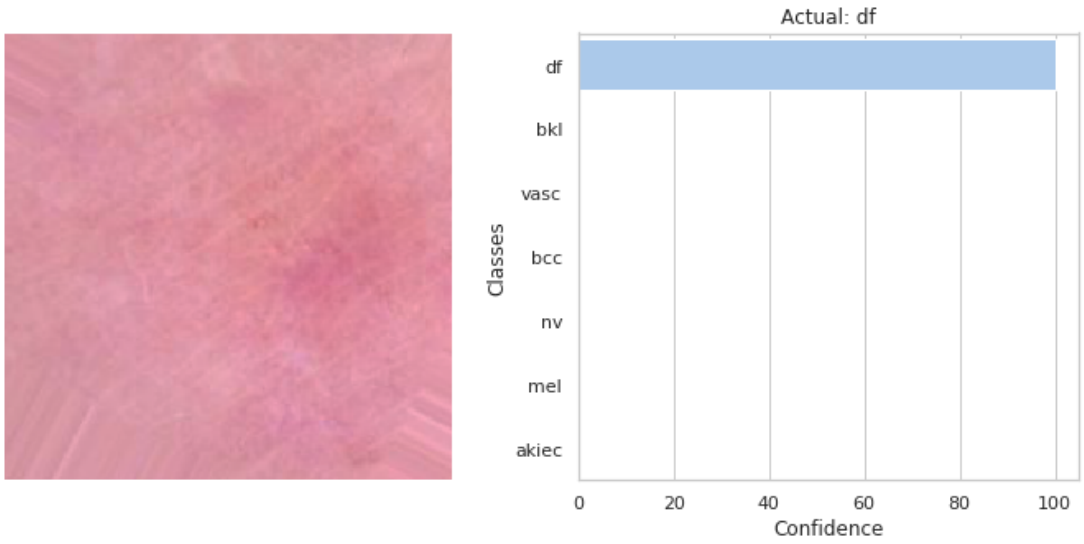
In [39]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



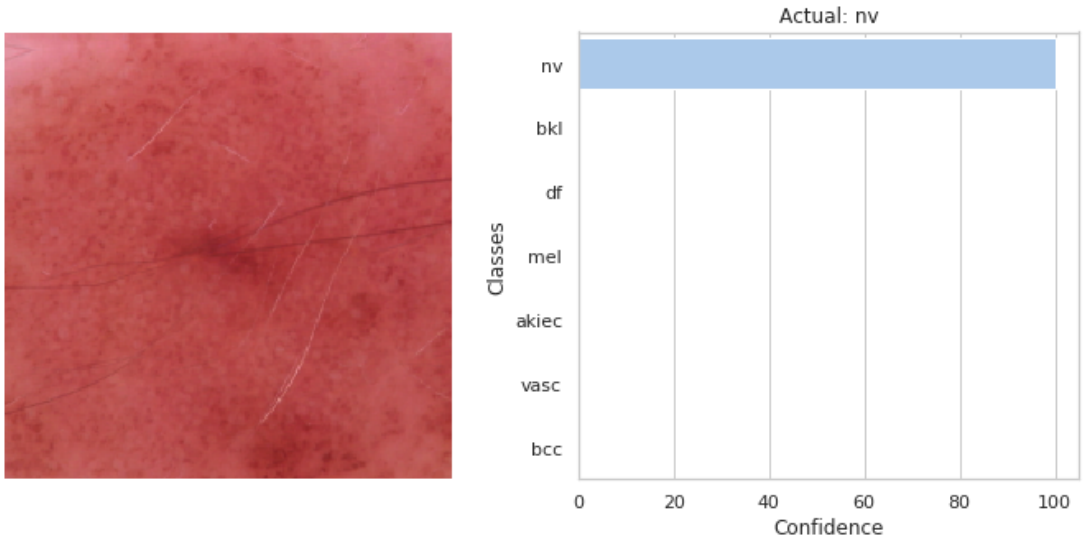
In [40]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



In [41]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



In [42]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



In [ ]:
```
```

Analysis report for Test images.

In [33]:
```
learn.export()
```

In [34]:
```
learn.path
```

Out[34]:
```
PosixPath('base')
```

In [35]:
```
learn = load_learner(base,test=ImageList.from_folder('/kaggle/working/base/test'))
```

In [36]:
```
preds,y = learn.get_preds(ds_type=DatasetType.Test)
preds = np.argmax(preds, 1).tolist()
```

In [37]:
```
for i in range(0,7):
    print('The count of element:', i ,'is ', preds.count(i))
```

```
The count of element: 0 is  22
The count of element: 1 is  78
The count of element: 2 is  170
The count of element: 3 is  11
The count of element: 4 is  100
The count of element: 5 is  1130
The count of element: 6 is  25
```

In [38]:
```
y_true=[]
for i in list(data.test_ds.items):
    if PurePath(i).parts[2]=="akiec":
        y_true.append(int(str(0)))
    elif PurePath(i).parts[2]=="bcc":
        y_true.append(int(str(1)))
    elif PurePath(i).parts[2]=="bkl":
        y_true.append(int(str(2)))
```

```
    elif PurePath(i).parts[2]=="df":
        y_true.append(int(str(3)))
    elif PurePath(i).parts[2]=="mel":
        y_true.append(int(str(4)))
    elif PurePath(i).parts[2]=="nv":
        y_true.append(int(str(5)))
    else:
        y_true.append(int(str(6)))
```

In [39]:
```
target_names = ['akiec', 'bcc','bkl','df','mel','nv','vasc']
print(classification_report(y_true, preds, target_names=target_names))
```

```
              precision    recall  f1-score   support

       akiec       0.77      0.35      0.49        48
         bcc       0.68      0.80      0.74        66
         bkl       0.67      0.63      0.65       181
          df       0.73      0.47      0.57        17
         mel       0.61      0.43      0.50       142
          nv       0.89      0.95      0.92      1058
        vasc       0.88      0.92      0.90        24

    accuracy                           0.83      1536
   macro avg       0.75      0.65      0.68      1536
weighted avg       0.82      0.83      0.82      1536
```

In [40]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    import itertools
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
```

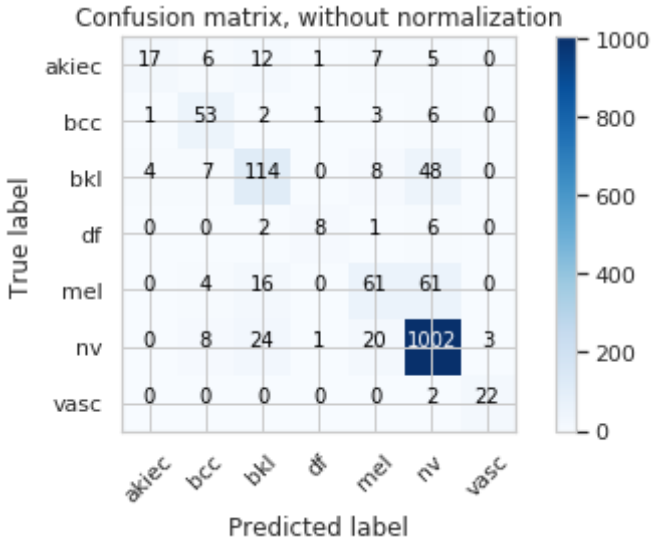
```
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1
])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [41]:

```
cnf_matrix = confusion_matrix(y_true, preds,labels=[0,1,2,3,4,5,6])
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['akiec', 'bcc','bkl','df','m
el','nv','vasc'],
                      title='Confusion matrix, without normalization')
```

```
Confusion matrix, without normalization
[[  17    6   12    1    7    5    0]
 [   1   53    2    1    3    6    0]
 [   4    7  114    0    8   48    0]
 [   0    0    2    8    1    6    0]
 [   0    4   16    0   61   61    0]
```

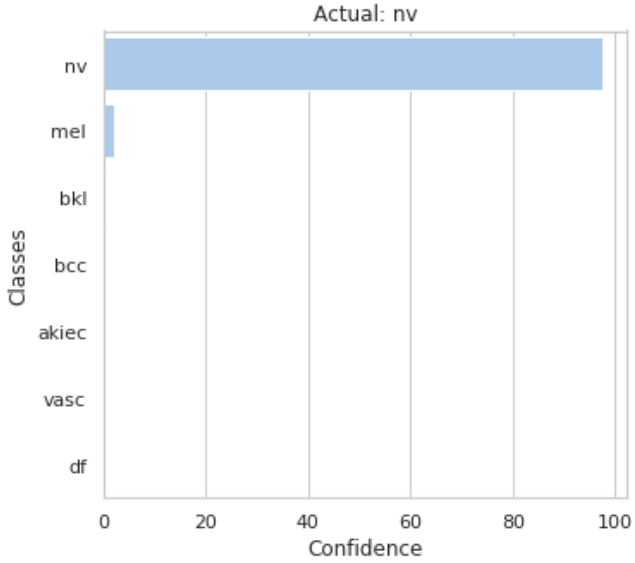
```
[   0    8   24    1   20 1002    3]
[   0    0    0    0    0    2   22]]
```



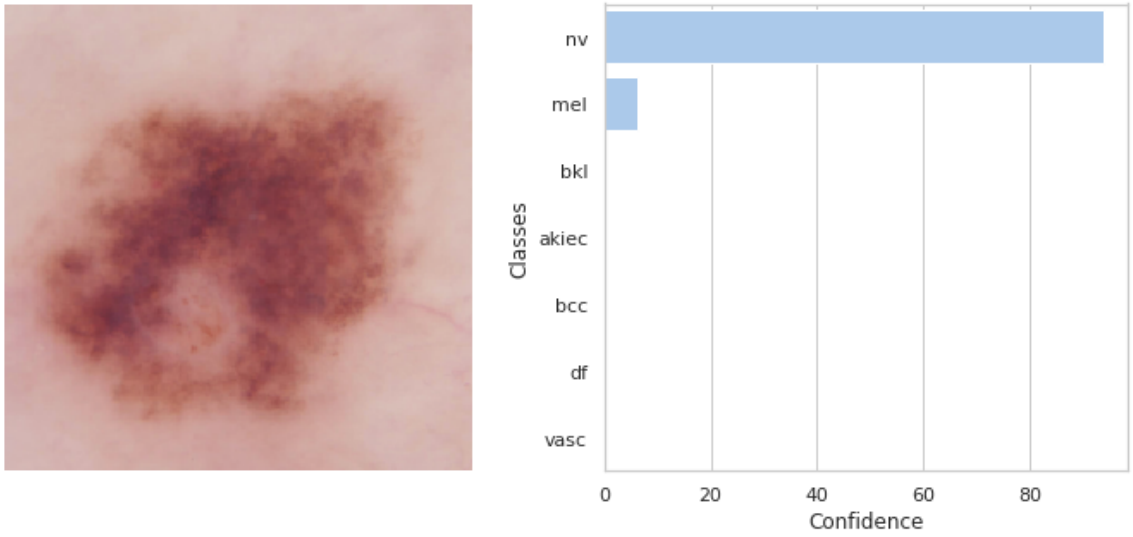
In [42]:
```
def plot_prediction(learn, index):
  data = learn.data.test_ds[index][0]
  pred = learn.predict(data)
  classes = learn.data.classes

  prediction = pd.DataFrame(to_np(pred[2]*100), columns=['Confidence'])
  prediction['Classes'] = classes
  prediction = prediction.sort_values(by='Confidence', ascending=False)

  fig = plt.figure(figsize=(12, 5))
  ax1 = fig.add_subplot(121)
  show_image(data, figsize=(5, 5), ax=ax1)
  ax2 = fig.add_subplot(122)
  sns.set_color_codes("pastel")
  sns.barplot(x='Confidence', y='Classes', data=prediction,
              label="Total", color="b")
  ax2.set_title(f'Actual: {PurePath(learn.data.test_ds.items[index]).pa
rts[5]}')
```

**Predictions

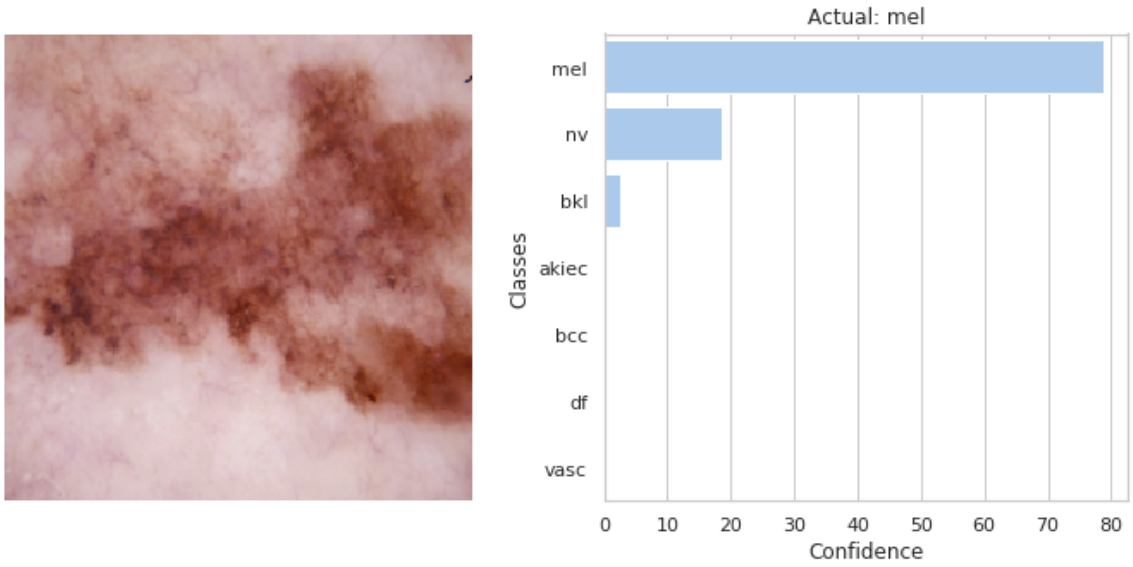
In [43]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



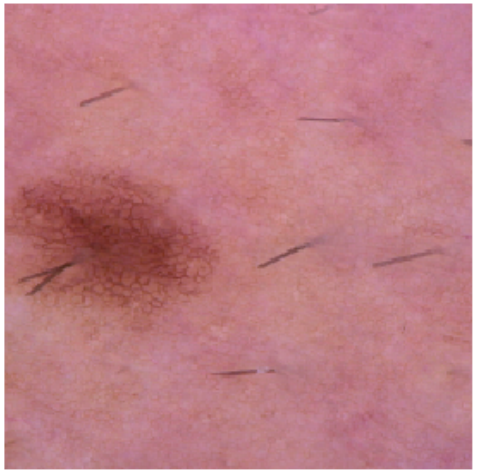
In [44]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```

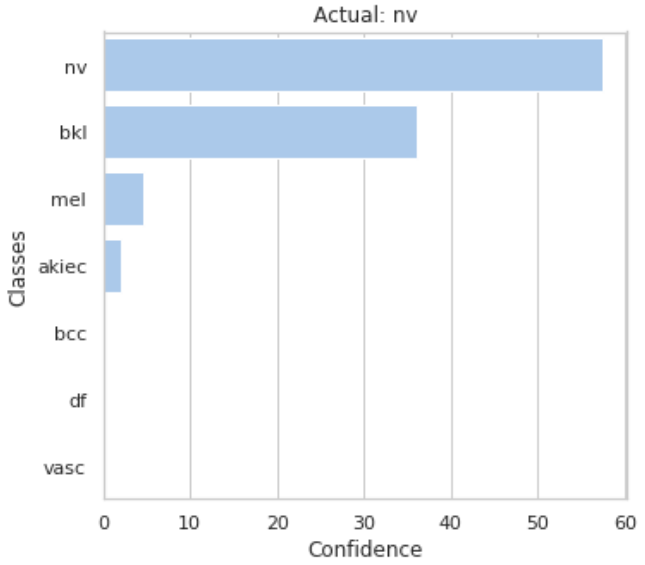


In [45]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```

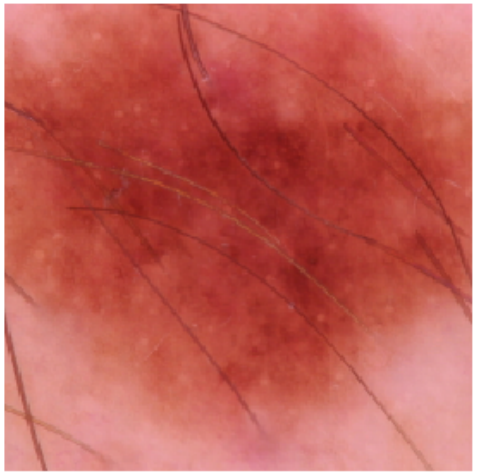


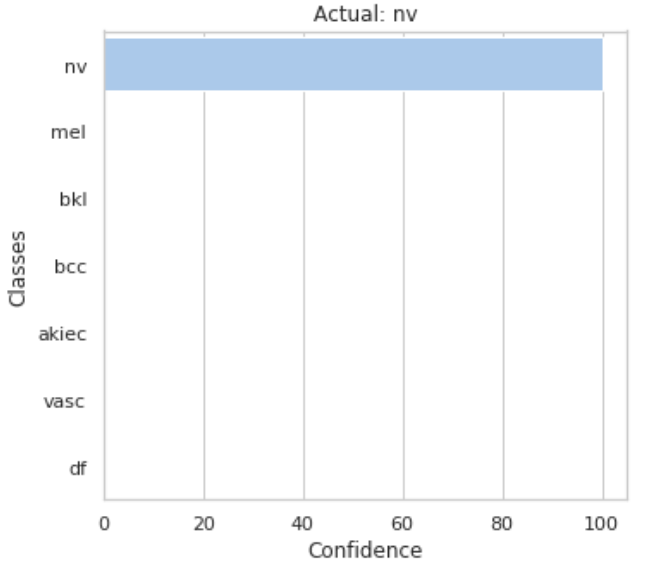
In [46]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



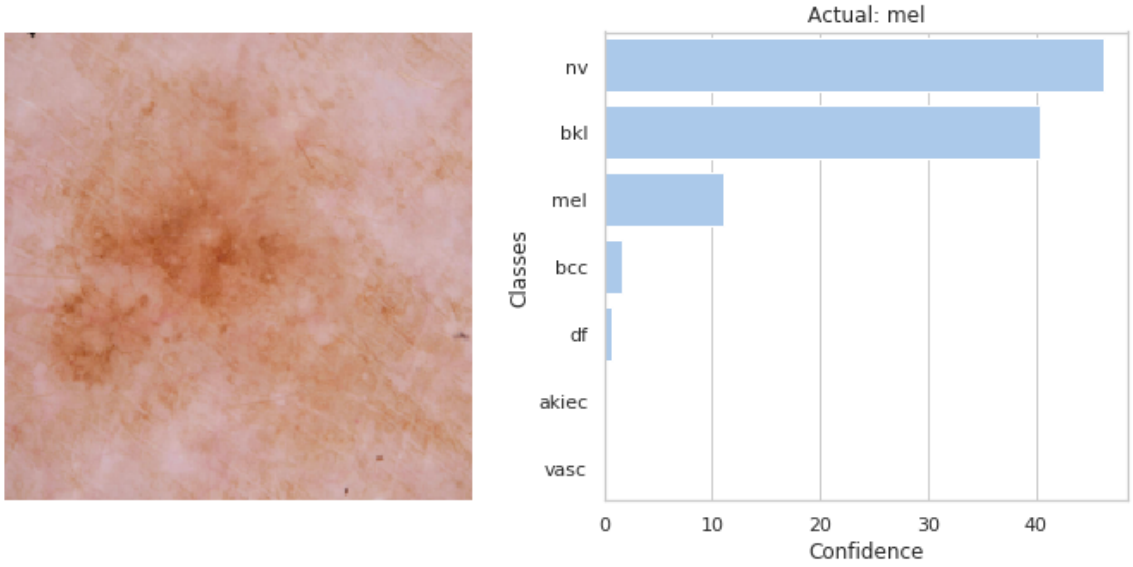


In [47]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```

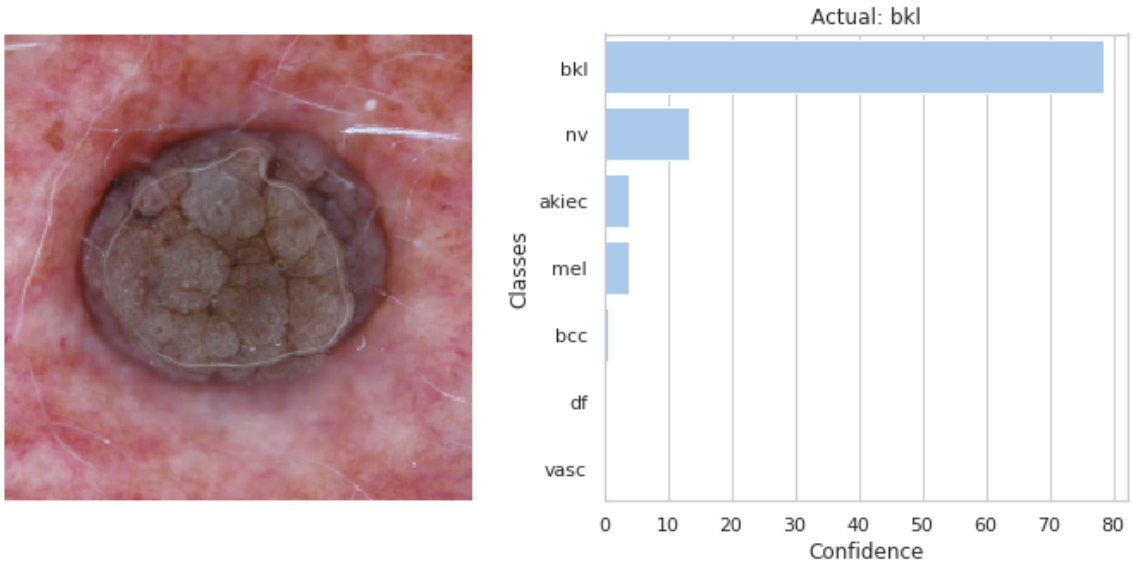




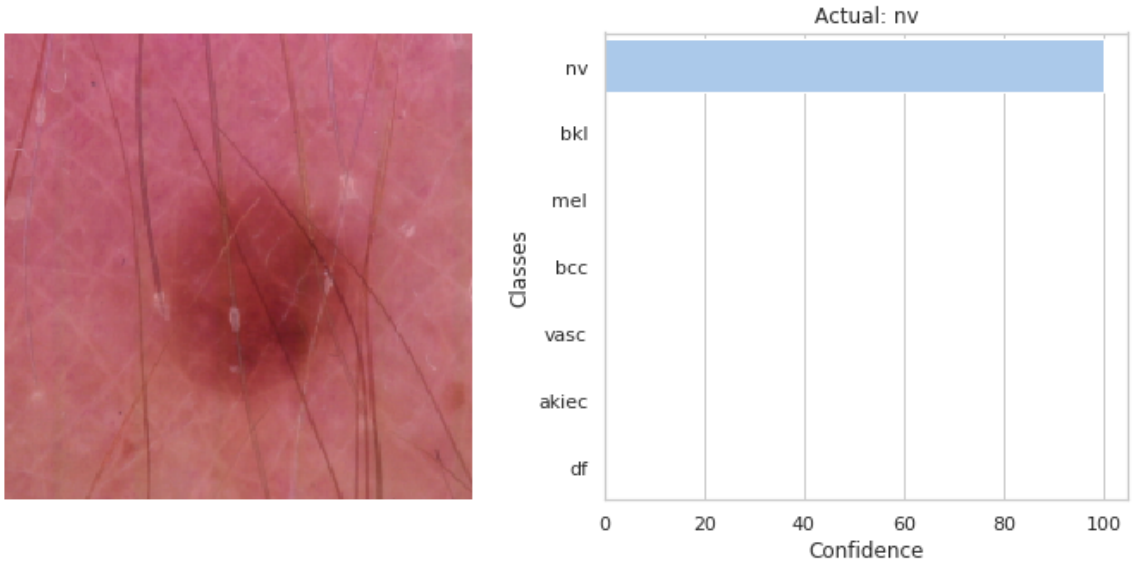
In [48]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



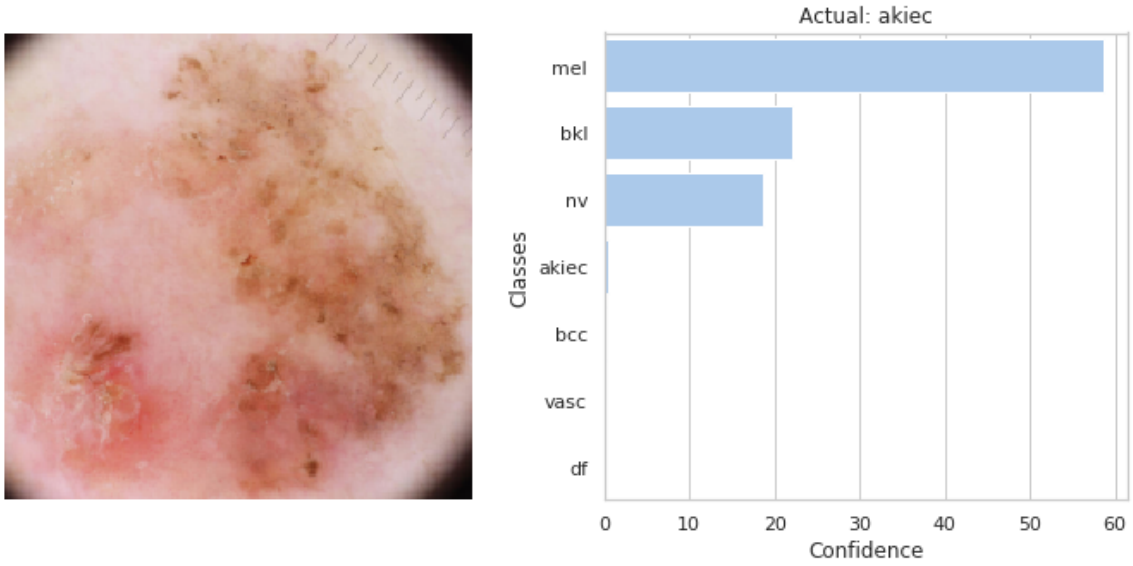
In [49]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



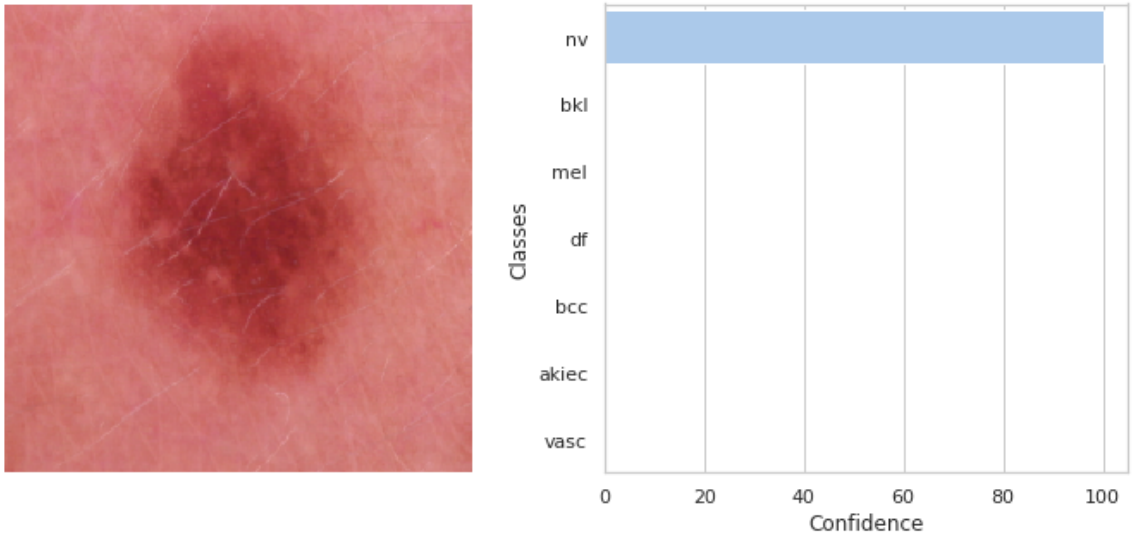
In [50]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



In [51]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



In [52]:
```
plot_prediction(learn, np.random.choice(len(learn.data.test_ds)))
```



In [ ]:
```

```