SKIN DISEASE CLASSIFICATION

Deep learning to predict the various skin diseases. The main objective of this project is to achieve maximum accuracy of skin disease prediction. Deep learning techniques helps in detection of skin disease at an initial stage. The feature extraction plays a key role in classification of skin diseases.The usage of Deep Learning algorithms reduces the need for human labor, such as manual feature extraction and data reconstruction for classification purpose. Moreover, Explainable AI is used to interpret the decisions made by our model.

ABOUT THE DATASET

HAM10000 ("Human Against Machine with 10000 training images") dataset - a large collection of multi-source dermatoscopic images of pigmented lesions

The dermatoscopic images are collected from different populations, acquired and stored by different modalities. The final dataset consists of 10015 dermatoscopic images.

It has 7 different classes of skin cancer which are listed below :

- Melanocytic nevi
- Melanoma
- Benign keratosis-like lesions
- Basal cell carcinoma
- Actinic keratoses
- Vascular lesions
- Dermatofibroma

In [30]:
```
#Importing required libraries
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
import numpy as np
import pandas as pd
import os
from tensorflow.keras.utils import to_categorical
from glob import glob
```

HAM10000_metadata.csv file is the main csv file that includes the data of all training images, the features of which are -

1. Lesion_id
2. Image_id
3. Dx
4. Dx_type
5. Age

6. Sex
7. Localization

In [31]:
```
# Reading the data from HAM_metadata.csv
df = pd.read_csv('filename_to_category_map_surnamesIJKLM.csv')
```

In [32]:
```
df
```

Out[32]:

	image_id	cell_type	is_benign	localization
0	ISIC_0027419	Benign keratosis-like lesions	1.0	scalp
1	ISIC_0026769	Benign keratosis-like lesions	1.0	scalp
2	ISIC_0031633	Benign keratosis-like lesions	1.0	ear
3	ISIC_0029176	Benign keratosis-like lesions	1.0	face
4	ISIC_0029068	Benign keratosis-like lesions	1.0	face
...	...	...	...	...
5371	ISIC_0032134	Melanocytic nevi	1.0	NaN
5372	ISIC_0032861	Benign keratosis-like lesions	1.0	NaN
5373	ISIC_0026054	Benign keratosis-like lesions	1.0	NaN
5374	ISIC_0032052	Melanoma	1.0	NaN
5375	ISIC_0031933	Benign keratosis-like lesions	1.0	NaN

5376 rows × 4 columns

In [33]:
```
df.dtypes
```

Out[33]:
```
image_id         object
cell_type        object
is_benign       float64
localization     object
dtype: object
```

In [34]:
```
df.describe()
```

Out[34]:

	is_benign
count	5276.000000
mean	0.807051
std	0.394651
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

A general statistical analysis of the numerical values of dataset (here : age)

Data Cleaning

Removing NULL values and performing visualizations to gain insights of dataset: Univariate and Bivariate Analysis

In [35]:
```
df.isnull().sum()
```

Out[35]:
```
image_id          0
cell_type       100
is_benign       100
localization     18
dtype: int64
```

The feature 'age' consists of 57 null records. Thus, we need to replace them with the mean of 'age' since dropping 57 records would lead to loss of data.

In [36]:
```
#df['age'].fillna(int(df['age'].mean()),inplace=True)
df1 = df.dropna()
df1.isnull().sum()
```

Out[36]:
```
image_id        0
cell_type       0
is_benign       0
localization    0
dtype: int64
```

In [37]:
```
df1.isnull().sum()
df1.to_csv('file_name.csv')
```

Now, the null values have been removed.

In [38]:
```
lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}
base_skin_dir = 'ham_data_surnamesIJKLM'

# Merge images from both folders into one dictionary

imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join(base_skin_dir, '*.jpg'))}
```

In [39]:
```
df1['path'] = df1['image_id'].map(imageid_path_dict.get)
#df['cell_type'] = df['dx'].map(lesion_type_dict.get)
#df['cell_type_idx'] = pd.Categorical(df['cell_type']).codes
df1.head()
```

```
C:\Users\wajah\AppData\Local\Temp/ipykernel_7072/3826673078.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1['path'] = df1['image_id'].map(imageid_path_dict.get)
```

Out[39]:

	image_id	cell_type	is_benign	localization	path
0	ISIC_0027419	Benign keratosis-like lesions	1.0	scalp	ham_data_surnamesIJKLM\ISIC_0027419.jpg
1	ISIC_0026769	Benign keratosis-like lesions	1.0	scalp	ham_data_surnamesIJKLM\ISIC_0026769.jpg
2	ISIC_0031633	Benign keratosis-like lesions	1.0	ear	ham_data_surnamesIJKLM\ISIC_0031633.jpg
3	ISIC_0029176	Benign keratosis-like lesions	1.0	face	ham_data_surnamesIJKLM\ISIC_0029176.jpg
4	ISIC_0029068	Benign keratosis-like lesions	1.0	face	ham_data_surnamesIJKLM\ISIC_0029068.jpg

Image Preprocessing

Resizing of images because the original dimensions of 450 *600* 3 take long time to process in Neural Networks

In [40]:
```
df1['image'] = df1['path'].map(lambda x: np.asarray(Image.open(x).resize((125,100))))
```

```
C:\Users\wajah\AppData\Local\Temp/ipykernel_7072/3360847188.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1['image'] = df1['path'].map(lambda x: np.asarray(Image.open(x).resize((125,10
0))))
```

Showcasing some samples of each class of the dataset in the images below:

In [41]:
```
n_samples = 5
fig, m_axs = plt.subplots(7, n_samples, figsize = (4*n_samples, 3*7))
for n_axs, (type_name, type_rows) in zip(m_axs,
                                         df1.sort_values(['cell_type']).groupby('cell_
    n_axs[0].set_title(type_name)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples, random_state=2018).
        c_ax.imshow(c_row['image'])
        c_ax.axis('off')
fig.savefig('category_samples.png', dpi=300)
```

In [42]:
```
# See the image size distribution - should just return one row (all images are uniform
df1['image'].map(lambda x: x.shape).value_counts()
```

Out[42]:
```
(100, 125, 3)    5258
Name: image, dtype: int64
```

Exploratory Data Analysis

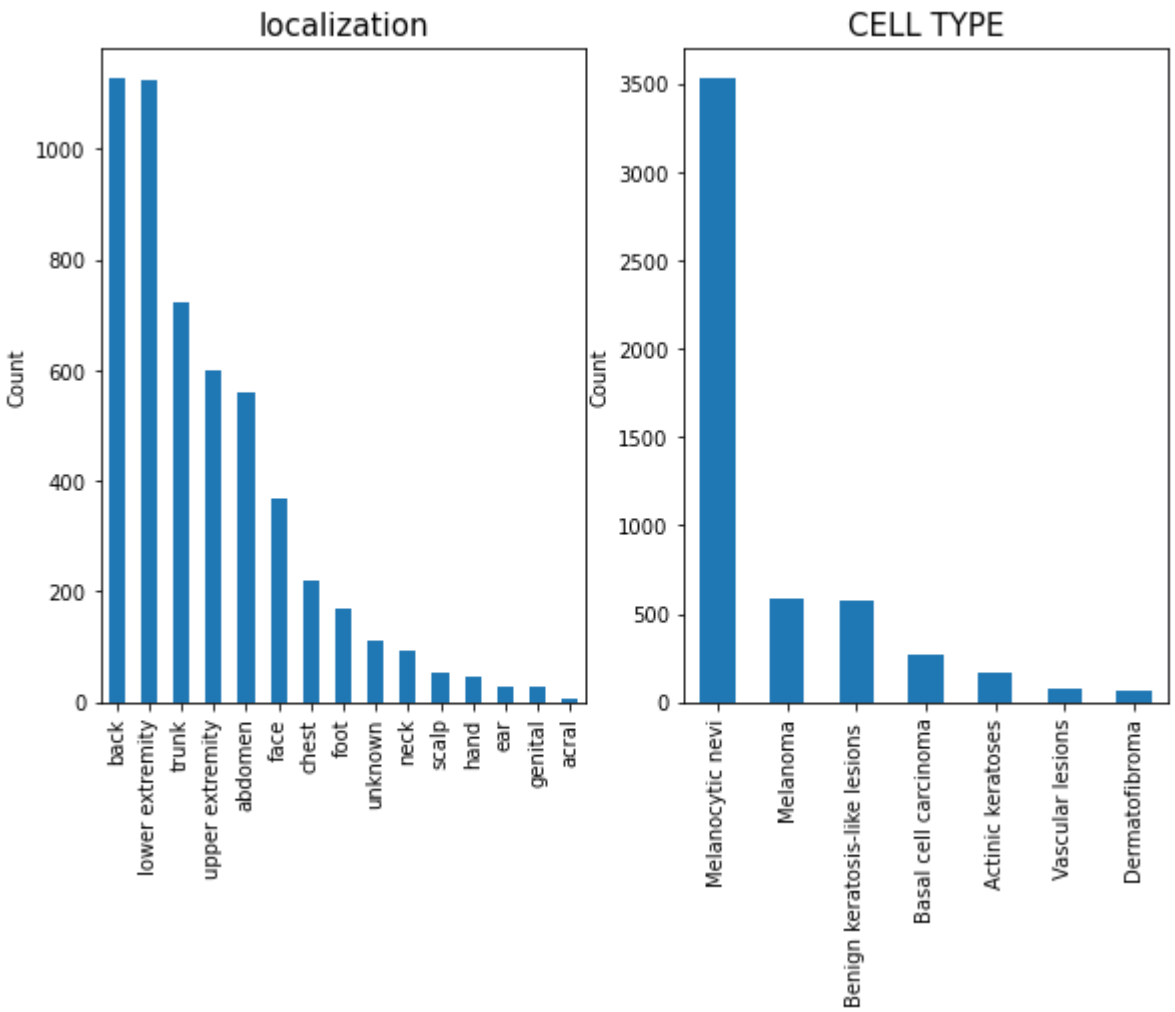
Exploratory data analysis can help detect obvious errors, identify outliers in datasets, understand relationships, unearth important factors, find patterns within data, and provide new insights.

In [43]:
```
#df= df[df['age'] != 0]
#df= df[df['sex'] != 'unknown']
```

UNIVARIATE ANALYSIS

In [44]:
```
plt.figure(figsize=(20,10))
plt.subplots_adjust(left=0.125, bottom=1, right=0.9, top=2, hspace=0.2)


plt.subplot(2,4,3)
plt.title("localization",fontsize=15)
plt.ylabel("Count")
plt.xticks(rotation=45)
df1['localization'].value_counts().plot.bar()

plt.subplot(2,4,4)
plt.title("CELL TYPE",fontsize=15)
plt.ylabel("Count")
df1['cell_type'].value_counts().plot.bar()
```

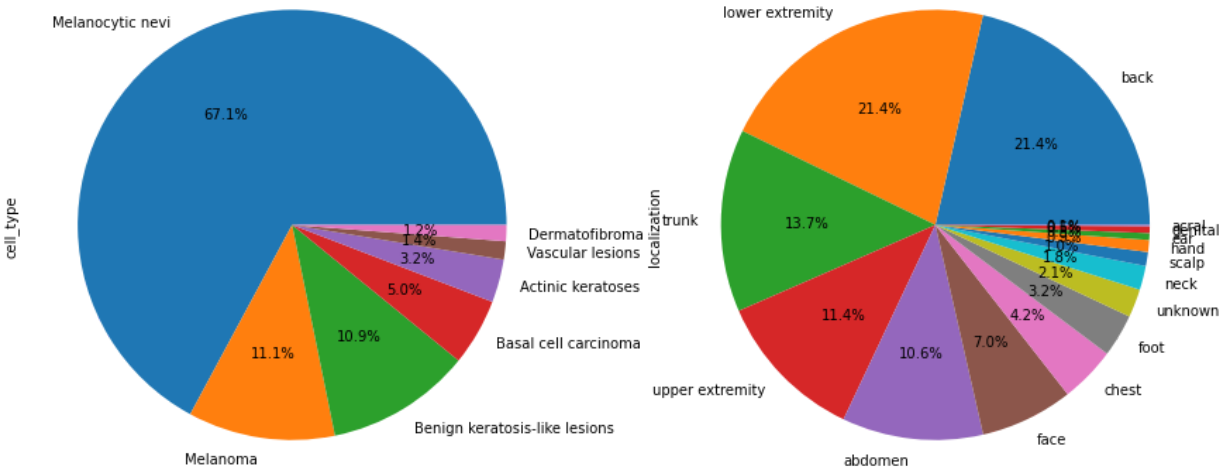
Out[44]:
```
<AxesSubplot:title={'center':'CELL TYPE'}, ylabel='Count'>
```



1. Skin diseases are found to be maximum in people aged around 45. Minimum for 10 and below. We also observe that the probability of having skin disease increases with the increase in age.
2. Skin diseases are more prominent in Men as compared to Women and other gender.

3. Skin diseases are more visible on the "back" of the body and least on the "acral surfaces" (such as limbs, fingers, or ears).
4. The most found disease among people is Melanocytic nevi while the least found is Dermatofibroma.

In [45]:
```
plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
df1['cell_type'].value_counts().plot.pie(autopct="%1.1f%%")
plt.subplot(1,2,2)
df1['localization'].value_counts().plot.pie(autopct="%1.1f%%")
plt.show()
```

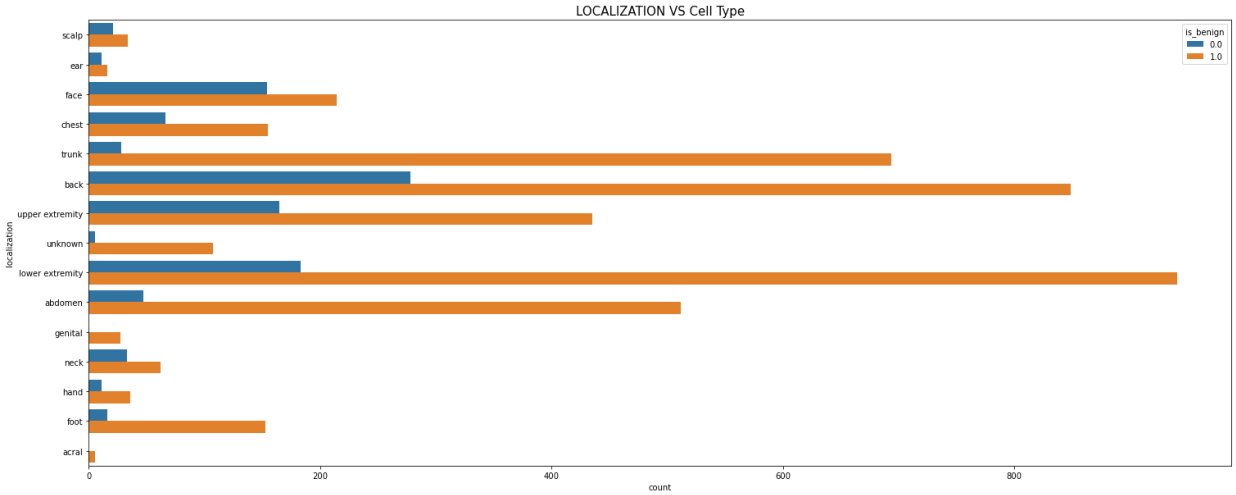


1. Type of skin disease:
    - nv: Melanocytic nevi - 69.9%
    - mel: Melanoma - 11.1 %
    - bkl: Benign keratosis-like lesions - 11.0%
    - bcc: Basal cell carcinoma - 5.1%
    - akiec: Actinic keratoses- 3.3%
    - vasc: Vascular lesions-1.4%
    - df: Dermatofibroma - 1.1%
2. How the skin disease was discovered:
    - histo - histopathology - 53.3%
    - follow_up - follow up examination - 37.0%
    - consensus - expert consensus - 9.0%
    - confocal - confirmation by in-vivo confocal microscopy - 0.7%

BIVARIATE ANALYSIS

In [46]:
```
plt.figure(figsize=(25,10))
plt.title('LOCALIZATION VS Cell Type',fontsize = 15)
sns.countplot(y='localization', hue='is_benign',data=df1)
```

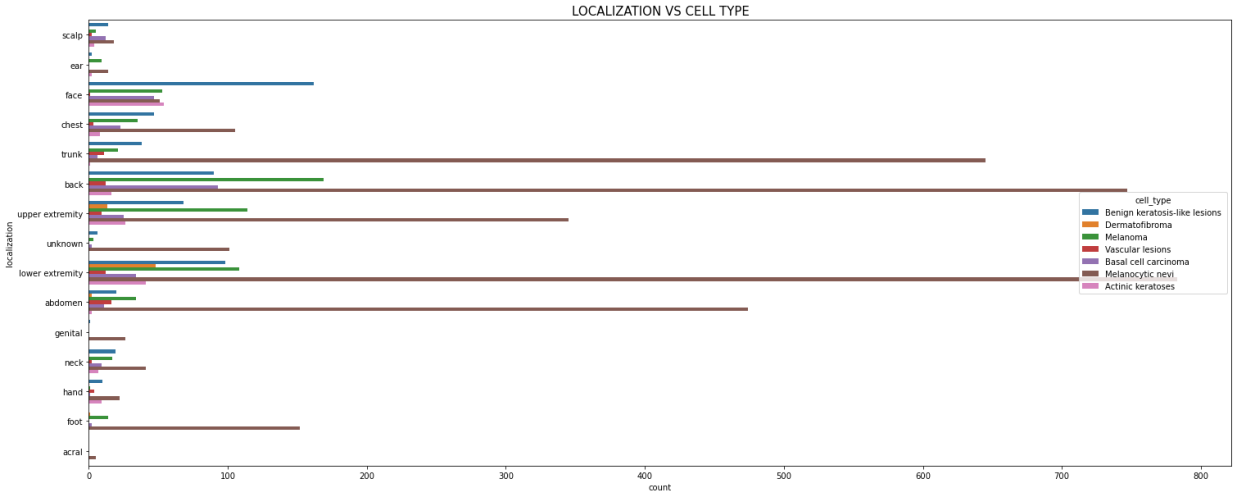
Out[46]:
```
<AxesSubplot:title={'center':'LOCALIZATION VS Cell Type'}, xlabel='count', ylabel='lo
calization'>
```



- Back are is the most affected among people and more prominent in men.
- Infection on Lower extremity of the body is more visible in women.
- Some unknown regions also show infections and it's visible in men, women and other genders.
- The acral surfaces show the least infection cases that too in men only. Other gender groups don't show this kind of infection.

In [47]:
```
plt.figure(figsize=(25,10))
plt.title('LOCALIZATION VS CELL TYPE',fontsize = 15)
sns.countplot(y='localization', hue='cell_type',data=df1)
```

Out[47]:
```
<AxesSubplot:title={'center':'LOCALIZATION VS CELL TYPE'}, xlabel='count', ylabel='lo
calization'>
```

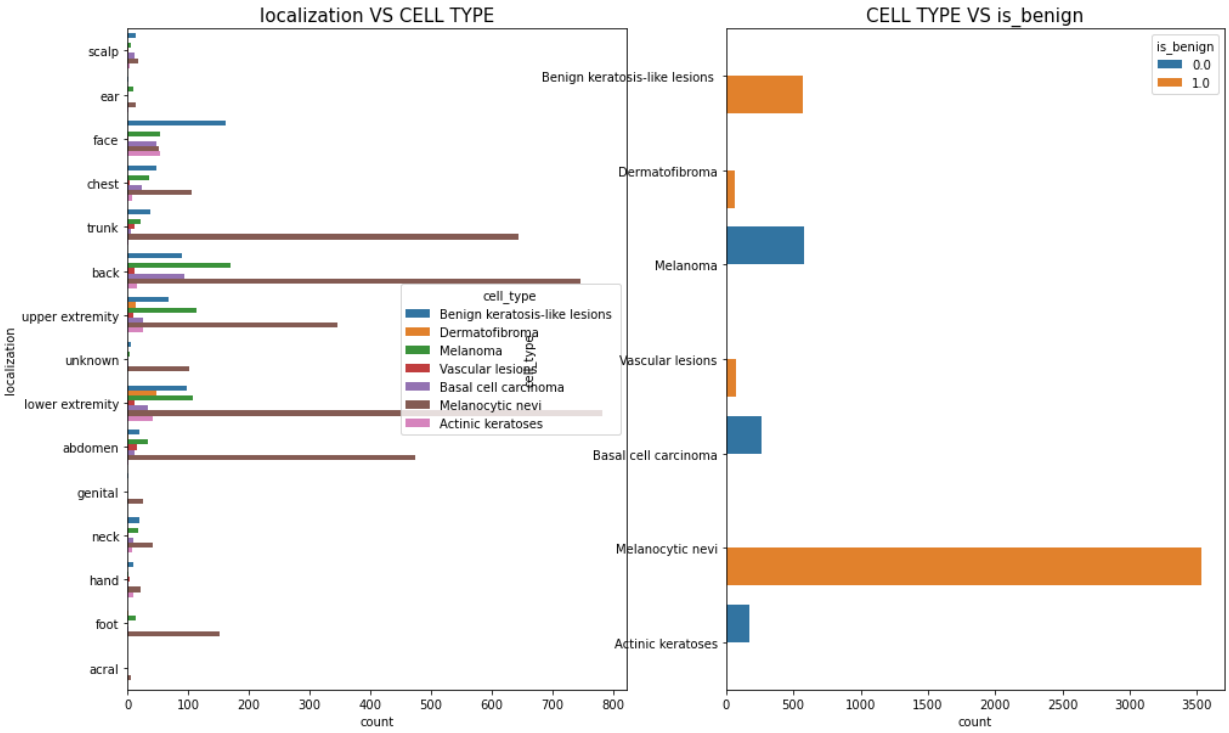


- The face is infected the most by Benign keratosis-like lesions.
- Body parts(except face) are infected the most by Melanocytic nevi.

In [48]:
```
plt.figure(figsize=(25,10))
plt.subplot(131)
plt.title('localization VS CELL TYPE',fontsize = 15)
sns.countplot(y='localization', hue='cell_type',data=df1)
```

```
plt.subplot(132)
plt.title('CELL TYPE VS is_benign',fontsize = 15)
sns.countplot(y='cell_type', hue='is_benign',data=df1)
```

Out[48]:
```
<AxesSubplot:title={'center':'CELL TYPE VS is_benign'}, xlabel='count', ylabel='cell_
type'>
```



1. The age group between 0-75 years is infected the most by Melanocytic nevi. On the other hand, the people aged 80-90 are affected more by Benign keratosis-like lesions.

2. All the gender groups are affected the most by Melanocytic nevi.

In [49]:
```
from sklearn.model_selection import train_test_split
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
```

ANN

A simple neural network is implemented first with the following layers to get patterns in images. The accuracy achieved is approximately 69 percent.

In [50]:
```
features=df1.drop(columns=['is_benign'],axis=1)
target=df1['is_benign']
```

In [51]:
```
features.head()
```

Out[51]:

	image_id	cell_type	localization	path	image
0	ISIC_0027419	Benign keratosis-like lesions	scalp	ham_data_surnamesIJKLM\ISIC_0027419.jpg	[[[189, 152, 194], [192, 156, 198], [191, 154,...
1	ISIC_0026769	Benign keratosis-like lesions	scalp	ham_data_surnamesIJKLM\ISIC_0026769.jpg	[[[186, 127, 135], [189, 133, 145], [192, 135,...
2	ISIC_0031633	Benign keratosis-like lesions	ear	ham_data_surnamesIJKLM\ISIC_0031633.jpg	[[[131, 88, 110], [142, 97, 120], [152, 107, 1...
3	ISIC_0029176	Benign keratosis-like lesions	face	ham_data_surnamesIJKLM\ISIC_0029176.jpg	[[[190, 144, 126], [192, 145, 130], [194, 146,...
4	ISIC_0029068	Benign keratosis-like lesions	face	ham_data_surnamesIJKLM\ISIC_0029068.jpg	[[[147, 103, 83], [153, 111, 91], [159, 119, 1...

In [52]:
```
x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(features, target, test_siz
tf.unique(x_train_o.cell_type.values)
```

Out[52]:
```
Unique(y=<tf.Tensor: shape=(7,), dtype=string, numpy=
array([b'Melanocytic nevi', b'Basal cell carcinoma',
       b'Benign keratosis-like lesions ', b'Melanoma',
       b'Actinic keratoses', b'Dermatofibroma', b'Vascular lesions'],
      dtype=object)>, idx=<tf.Tensor: shape=(3943,), dtype=int32, numpy=array([0, 0,
0, ..., 0, 0, 0])>)
```

In [53]:
```
x_train = np.asarray(x_train_o['image'].tolist())
x_test = np.asarray(x_test_o['image'].tolist())

x_train_mean = np.mean(x_train)
x_train_std = np.std(x_train)

x_test_mean = np.mean(x_test)
x_test_std = np.std(x_test)

x_train = (x_train - x_train_mean)/x_train_std
x_test = (x_test - x_test_mean)/x_test_std
```

In [54]:
```
# Perform one-hot encoding on the labels
y_train = to_categorical(y_train_o, num_classes = 7)
```

```
y_test = to_categorical(y_test_o, num_classes = 7)
y_test
```

Out[54]:
```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

In [55]:
```
x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, test_siz
# Reshape image in 3 dimensions (height = 100, width = 125 , canal = 3)
x_train = x_train.reshape(x_train.shape[0], *(100, 125, 3))
x_test = x_test.reshape(x_test.shape[0], *(100, 125, 3))
x_validate = x_validate.reshape(x_validate.shape[0], *(100, 125, 3))
```

In [57]:
```
x_train = x_train.reshape(3548,125*100*3)
x_test = x_test.reshape(1315,125*100*3)
print(x_train.shape)
print(x_test.shape)
```

```
(3548, 37500)
(1315, 37500)
```

In [58]:
```
# define the keras model
model = Sequential()

model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu', input_
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 7, kernel_initializer = 'uniform', activation = 'softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate = 0.00075,
                                     beta_1 = 0.9,
                                     beta_2 = 0.999,
                                     epsilon = 1e-8)

# compile the keras model
model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['ac

# fit the keras model on the dataset
history = model.fit(x_train, y_train, batch_size = 10, epochs = 50)

accuracy = model.evaluate(x_test, y_test, verbose=1)[1]
print("Test: accuracy = ",accuracy*100,"%")
```

```
Epoch 1/50
355/355 [==============================] - 8s 18ms/step - loss: 0.5011 - accuracy: 0.
8047
Epoch 2/50
355/355 [==============================] - 8s 21ms/step - loss: 0.4329 - accuracy: 0.
8075
Epoch 3/50
355/355 [==============================] - 8s 22ms/step - loss: 0.4060 - accuracy: 0.
8120 4s - loss: 0.3981 - accuracy: 0.81 - ETA: 4s - loss: 0.3
Epoch 4/50
355/355 [==============================] - 9s 25ms/step - loss: 0.3898 - accuracy: 0.
8247
Epoch 5/50
355/355 [==============================] - 9s 24ms/step - loss: 0.3829 - accuracy: 0.
8269
Epoch 6/50
355/355 [==============================] - 9s 25ms/step - loss: 0.3559 - accuracy: 0.
8391
Epoch 7/50
355/355 [==============================] - 7s 20ms/step - loss: 0.3482 - accuracy: 0.
8464
Epoch 8/50
355/355 [==============================] - 9s 24ms/step - loss: 0.3309 - accuracy: 0.
8591
Epoch 9/50
355/355 [==============================] - 9s 25ms/step - loss: 0.3120 - accuracy: 0.
8630
Epoch 10/50
355/355 [==============================] - 9s 24ms/step - loss: 0.3024 - accuracy: 0.
8701
Epoch 11/50
355/355 [==============================] - 9s 25ms/step - loss: 0.2753 - accuracy: 0.
8842
Epoch 12/50
355/355 [==============================] - 8s 22ms/step - loss: 0.2632 - accuracy: 0.
8943
Epoch 13/50
355/355 [==============================] - 6s 17ms/step - loss: 0.2401 - accuracy: 0.
9042
Epoch 14/50
355/355 [==============================] - 6s 17ms/step - loss: 0.2369 - accuracy: 0.
9030
Epoch 15/50
355/355 [==============================] - 6s 17ms/step - loss: 0.2137 - accuracy: 0.
9169
Epoch 16/50
355/355 [==============================] - 7s 20ms/step - loss: 0.2193 - accuracy: 0.
9101
Epoch 17/50
355/355 [==============================] - 6s 18ms/step - loss: 0.1817 - accuracy: 0.
9304
Epoch 18/50
355/355 [==============================] - 6s 17ms/step - loss: 0.1892 - accuracy: 0.
9278
Epoch 19/50
355/355 [==============================] - 6s 18ms/step - loss: 0.1669 - accuracy: 0.
9346
Epoch 20/50
355/355 [==============================] - 7s 20ms/step - loss: 0.1586 - accuracy: 0.
9360
```

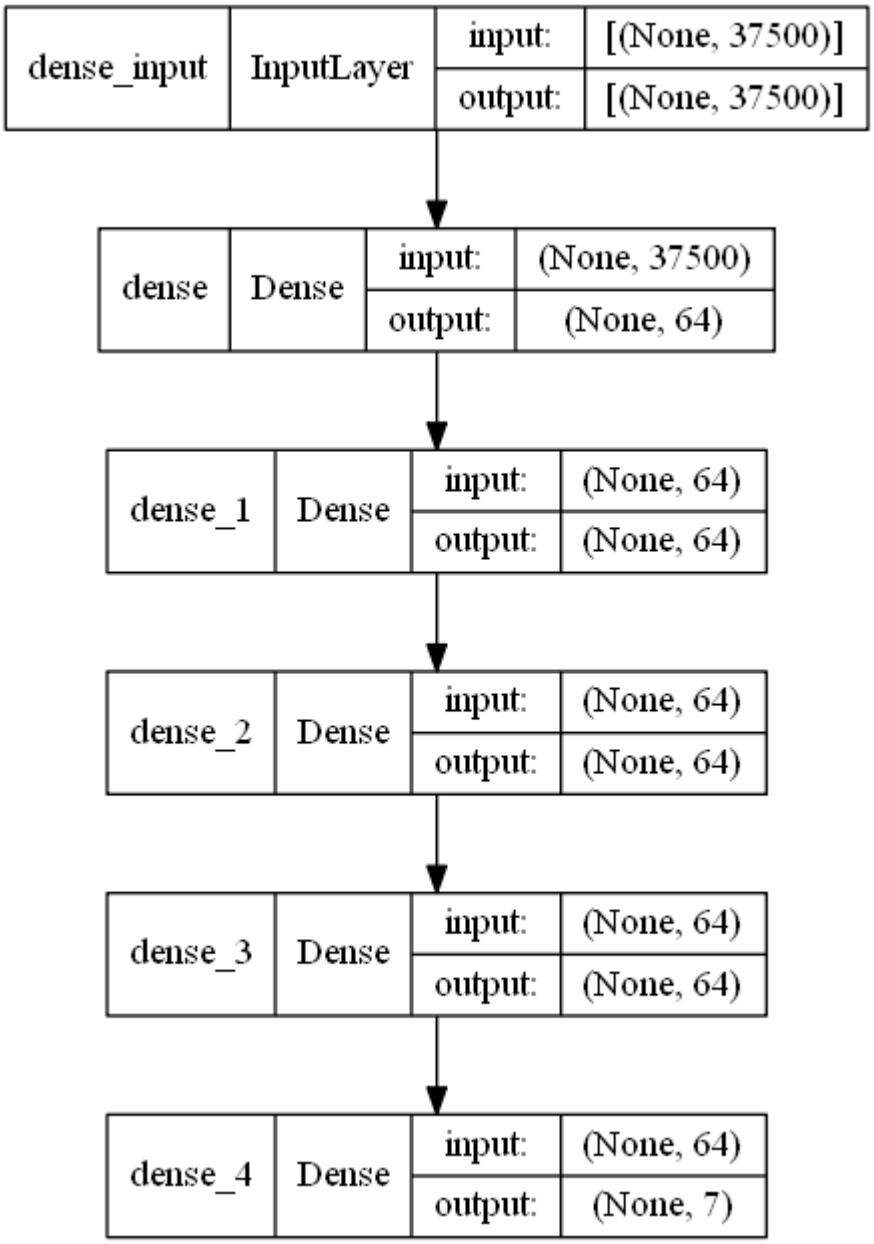
```
Epoch 21/50
355/355 [==============================] - 6s 16ms/step - loss: 0.1533 - accuracy: 0.
9394
Epoch 22/50
355/355 [==============================] - 6s 18ms/step - loss: 0.1379 - accuracy: 0.
9490
Epoch 23/50
355/355 [==============================] - 6s 17ms/step - loss: 0.1384 - accuracy: 0.
9467
Epoch 24/50
355/355 [==============================] - 6s 17ms/step - loss: 0.1369 - accuracy: 0.
9467
Epoch 25/50
355/355 [==============================] - 6s 17ms/step - loss: 0.1268 - accuracy: 0.
9512
Epoch 26/50
355/355 [==============================] - 6s 16ms/step - loss: 0.1239 - accuracy: 0.
9498
Epoch 27/50
355/355 [==============================] - 6s 17ms/step - loss: 0.1092 - accuracy: 0.
9583
Epoch 28/50
355/355 [==============================] - 6s 16ms/step - loss: 0.1105 - accuracy: 0.
9583
Epoch 29/50
355/355 [==============================] - 6s 17ms/step - loss: 0.0962 - accuracy: 0.
9636
Epoch 30/50
355/355 [==============================] - 6s 17ms/step - loss: 0.0931 - accuracy: 0.
9673
Epoch 31/50
355/355 [==============================] - 6s 18ms/step - loss: 0.0999 - accuracy: 0.
9636
Epoch 32/50
355/355 [==============================] - 7s 18ms/step - loss: 0.0807 - accuracy: 0.
9710
Epoch 33/50
355/355 [==============================] - 6s 17ms/step - loss: 0.0750 - accuracy: 0.
9690
Epoch 34/50
355/355 [==============================] - 6s 17ms/step - loss: 0.0884 - accuracy: 0.
9676
Epoch 35/50
355/355 [==============================] - 6s 17ms/step - loss: 0.0882 - accuracy: 0.
9713
Epoch 36/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0648 - accuracy: 0.
9741
Epoch 37/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0952 - accuracy: 0.
9696
Epoch 38/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0634 - accuracy: 0.
9766
Epoch 39/50
355/355 [==============================] - 5s 15ms/step - loss: 0.0766 - accuracy: 0.
9732
Epoch 40/50
355/355 [==============================] - 5s 15ms/step - loss: 0.0669 - accuracy: 0.
9760
```

```
Epoch 41/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0727 - accuracy: 0.
9797
Epoch 42/50
355/355 [==============================] - 5s 14ms/step - loss: 0.0797 - accuracy: 0.
9760
Epoch 43/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0624 - accuracy: 0.
9820
Epoch 44/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0586 - accuracy: 0.
9794
Epoch 45/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0720 - accuracy: 0.
9752
Epoch 46/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0561 - accuracy: 0.
9825
Epoch 47/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0596 - accuracy: 0.
9763
Epoch 48/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0568 - accuracy: 0.
9822
Epoch 49/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0632 - accuracy: 0.
9780 0s - loss: 0.0
Epoch 50/50
355/355 [==============================] - 6s 16ms/step - loss: 0.0777 - accuracy: 0.
9735
42/42 [==============================] - 1s 7ms/step - loss: 1.2581 - accuracy: 0.820
5
Test: accuracy =  82.05323219299316 %
```

In [59]:

```
from keras.utils.vis_utils import plot_model
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Out[59]:



CNN

CNN is ideal for image classification. It is better since CNN has features parameter sharing and dimensionality reduction. Because of parameter sharing in CNN, the number of parameters is reduced thus the computations get decreased.

Applied Data augmentation using ImageDatagenerator before model training

Since the data is less, we apply data augmentation using ImageDataGenerator.

ImageDataGenerator generates augmentation of images in real-time while the model is still training. One can apply any random transformations on each training image as it is passed to the model.

The CNN model is a repeated network of the following layers:

1. Convolutional
2. Pooling
3. Dropout
4. Flatten
5. Dense

Optimizer: Adam

Activation function used: Softmax

In [60]:
```
from tensorflow.keras.layers import Flatten,Dense,Dropout,BatchNormalization,Conv2D, M
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [61]:
```
# Set the CNN model
# my CNN architechture is In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*3 -> Flatte
input_shape = (100, 125, 3)
num_classes = 7

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding = 'Same',input_shape
model.add(Conv2D(32,kernel_size=(3, 3), activation='relu',padding = 'Same',))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.16))

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding = 'Same'))
model.add(Conv2D(32,kernel_size=(3, 3), activation='relu',padding = 'Same',))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.20))

model.add(Conv2D(64, (3, 3), activation='relu',padding = 'same'))
model.add(Conv2D(64, (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(Dense(128, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 100, 125, 32)      896

 conv2d_1 (Conv2D)           (None, 100, 125, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 50, 62, 32)       0
 )

 dropout (Dropout)           (None, 50, 62, 32)        0

 conv2d_2 (Conv2D)           (None, 50, 62, 32)        9248

 conv2d_3 (Conv2D)           (None, 50, 62, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 25, 31, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 25, 31, 32)        0

 conv2d_4 (Conv2D)           (None, 25, 31, 64)        18496

 conv2d_5 (Conv2D)           (None, 25, 31, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 12, 15, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 12, 15, 64)        0

 flatten (Flatten)           (None, 11520)             0

 dense_5 (Dense)             (None, 256)               2949376

 dense_6 (Dense)             (None, 128)               32896

 dropout_3 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 7)                 903

=================================================================
Total params: 3,067,239
Trainable params: 3,067,239
Non-trainable params: 0
_________________________________________________________________
```

In [62]:
```
# Define the optimizer
optimizer = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad
```

```
C:\Users\wajah\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:105: UserWarnin
g: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [63]:
```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["acc
```

In [64]:
```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
```

```
                                            patience=4,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

In [65]:
```
x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, test_siz
# Reshape image in 3 dimensions (height = 100, width = 125 , canal = 3)
x_train = x_train.reshape(x_train.shape[0], *(100, 125, 3))
x_test = x_test.reshape(x_test.shape[0], *(100, 125, 3))
x_validate = x_validate.reshape(x_validate.shape[0], *(100, 125, 3))
# With data augmentation to prevent overfitting

datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.12,  # randomly shift images horizontally (fraction of tot
        height_shift_range=0.12,  # randomly shift images vertically (fraction of tota
        horizontal_flip=True,  # randomly flip images
        vertical_flip=True)  # randomly flip images

datagen.fit(x_train)
```

In [66]:
```
# Fit the model
epochs = 60
batch_size = 16
history = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (x_validate,y_validat
                              verbose = 1, steps_per_epoch=x_train.shape[0] // batch_s
                              , callbacks=[learning_rate_reduction])

from tensorflow.keras.metrics import Recall
from sklearn.metrics import classification_report,confusion_matrix
```

```
C:\Users\wajah\AppData\Local\Temp/ipykernel_7072/87963123.py:4: UserWarning: `Model.f
it_generator` is deprecated and will be removed in a future version. Please use `Mode
l.fit`, which supports generators.
  history = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
```

```
Epoch 1/60
199/199 [==============================] - 126s 629ms/step - loss: 0.5866 - accuracy:
0.7919 - val_loss: 0.5652 - val_accuracy: 0.8113 - lr: 1.0000e-04
Epoch 2/60
199/199 [==============================] - 113s 568ms/step - loss: 0.5008 - accuracy:
0.8023 - val_loss: 0.4964 - val_accuracy: 0.8113 - lr: 1.0000e-04
Epoch 3/60
199/199 [==============================] - 113s 570ms/step - loss: 0.4727 - accuracy:
0.8017 - val_loss: 0.5108 - val_accuracy: 0.8113 - lr: 1.0000e-04
Epoch 4/60
199/199 [==============================] - 113s 569ms/step - loss: 0.4502 - accuracy:
0.8011 - val_loss: 0.4127 - val_accuracy: 0.8113 - lr: 1.0000e-04
Epoch 5/60
199/199 [==============================] - ETA: 0s - loss: 0.4285 - accuracy: 0.8061
Epoch 00005: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
199/199 [==============================] - 112s 565ms/step - loss: 0.4285 - accuracy:
0.8061 - val_loss: 0.4019 - val_accuracy: 0.8113 - lr: 1.0000e-04
Epoch 6/60
199/199 [==============================] - 113s 567ms/step - loss: 0.4099 - accuracy:
0.8067 - val_loss: 0.3872 - val_accuracy: 0.8085 - lr: 5.0000e-05
Epoch 7/60
199/199 [==============================] - 113s 569ms/step - loss: 0.4052 - accuracy:
0.8001 - val_loss: 0.3824 - val_accuracy: 0.8197 - lr: 5.0000e-05
Epoch 8/60
199/199 [==============================] - 113s 568ms/step - loss: 0.4018 - accuracy:
0.8111 - val_loss: 0.3852 - val_accuracy: 0.8085 - lr: 5.0000e-05
Epoch 9/60
199/199 [==============================] - 113s 568ms/step - loss: 0.3900 - accuracy:
0.8061 - val_loss: 0.3776 - val_accuracy: 0.8197 - lr: 5.0000e-05
Epoch 10/60
199/199 [==============================] - 113s 567ms/step - loss: 0.3901 - accuracy:
0.8048 - val_loss: 0.3829 - val_accuracy: 0.8085 - lr: 5.0000e-05
Epoch 11/60
199/199 [==============================] - ETA: 0s - loss: 0.3885 - accuracy: 0.8149
Epoch 00011: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
199/199 [==============================] - 113s 569ms/step - loss: 0.3885 - accuracy:
0.8149 - val_loss: 0.3723 - val_accuracy: 0.8169 - lr: 5.0000e-05
Epoch 12/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3902 - accuracy:
0.8130 - val_loss: 0.3709 - val_accuracy: 0.8225 - lr: 2.5000e-05
Epoch 13/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3796 - accuracy:
0.8118 - val_loss: 0.3708 - val_accuracy: 0.8169 - lr: 2.5000e-05
Epoch 14/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3820 - accuracy:
0.8137 - val_loss: 0.3743 - val_accuracy: 0.8141 - lr: 2.5000e-05
Epoch 15/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3781 - accuracy:
0.8171 - val_loss: 0.3644 - val_accuracy: 0.8366 - lr: 2.5000e-05
Epoch 16/60
199/199 [==============================] - 112s 561ms/step - loss: 0.3768 - accuracy:
0.8174 - val_loss: 0.3711 - val_accuracy: 0.8141 - lr: 2.5000e-05
Epoch 17/60
199/199 [==============================] - 112s 562ms/step - loss: 0.3777 - accuracy:
0.8174 - val_loss: 0.3614 - val_accuracy: 0.8338 - lr: 2.5000e-05
Epoch 18/60
199/199 [==============================] - 112s 565ms/step - loss: 0.3723 - accuracy:
0.8196 - val_loss: 0.3674 - val_accuracy: 0.8310 - lr: 2.5000e-05
Epoch 19/60
199/199 [==============================] - ETA: 0s - loss: 0.3727 - accuracy: 0.8149
```

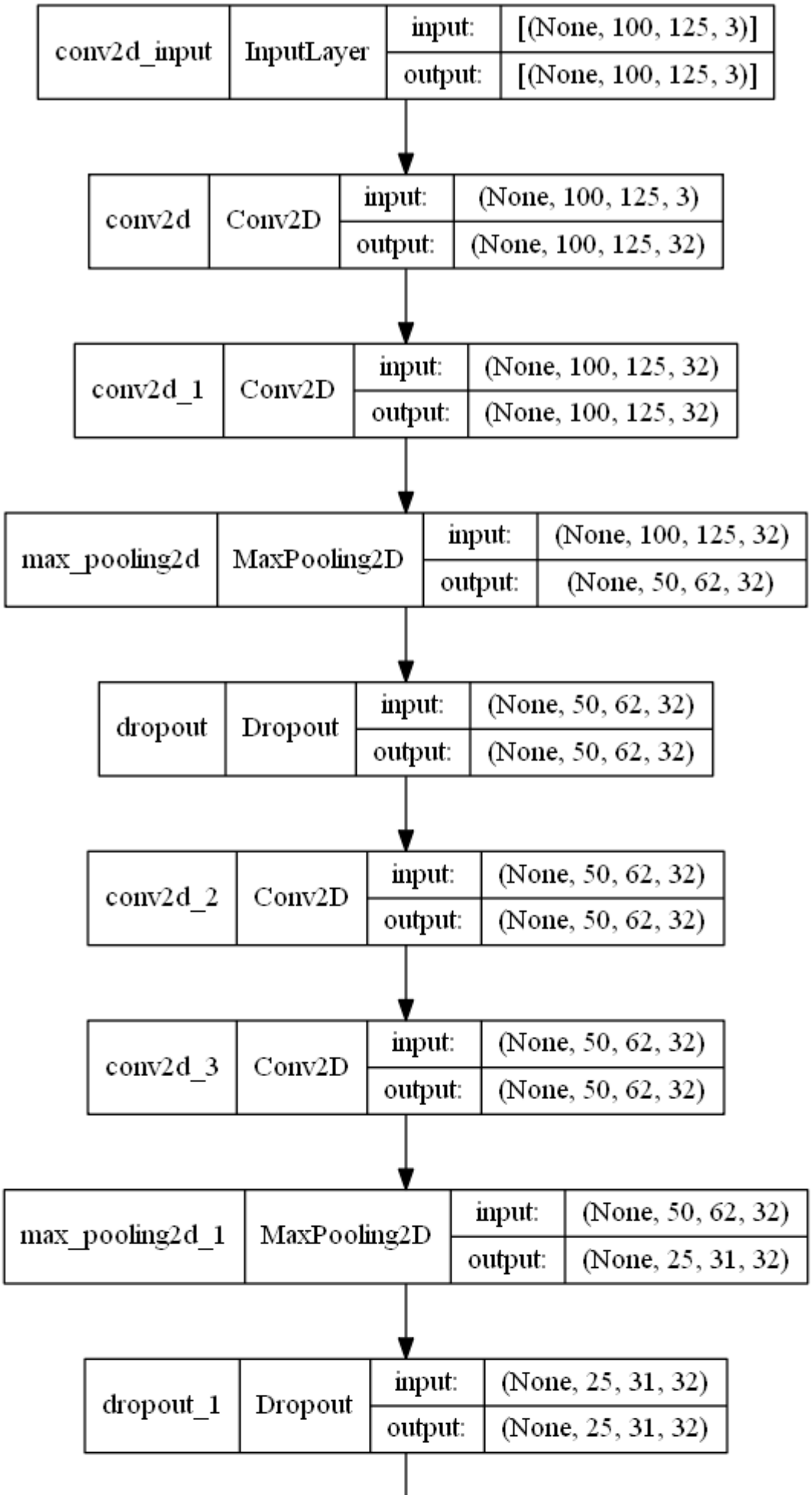
```
Epoch 00019: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
199/199 [==============================] - 112s 565ms/step - loss: 0.3727 - accuracy:
0.8149 - val_loss: 0.3679 - val_accuracy: 0.8338 - lr: 2.5000e-05
Epoch 20/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3721 - accuracy:
0.8155 - val_loss: 0.3587 - val_accuracy: 0.8394 - lr: 1.2500e-05
Epoch 21/60
199/199 [==============================] - 112s 562ms/step - loss: 0.3659 - accuracy:
0.8140 - val_loss: 0.3718 - val_accuracy: 0.8310 - lr: 1.2500e-05
Epoch 22/60
199/199 [==============================] - 112s 562ms/step - loss: 0.3657 - accuracy:
0.8171 - val_loss: 0.3630 - val_accuracy: 0.8310 - lr: 1.2500e-05
Epoch 23/60
199/199 [==============================] - 120s 604ms/step - loss: 0.3673 - accuracy:
0.8162 - val_loss: 0.3622 - val_accuracy: 0.8310 - lr: 1.2500e-05
Epoch 24/60
199/199 [==============================] - ETA: 0s - loss: 0.3641 - accuracy: 0.8209
Epoch 00024: ReduceLROnPlateau reducing learning rate to 1e-05.
199/199 [==============================] - 113s 565ms/step - loss: 0.3641 - accuracy:
0.8209 - val_loss: 0.3600 - val_accuracy: 0.8394 - lr: 1.2500e-05
Epoch 25/60
199/199 [==============================] - 112s 561ms/step - loss: 0.3598 - accuracy:
0.8294 - val_loss: 0.3596 - val_accuracy: 0.8366 - lr: 1.0000e-05
Epoch 26/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3692 - accuracy:
0.8196 - val_loss: 0.3582 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 27/60
199/199 [==============================] - 113s 565ms/step - loss: 0.3622 - accuracy:
0.8234 - val_loss: 0.3592 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 28/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3641 - accuracy:
0.8137 - val_loss: 0.3592 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 29/60
199/199 [==============================] - 113s 565ms/step - loss: 0.3622 - accuracy:
0.8234 - val_loss: 0.3700 - val_accuracy: 0.8338 - lr: 1.0000e-05
Epoch 30/60
199/199 [==============================] - 112s 562ms/step - loss: 0.3617 - accuracy:
0.8209 - val_loss: 0.3536 - val_accuracy: 0.8366 - lr: 1.0000e-05
Epoch 31/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3593 - accuracy:
0.8181 - val_loss: 0.3534 - val_accuracy: 0.8366 - lr: 1.0000e-05
Epoch 32/60
199/199 [==============================] - 112s 561ms/step - loss: 0.3559 - accuracy:
0.8228 - val_loss: 0.3563 - val_accuracy: 0.8338 - lr: 1.0000e-05
Epoch 33/60
199/199 [==============================] - 115s 578ms/step - loss: 0.3569 - accuracy:
0.8234 - val_loss: 0.3570 - val_accuracy: 0.8366 - lr: 1.0000e-05
Epoch 34/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3567 - accuracy:
0.8222 - val_loss: 0.3615 - val_accuracy: 0.8366 - lr: 1.0000e-05
Epoch 35/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3562 - accuracy:
0.8225 - val_loss: 0.3546 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 36/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3597 - accuracy:
0.8190 - val_loss: 0.3619 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 37/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3575 - accuracy:
0.8234 - val_loss: 0.3555 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 38/60
```

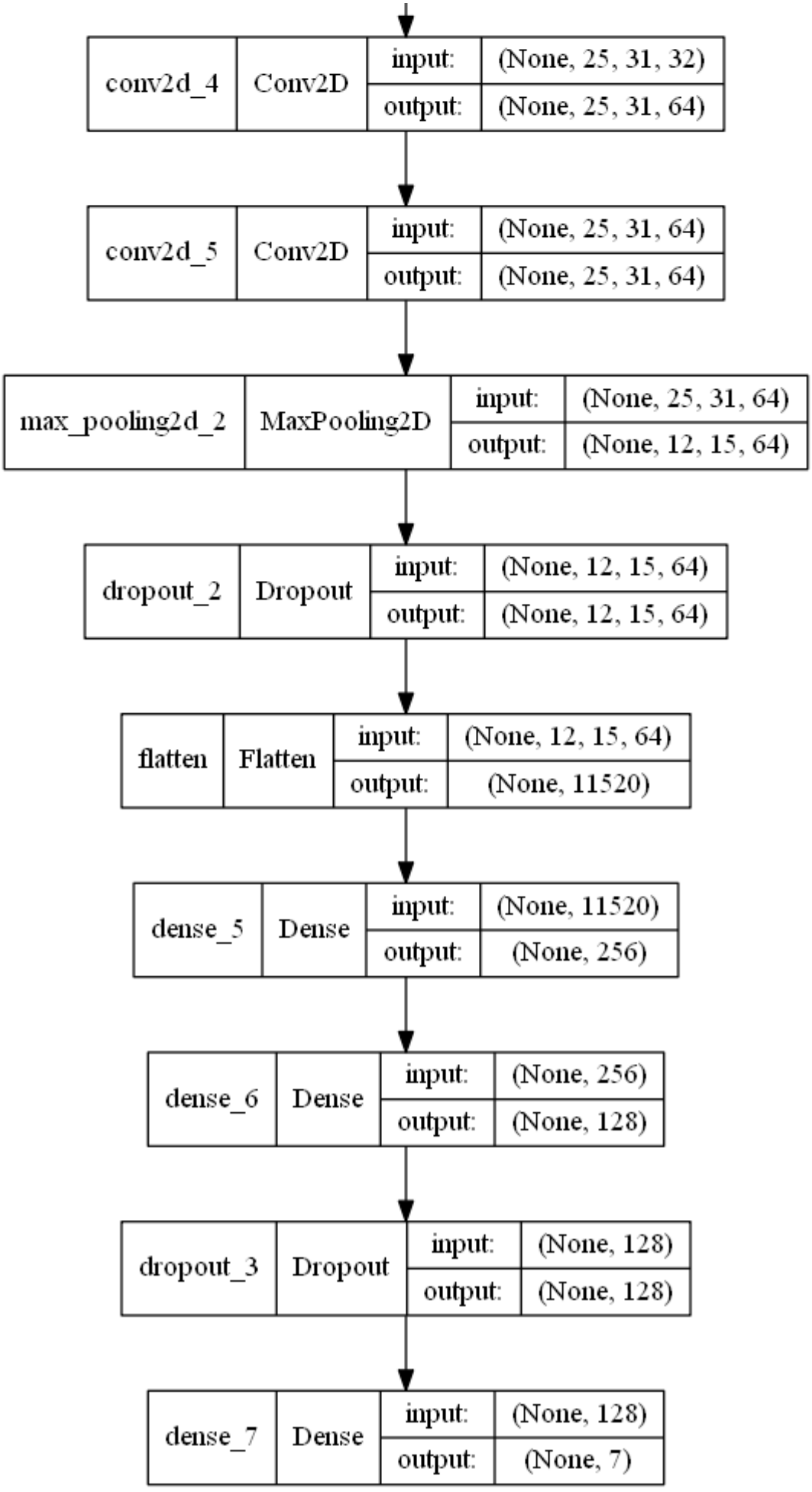
```
199/199 [==============================] - 112s 564ms/step - loss: 0.3538 - accuracy:
0.8291 - val_loss: 0.3603 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 39/60
199/199 [==============================] - 113s 566ms/step - loss: 0.3575 - accuracy:
0.8228 - val_loss: 0.3568 - val_accuracy: 0.8479 - lr: 1.0000e-05
Epoch 40/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3561 - accuracy:
0.8250 - val_loss: 0.3585 - val_accuracy: 0.8451 - lr: 1.0000e-05
Epoch 41/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3551 - accuracy:
0.8215 - val_loss: 0.3522 - val_accuracy: 0.8479 - lr: 1.0000e-05
Epoch 42/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3551 - accuracy:
0.8275 - val_loss: 0.3569 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 43/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3529 - accuracy:
0.8247 - val_loss: 0.3566 - val_accuracy: 0.8479 - lr: 1.0000e-05
Epoch 44/60
199/199 [==============================] - 112s 564ms/step - loss: 0.3560 - accuracy:
0.8253 - val_loss: 0.3608 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 45/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3550 - accuracy:
0.8237 - val_loss: 0.3556 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 46/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3488 - accuracy:
0.8341 - val_loss: 0.3608 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 47/60
199/199 [==============================] - 112s 561ms/step - loss: 0.3505 - accuracy:
0.8332 - val_loss: 0.3534 - val_accuracy: 0.8451 - lr: 1.0000e-05
Epoch 48/60
199/199 [==============================] - 111s 558ms/step - loss: 0.3505 - accuracy:
0.8250 - val_loss: 0.3497 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 49/60
199/199 [==============================] - 111s 560ms/step - loss: 0.3505 - accuracy:
0.8281 - val_loss: 0.3543 - val_accuracy: 0.8451 - lr: 1.0000e-05
Epoch 50/60
199/199 [==============================] - 111s 559ms/step - loss: 0.3452 - accuracy:
0.8341 - val_loss: 0.3667 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 51/60
199/199 [==============================] - 111s 559ms/step - loss: 0.3517 - accuracy:
0.8269 - val_loss: 0.3552 - val_accuracy: 0.8507 - lr: 1.0000e-05
Epoch 52/60
199/199 [==============================] - 111s 560ms/step - loss: 0.3481 - accuracy:
0.8332 - val_loss: 0.3563 - val_accuracy: 0.8479 - lr: 1.0000e-05
Epoch 53/60
199/199 [==============================] - 111s 559ms/step - loss: 0.3537 - accuracy:
0.8269 - val_loss: 0.3504 - val_accuracy: 0.8479 - lr: 1.0000e-05
Epoch 54/60
199/199 [==============================] - 112s 562ms/step - loss: 0.3514 - accuracy:
0.8291 - val_loss: 0.3519 - val_accuracy: 0.8394 - lr: 1.0000e-05
Epoch 55/60
199/199 [==============================] - 118s 590ms/step - loss: 0.3540 - accuracy:
0.8275 - val_loss: 0.3544 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 56/60
199/199 [==============================] - 113s 566ms/step - loss: 0.3508 - accuracy:
0.8288 - val_loss: 0.3583 - val_accuracy: 0.8423 - lr: 1.0000e-05
Epoch 57/60
199/199 [==============================] - 111s 559ms/step - loss: 0.3451 - accuracy:
0.8322 - val_loss: 0.3593 - val_accuracy: 0.8479 - lr: 1.0000e-05
Epoch 58/60
```

```
199/199 [==============================] - 112s 562ms/step - loss: 0.3522 - accuracy:
0.8218 - val_loss: 0.3578 - val_accuracy: 0.8451 - lr: 1.0000e-05
Epoch 59/60
199/199 [==============================] - 112s 563ms/step - loss: 0.3484 - accuracy:
0.8347 - val_loss: 0.3604 - val_accuracy: 0.8451 - lr: 1.0000e-05
Epoch 60/60
199/199 [==============================] - 111s 560ms/step - loss: 0.3521 - accuracy:
0.8303 - val_loss: 0.3562 - val_accuracy: 0.8451 - lr: 1.0000e-05
```

In [67]:
```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Out[67]:


conv2d_input
InputLayer
input: [(None, 100, 125, 3)]
output: [(None, 100, 125, 3)]
conv2d
Conv2D
input: (None, 100, 125, 3)
output: (None, 100, 125, 32)
conv2d_1
Conv2D
input: (None, 100, 125, 32)
output: (None, 100, 125, 32)
max_pooling2d
MaxPooling2D
input: (None, 100, 125, 32)
output: (None, 50, 62, 32)
dropout
Dropout
input: (None, 50, 62, 32)
output: (None, 50, 62, 32)
conv2d_2
Conv2D
input: (None, 50, 62, 32)
output: (None, 50, 62, 32)
conv2d_3
Conv2D
input: (None, 50, 62, 32)
output: (None, 50, 62, 32)
max_pooling2d_1
MaxPooling2D
input: (None, 50, 62, 32)
output: (None, 25, 31, 32)
dropout_1
Dropout
input: (None, 25, 31, 32)
output: (None, 25, 31, 32)

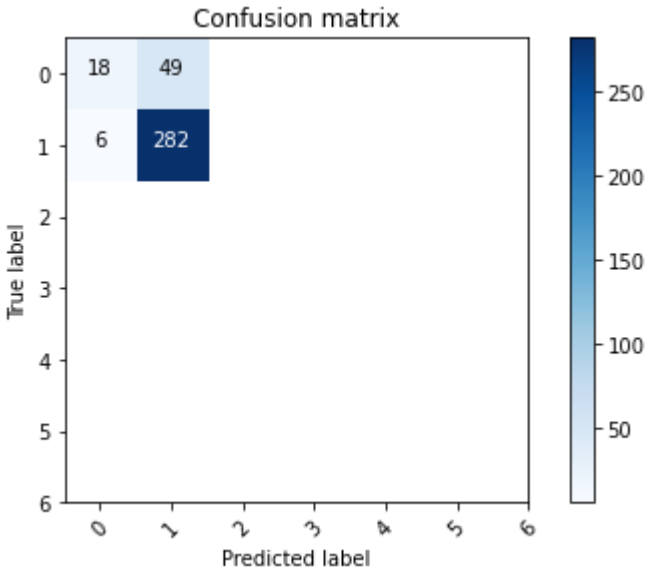

conv2d_4 Conv2D input: (None, 25, 31, 32) output: (None, 25, 31, 64)
conv2d_5 Conv2D input: (None, 25, 31, 64) output: (None, 25, 31, 64)
max_pooling2d_2 MaxPooling2D input: (None, 25, 31, 64) output: (None, 12, 15, 64)
dropout_2 Dropout input: (None, 12, 15, 64) output: (None, 12, 15, 64)
flatten Flatten input: (None, 12, 15, 64) output: (None, 11520)
dense_5 Dense input: (None, 11520) output: (None, 256)
dense_6 Dense input: (None, 256) output: (None, 128)
dropout_3 Dropout input: (None, 128) output: (None, 128)
dense_7 Dense input: (None, 128) output: (None, 7)

The CNN model has been visualised in the model attached above.

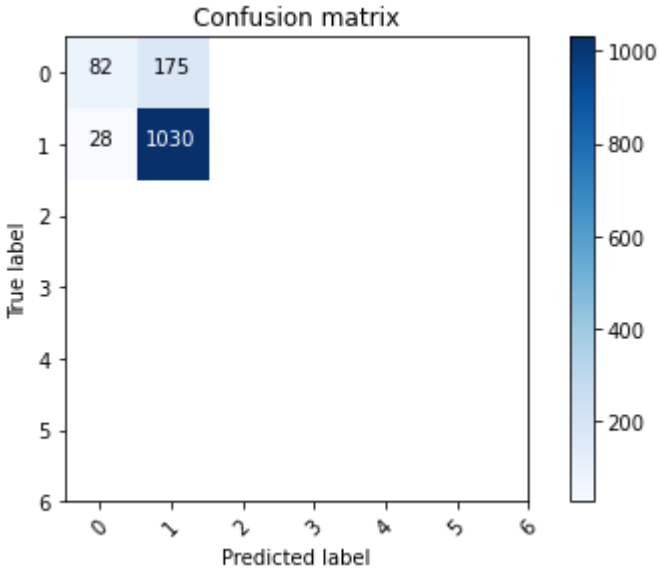
In [68]:
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
loss_v, accuracy_v = model.evaluate(x_validate, y_validate, verbose=1)
print("Validation: accuracy = %f  ;  loss_v = %f" % (accuracy_v, loss_v))
print("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))
model.save("model.h5")
```

```
42/42 [==============================] - 10s 231ms/step - loss: 0.3513 - accuracy: 0.
8456
12/12 [==============================] - 3s 222ms/step - loss: 0.3562 - accuracy: 0.8
451
Validation: accuracy = 0.845070  ;  loss_v = 0.356162
Test: accuracy = 0.845627  ;  loss = 0.351293
```

In [69]:
```
import itertools
# Function to plot confusion matrix
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`.
    """
    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)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

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

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

# Predict the values from the validation dataset
Y_pred = model.predict(x_validate)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_validate,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)



# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(7))
```

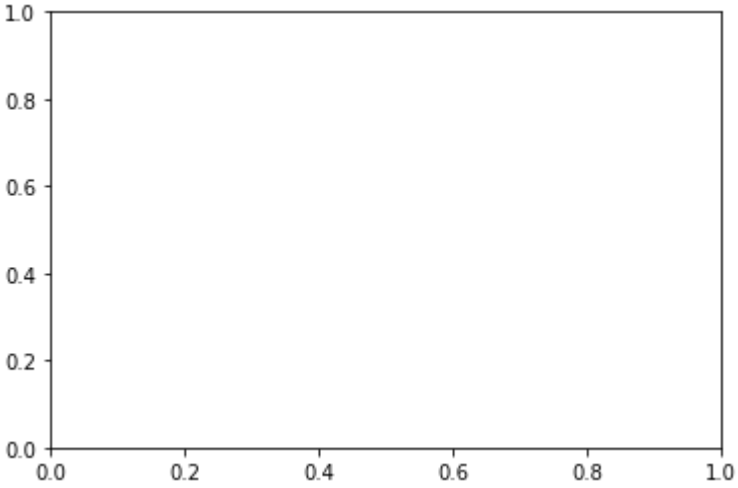


In [70]:
```
# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)



# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(7))
```



In [71]:
```
label_frac_error = 1 - np.diag(confusion_mtx) / np.sum(confusion_mtx, axis=1)
plt.bar(np.arange(7),label_frac_error)
plt.xlabel('True Label')
plt.ylabel('Fraction classified incorrectly')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7072/1127229655.py in <module>
      1 label_frac_error = 1 - np.diag(confusion_mtx) / np.sum(confusion_mtx, axis=1)
----> 2 plt.bar(np.arange(7),label_frac_error)
      3 plt.xlabel('True Label')
      4 plt.ylabel('Fraction classified incorrectly')

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in bar(x, height, width, bottom, a
lign, data, **kwargs)
   2385         x, height, width=0.8, bottom=None, *, align='center',
   2386         data=None, **kwargs):
-> 2387     return gca().bar(
   2388         x, height, width=width, bottom=bottom, align=align,
   2389         **({"data": data} if data is not None else {}), **kwargs)

~\anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, data, *args, **kwar
gs)
   1410     def inner(ax, *args, data=None, **kwargs):
   1411         if data is None:
-> 1412             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1413
   1414         bound = new_sig.bind(ax, *args, **kwargs)

~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py in bar(self, x, height, width,
bottom, align, **kwargs)
   2340                 yerr = self._convert_dx(yerr, y0, y, self.convert_yunits)
   2341
-> 2342         x, height, width, y, linewidth, hatch = np.broadcast_arrays(
   2343             # Make args iterable too.
   2344             np.atleast_1d(x), height, width, y, linewidth, hatch)

<__array_function__ internals> in broadcast_arrays(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\lib\stride_tricks.py in broadcast_arrays(subok, *
args)
    536     args = [np.array(_m, copy=False, subok=subok) for _m in args]
    537
--> 538     shape = _broadcast_shape(*args)
    539
    540     if all(array.shape == shape for array in args):

~\anaconda3\lib\site-packages\numpy\lib\stride_tricks.py in _broadcast_shape(*args)
    418     # use the old-iterator because np.nditer does not handle size 0 arrays
    419     # consistently
--> 420     b = np.broadcast(*args[:32])
    421     # unfortunately, it cannot handle 32 or more arguments directly
    422     for pos in range(32, len(args), 31):

ValueError: shape mismatch: objects cannot be broadcast to a single shape
```



In [72]:
```
# # Function to plot model's validation loss and validation accuracy
# def plot_model_history(model_history):
#     fig, axs = plt.subplots(1,2,figsize=(15,5))
#     # summarize history for accuracy
#     axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.hist
#     axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.
#     axs[0].set_title('Model Accuracy')
#     axs[0].set_ylabel('Accuracy')
#     axs[0].set_xlabel('Epoch')
#     axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(mode
#     axs[0].legend(['train', 'val'], loc='best')
#     # summarize history for loss
#     axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history[
#     axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.hist
#     axs[1].set_title('Model Loss')
#     axs[1].set_ylabel('Loss')
#     axs[1].set_xlabel('Epoch')
#     axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_hi
#     axs[1].legend(['train', 'val'], loc='best')
#     plt.show()
# plot_model_history(history)
```

Tranfer Learning

Due to lack of dataset, pretrained model of MobileNet is used.

Why MobileNet?

MobileNet significantly reduces the number of parameters when compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks.

The 2 layers in addition to the ones used for CNN are: Batch Normalization Zero Padding

In [79]:
```
import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoi
```

In [80]:
```
df1['image'] = df1['path'].map(lambda x: np.asarray(Image.open(x).resize((450,600))))
```

```
C:\Users\wajah\AppData\Local\Temp/ipykernel_7072/3252559109.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df1['image'] = df1['path'].map(lambda x: np.asarray(Image.open(x).resize((450,60
0))))
```

In [83]:
```
features=df1.drop(columns=['is_benign'],axis=1)
target=df1['is_benign']
```

In [84]:
```
x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(features, target, test_siz
tf.unique(x_train_o.cell_type.values)
```

Out[84]:
```
Unique(y=<tf.Tensor: shape=(7,), dtype=string, numpy=
array([b'Melanocytic nevi', b'Basal cell carcinoma',
       b'Benign keratosis-like lesions ', b'Melanoma',
       b'Actinic keratoses', b'Dermatofibroma', b'Vascular lesions'],
      dtype=object)>, idx=<tf.Tensor: shape=(3943,), dtype=int32, numpy=array([0, 0,
0, ..., 0, 0, 0])>)
```

In [85]:
```
x_train = np.asarray(x_train_o['image'].tolist())
x_test = np.asarray(x_test_o['image'].tolist())

x_train_mean = np.mean(x_train)
x_train_std = np.std(x_train)

x_test_mean = np.mean(x_test)
x_test_std = np.std(x_test)

x_train = (x_train - x_train_mean)/x_train_std
x_test = (x_test - x_test_mean)/x_test_std
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7072/2939144667.py in <module>
      3
      4 x_train_mean = np.mean(x_train)
----> 5 x_train_std = np.std(x_train)
      6
      7 x_test_mean = np.mean(x_test)

<__array_function__ internals> in std(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\core\fromnumeric.py in std(a, axis, dtype, out, d
dof, keepdims, where)
   3579             return std(axis=axis, dtype=dtype, out=out, ddof=ddof, **kwargs)
   3580
-> 3581     return _methods._std(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
   3582                          **kwargs)
   3583

~\anaconda3\lib\site-packages\numpy\core\_methods.py in _std(a, axis, dtype, out, ddo
f, keepdims, where)
    260 def _std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=False, *,
    261          where=True):
--> 262     ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
    263                keepdims=keepdims, where=where)
    264

~\anaconda3\lib\site-packages\numpy\core\_methods.py in _var(a, axis, dtype, out, ddo
f, keepdims, where)
    228     # Note that x may not be inexact and that we need it to be an array,
    229     # not a scalar.
--> 230     x = asanyarray(arr - arrmean)
    231
    232     if issubclass(arr.dtype.type, (nt.floating, nt.integer)):

MemoryError: Unable to allocate 23.8 GiB for an array with shape (3943, 600, 450, 3)
 and data type float64
```

In [86]:
```
# Perform one-hot encoding on the labels
y_train = to_categorical(y_train_o, num_classes = 7)
y_test = to_categorical(y_test_o, num_classes = 7)
y_test
```

Out[86]:
```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

In [87]:
```
x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, test_siz
# Reshape image in 3 dimensions (height = 100, width = 125 , canal = 3)
x_train = x_train.reshape(x_train.shape[0], *(224, 224, 3))
x_test = x_test.reshape(x_test.shape[0], *(224, 224, 3))
x_validate = x_validate.reshape(x_validate.shape[0], *(224, 224, 3))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7072/2454700661.py in <module>
      1 x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train,
test_size = 0.1, random_state = 999)
      2 # Reshape image in 3 dimensions (height = 100, width = 125 , canal = 3)
----> 3 x_train = x_train.reshape(x_train.shape[0], *(224, 224, 3))
      4 x_test = x_test.reshape(x_test.shape[0], *(224, 224, 3))
      5 x_validate = x_validate.reshape(x_validate.shape[0], *(224, 224, 3))

ValueError: cannot reshape array of size 2873880000 into shape (3548,224,224,3)
```

In [88]:
```
print(x_train.shape)
```

```
(3548, 600, 450, 3)
```

In [89]:
```
# create a copy of a mobilenet model

mobile = tensorflow.keras.applications.mobilenet.MobileNet()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mo
bilenet/mobilenet_1_0_224_tf.h5
17227776/17225924 [==============================] - 19s 1us/step
17235968/17225924 [==============================] - 19s 1us/step
```

In [ ]:
```
mobile.summary()
```

In [ ]:
```
def change_model(model, new_input_shape=(None, 40, 40, 3),custom_objects=None):
    # replace input shape of first layer

    config = model.layers[0].get_config()
    config['batch_input_shape']=new_input_shape
    model._layers[0]=model.layers[0].from_config(config)

    # rebuild model architecture by exporting and importing via json
    new_model = tensorflow.keras.models.model_from_json(model.to_json(),custom_objects

    # copy weights from old model to new one
    for layer in new_model._layers:
        try:
            layer.set_weights(model.get_layer(name=layer.name).get_weights())
            print("Loaded layer {}".format(layer.name))
        except:
            print("Could not transfer weights for layer {}".format(layer.name))

    return new_model
```

In [ ]:
```
new_model = change_model(mobile, new_input_shape=[None] + [100,125,3])
new_model.summary()
```

In [ ]:
```
# CREATE THE MODEL ARCHITECTURE

# Exclude the last 5 layers of the above model.
# This will include all layers up to and including global_average_pooling2d_1
x = new_model.layers[-6].output

# Create a new dense layer for predictions
# 7 corresponds to the number of classes
x = Dropout(0.25)(x)
```

```
predictions = Dense(7, activation='softmax')(x)

# inputs=mobile.input selects the input layer, outputs=predictions refers to the
# dense layer we created above.

model = Model(inputs=new_model.input, outputs=predictions)
```

In [ ]:
```
# We need to choose how many layers we actually want to be trained.

# Here we are freezing the weights of all layers except the
# last 23 layers in the new model.
# The last 23 layers of the model will be trained.

for layer in model.layers[:-23]:
    layer.trainable = False
```

In [ ]:
```
# Define Top2 and Top3 Accuracy

from tensorflow.keras.metrics import categorical_accuracy, top_k_categorical_accuracy

def top_3_accuracy(y_true, y_pred):
    return top_k_categorical_accuracy(y_true, y_pred, k=3)

def top_2_accuracy(y_true, y_pred):
    return top_k_categorical_accuracy(y_true, y_pred, k=2)
```

In [ ]:
```
model.compile(Adam(lr=0.01), loss='categorical_crossentropy',
              metrics=[categorical_accuracy, top_2_accuracy, top_3_accuracy])
```

In [ ]:
```
# Add weights to try to make the model more sensitive to melanoma

class_weights={
    0: 1.0, # akiec
    1: 1.0, # bcc
    2: 1.0, # bkl
    3: 1.0, # df
    4: 3.0, # mel # Try to make the model more sensitive to Melanoma.
    5: 1.0, # nv
    6: 1.0, # vasc
}
```

In [ ]:
```
filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_top_3_accuracy', verbose=1,
                             save_best_only=True, mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_top_3_accuracy', factor=0.5, patience=2,
                                    verbose=1, mode='max', min_lr=0.00001)


callbacks_list = [checkpoint, reduce_lr]

history = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              class_weight=class_weights,
                    validation_data=(x_validate,y_validate),steps_per_epoch=x_train.sh
                    epochs=10, verbose=1,
                   callbacks=callbacks_list)
```

In [ ]:
```
from keras.utils.vis_utils import plot_model
```

```
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

In [ ]:
```
# get the metric names so we can use evaulate_generator
model.metrics_names
```

In [ ]:
```
# Here the the last epoch will be used.

val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = \
model.evaluate(datagen.flow(x_test,y_test, batch_size=16) )

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)
print('val_top_2_acc:', val_top_2_acc)
print('val_top_3_acc:', val_top_3_acc)
```

In [ ]:
```
# Here the best epoch will be used.

model.load_weights('model.h5')

val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = \
model.evaluate_generator(datagen.flow(x_test,y_test, batch_size=16)
                         )

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)
print('val_top_2_acc:', val_top_2_acc)
print('val_top_3_acc:', val_top_3_acc)
```

Plot the Training Curves

In [ ]:
```
# display the loss and accuracy curves

import matplotlib.pyplot as plt

acc = history.history['categorical_accuracy']
val_acc = history.history['val_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
train_top2_acc = history.history['top_2_accuracy']
val_top2_acc = history.history['val_top_2_accuracy']
train_top3_acc = history.history['top_3_accuracy']
val_top3_acc = history.history['val_top_3_accuracy']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()
plt.figure()
```

```
plt.plot(epochs, train_top2_acc, 'bo', label='Training top2 acc')
plt.plot(epochs, val_top2_acc, 'b', label='Validation top2 acc')
plt.title('Training and validation top2 accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, train_top3_acc, 'bo', label='Training top3 acc')
plt.plot(epochs, val_top3_acc, 'b', label='Validation top3 acc')
plt.title('Training and validation top3 accuracy')
plt.legend()


plt.show()
```

In [ ]:
```
accuracy = model.evaluate(x_test, y_test,verbose=1)[1]
accuracy_v = model.evaluate(x_validate, y_validate)[1]
print("Validation: accuracy = ", accuracy_v)
print("Test: accuracy = ",accuracy)
model.save("model.h5")
```

Create a Confusion Matrix

In [ ]:
```
# make a prediction
predictions = model.predict_generator(datagen.flow(x_test,y_test, batch_size=16), verb
```

In [ ]:
```
predictions.shape
```

In [ ]:
```
test_batches = datagen.flow(x_test,y_test, batch_size=16)
test_batches
```

In [ ]:
```
# Source: Scikit Learn website
# http://scikit-learn.org/stable/auto_examples/
# model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-
# selection-plot-confusion-matrix-py


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.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 [ ]:

```
# Function to plot confusion matrix
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`.
    """
    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)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

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

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

# Predict the values from the validation dataset
Y_pred = model.predict(x_validate)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_validate,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)



# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(7))
```

In [ ]:

```
# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
```

```
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)



# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(7))
```

Generate the Classification Report

In [ ]:
```
y_pred = model.predict(x_test)
y_pred =y_pred>0.5
```

In [ ]:
```
cm_plot_labels = ['Actinic Keratoses', 'Basal cell carcinoma', 'Benign Keratosis-like
```

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

# Generate a classification report
report = classification_report(y_test, y_pred, target_names=cm_plot_labels)

print(report)
```

Recall = Given a class, will the classifier be able to detect it?
Precision = Given a class prediction from a classifier, how likely is it to be correct?
F1 Score = The harmonic mean of the recall and precision. Essentially, it punishes extreme values.

In [ ]:
```
model.save("lesion_model.h5")
```

XAI

Interpretation of results of model using Feature Importance techniques

The increasing trend in the use of machine learning for critical applications such as medical diagnosis suggests an imperative need for methodologies that can help to understand and evaluate the predictions of machine-learning models.

There are two main ways to look at a classification or a regression model:

1. inspect model parameters and try to figure out how the model works globally;
2. inspect an individual prediction of a model, try to figure out why the model makes the decision it makes.

For example, LIME, or Local Interpretable Model-Agnostic Explanations, is an algorithm that can explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model.-

Techniques applied: LIME, PDP, SHAP, etc.

In [ ]:
```
tile_df = df.copy()
tile_df
```

In [ ]:
```
tile_df.drop('lesion_id', inplace=True, axis=1)
tile_df.drop('image_id', inplace=True, axis=1)
tile_df.drop('cell_type', inplace=True, axis=1)
tile_df.drop('path', inplace=True, axis=1)
tile_df.drop('dx', inplace=True, axis=1)
tile_df.head()
```

In [ ]:
```
X = tile_df.drop(['cell_type_idx'],axis=1).values
y = tile_df['cell_type_idx'].values
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)
```

In [ ]:
```
!pip install alibi
```

In [ ]:
```
pip install shap
```

In [ ]:
```
import shap
shap.initjs()

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from alibi.explainers import KernelShap
from scipy.special import logit
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

As the dataframe consisted of a few features in the form of strings, we have processed these features by localization one hot method.

In [ ]:
```
tile_df['localization_onehot'] = tile_df.localization.map({'scalp':0, 'ear':1, 'face':
       'upper extremity':7, 'abdomen':8, 'lower extremity':9,
       'genital':10, 'hand':11, 'foot':12, 'acral':13, 'unknown':14})
tile_df.head()
```

In [ ]:
```
tile_df['dx_type_onehot'] = tile_df.dx_type.map({'confocal':0,'consensus':1,'follow_up
tile_df.head()
```

In [ ]:
```
tile_df['gender_male'] = tile_df.sex.map({'female':0, 'male':1, 'unknown':2})
tile_df.head()
```

In [ ]:
```
tile_df.columns
```

In [ ]:
```
features = ['age',  'localization_onehot', 'dx_type_onehot','gender_male']
```

In [ ]:
```
X = tile_df[features]
y = tile_df['cell_type_idx'].values
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)
```

In [ ]:
```
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
```

Creating a XGB Classifier model for the prediction of the type of skin disease using a tabular dataset:

In [ ]:
```
model = XGBClassifier(random_state=1)
model = model.fit(X_train, y_train)
```

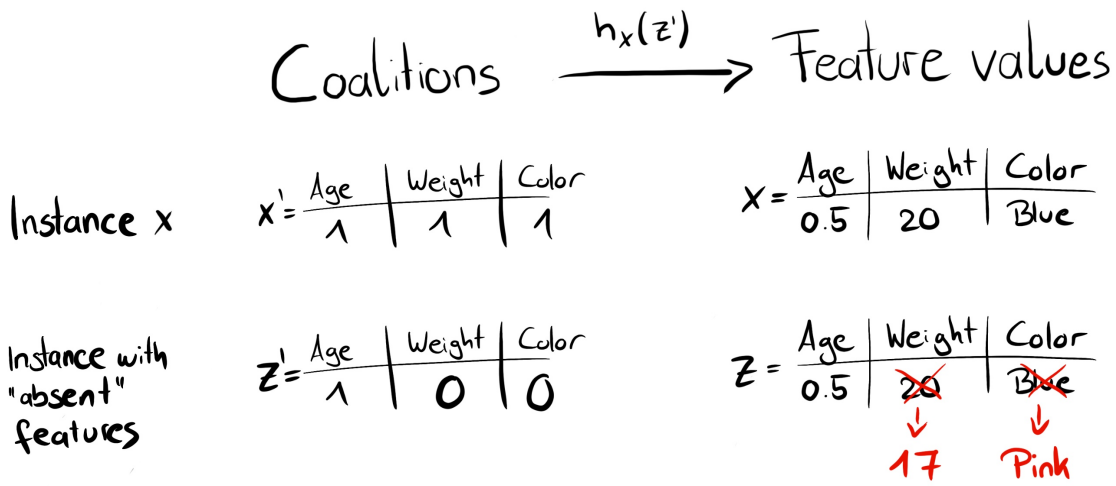
In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
predictions = [round(value) for value in y_pred]
```

In [ ]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Shap:

The goal of SHAP is to explain the prediction of an instance x by computing the contribution of each feature to the prediction. The SHAP explanation method computes Shapley values from coalitional theory.



In [ ]:
```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
```

In [ ]:
```
print('Expected Value: ', explainer.expected_value)
```

In [ ]:
```
shap.summary_plot(shap_values, X_test, plot_type="bar")
```

The feature importance plot, other than showing us which features are the most important, also show us which feature plays a major role in the prediction of which class. For example, we observe that the localization plays a major role in predicting whether the disease belongs to

class 3. Similarly, dx_type_onehot plays a major role in predicting whether the disease belongs to class 1, 5, etc.

In [ ]:
```
shap.summary_plot(shap_values[0], X_test)
```

From this summary plot, we observe the effect of the different values of each feature on the SHAP value, which is used for prediction. For example, we see that the higher values of dx_type_onehot (follow up and histopathology) are associated with a higher SHAP value.

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

## Preprocess training and test target (y) after having performed train-test split
le = LabelEncoder()
y_multi_train = pd.Series(le.fit_transform(y_train))
y_multi_test = pd.Series(le.transform(y_test))

## Check classes
le.classes_
```

In [ ]:
```
shap.initjs()
shap.dependence_plot('dx_type_onehot', interaction_index='age',
                     shap_values=shap_values[0],
                     features=X_test,
                     display_features=X_test)
```

In this graph, the effect of dx_type_onehot at specific ages is seen on the SHAP value. In congruence to what we observed in the earlier graph, the higher the value for dx_type_onehot (expert consensus or confirmation by microscopy)

In [ ]:
```
shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0][:100,:], X_test.iloc[:100,
```

In [ ]:
```
shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0][15,:], X_test.iloc[15,:])
```

Here, the force plot shows a local instance of the dataset. The localisation and type of diagnosis impact the prediction positively due to their positive SHAP value. On the other hand, the age and gender of the patient impacts the model negatively.

Feature Importance:

Feature importance measures the increase in the prediction error of the model after we permuted the feature values.

A feature is "important" if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.

A feature is "unimportant" if shuffling its values leaves the model error unchanged, because in this case the model ignored the feature for the prediction.

In [ ]:
```
pip install eli5
```

In [ ]:
```
import eli5
from eli5.sklearn import PermutationImportance
```

In [ ]:
```
eli5.show_weights(model.get_booster(), top=15)
```

We observe that the way the skin disease was diagnosed is extremely important to the prediction of the type of disease. Age as well as the localization of the disease (area of the body the disease is present on) also play major roles in the prediction. However, we observe that the gender of the patient does not play a major part in the prediction of the model. This shows us that the probability of having a specific disease does not change based on the fact that the patient is male or female, as the innate human biology is the same for either of the sexes.

In [ ]:
```
tgt = 6
print('Reference:', y_test[tgt])
print('Predicted:', predictions[tgt])
eli5.show_prediction(model.get_booster(), X_test.iloc[tgt],
                     feature_names=features, show_feature_values=True)
```

Here, we have taken a local instance where we observe that the prediction has been made based off of the type of diagnosis and age.

PDP :

The partial dependence plot shows the marginal effect one or two features have on the predicted outcome of a machine learning model.

A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic or more complex.

For each of the categories, we get a PDP estimate by forcing all data instances to have the same category.

In [ ]:
```
pip install pdpbox
```

In [ ]:
```
from pdpbox import pdp, get_dataset, info_plots
```

In [ ]:
```
pdp_feat_67_rf = pdp.pdp_isolate(model=model,
                                 dataset=X_train,
                                 model_features=features,
                                 feature='dx_type_onehot')
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_feat_67_rf,
                         feature_name='type of diagnosis',
                         center=True,
                         x_quantile=True,
                         ncols=3,
                         plot_lines=True,
                         frac_to_plot=100)
```

The PDP (Partial Dependence Plot) shows us the relation between an increase/decrease of one feature to the prediction of the model.

For example: In figure 1 (class 0), we observe that the chances of the skin disease belonging to class 0 increases when the value of dx_type_onehot changes from 2 (follow up) to 3 (histopathology).

Similarly, in figure 5 (class 4), we observe that the probability of the skin disease belonging to class 4 is extremely high when the value of dx_type_onehot lies between 0 and 2, and decreases comparatively when it lies between 2 and 3.

Similarly, probability of the skin disease belonging to class 6 is extremely low when the value of dx_type_onehot lies between 0 and 2 (confocal, consensus and follow up), and increases comparatively when it changes from 2 to 3.

LIME

LIME is a technique that explains how the input features of a machine learning model affect its predictions. For instance, for image classification tasks, LIME finds the region of an image (set of super-pixels) with the strongest association with a prediction label.

LIME creates explanations by generating a new dataset of random perturbations (with their respective predictions) around the instance being explained and then fitting a weighted local surrogate model - model that gives explanation of individual predictions.

Step 1: Generate random perturbations for input image

Step 2: Predict class for perturbations

Step 3: Compute weights (importance) for the perturbations

Step 4: Fit a explainable linear model using the perturbations, predictions and weights

In [ ]:
```
import skimage.io
import skimage.segmentation
```

In [ ]:
```
np.random.seed(222)

Xi = x_test[3]
preds = model.predict(Xi[np.newaxis,:,:,:])
top_pred_classes = preds[0].argsort()[-5:][::-1] # Save ids of top 5 classes
top_pred_classes
```

In [ ]:
```
print(y_test[3])
```

In [ ]:
```
skimage.io.imshow(Xi)
```

In [ ]:
```
#Generate segmentation for image
```

```
superpixels = skimage.segmentation.quickshift(Xi, kernel_size=4,max_dist=200, ratio=0.
num_superpixels = np.unique(superpixels).shape[0]
skimage.io.imshow(skimage.segmentation.mark_boundaries(Xi, superpixels))
print("The number of super pixels generated")
num_superpixels
```

In [ ]:
```
#Generate perturbations
num_perturb = 150
perturbations = np.random.binomial(1, 0.5, size=(num_perturb, num_superpixels))

#Create function to apply perturbations to images
import copy
def perturb_image(img,perturbation,segments):
  active_pixels = np.where(perturbation == 1)[0]
  mask = np.zeros(segments.shape)
  for active in active_pixels:
      mask[segments == active] = 1
  perturbed_image = copy.deepcopy(img)
  perturbed_image = perturbed_image*mask[:,:,np.newaxis]
  return perturbed_image

#Show example of perturbations
print(perturbations[0])
```

In [ ]:
```
predictions = []
for pert in perturbations:
  perturbed_img = perturb_image(Xi,pert,superpixels)
  pred = model.predict(perturbed_img[np.newaxis,:,:,:])
  predictions.append(pred)

predictions = np.array(predictions)
print(predictions.shape)
```

In [ ]:
```
skimage.io.imshow(perturb_image(Xi,perturbations[0],superpixels))
```

In [ ]:
```
skimage.io.imshow(perturb_image(Xi,perturbations[11],superpixels))
```

In [ ]:
```
skimage.io.imshow(perturb_image(Xi,perturbations[2],superpixels))
```

In [ ]:
```
#Compute distances to original image
import sklearn.metrics
original_image = np.ones(num_superpixels)[np.newaxis,:] #Perturbation with all superpi
distances = sklearn.metrics.pairwise_distances(perturbations,original_image, metric='c
print(distances.shape)

#Transform distances to a value between 0 an 1 (weights) using a kernel function
kernel_width = 0.25
weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2)) #Kernel function
print(weights.shape)
```

In [ ]:
```
#Estimate linear model
from sklearn.linear_model import LinearRegression
class_to_explain = 4
simpler_model = LinearRegression()
simpler_model.fit(X=perturbations, y=predictions[:,:,class_to_explain], sample_weight=
coeff = simpler_model.coef_[0]
```

```
#Use coefficients from linear model to extract top features
num_top_features = 4
top_features = np.argsort(coeff)[-num_top_features:]

#Show only the superpixels corresponding to the top features
mask = np.zeros(num_superpixels)
mask[top_features]= True #Activate top superpixels
skimage.io.imshow(perturb_image(Xi,mask,superpixels))
```

This is what LIME returns as explanation. The area of the image (super-pixels) that has a stronger association with the prediction of the disease of class 4. This explanation suggests that the model is doing a good job predicting the class for the given image.

This example shows how LIME can help to increase confidence in a machine-learning model by understanding why it is returning certain predictions.

In [ ]: