loadData

January 25, 2019

```
In [7]: %%javascript
        $('<div id="toc"></div>').css({position: 'fixed', top: '120px', left: 0}).appendTo(docu
        $.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_toc
```

```
<IPython.core.display.Javascript object>
```

0.1 Introduction

Deep Learning become one of the most important topic nowadays, not only in the field computer science but it starting to penetrate it's way to real industries outside information technology. This situation triggered by how Deep Learning can tackle almost all problem that related to pattern recognition. Furthermore Healthcare, one of the most sensitive industry has began intensive research and implementation of deep learning. i.e. (Bhf.org.uk, 2019)

This project also related to implementation of deep learning in healthcare area, more specific about automation of diagnosis types of skin lesion. Focus of this project is to develop deep learning model that able to classify type of skin lesion based on images/photographs.

This project expected to solve classification problem, furthermore image of skin lesions classification problem. Different from prediction/regression which predict a new value based on several input, the classification will only choose from given label/answer.

This can be important for healthcare industry if this project can help people identify skin lesion sooner and can be treated immediately.

0.1.1 Similar cases

Paper that written by Andre Esteva et al. implemented the similar method which is CNN. By using 129,450 clinical images of skin disease to train pre-trained model GoogleNet Inception v3. The classification result performance is similar with all tested experts. This demonstrating an artificial intelligence(CNN) capable of classifying skin cancer with a level of competence comparable to dermatologists.

There is also, an active study/working group that consists of people from industry and academia named International Skin Imaging Collaboration (ISIC) that actively doing research in this particular skin lesion. This group also held several machine learning challenge that very similar with this project. (Challenge2018.isic-archive.com, 2019)

```
In [10]: import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
```

```
import os
from glob import glob
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils import compute_class_weight
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Concatenate, concatenate, Input
from keras.layers import Conv2D, MaxPooling2D, MaxPool2D
from keras.layers. normalization import BatchNormalization
import keras
import itertools
from keras.utils.np_utils import to_categorical
from keras.applications.densenet import preprocess_input,DenseNet121
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
from keras import layers
from keras.callbacks import ModelCheckpoint

from PIL import Image
```

```
In [2]: base_skin_dir = '../src/data'
```

0.2 Methods

This project implements Convolutional Neural Network (CNN) to solve the image classification. The implementation use Python as programming language, Keras (keras.io/, 2019) and TensorFlow (tensorflow.org, 2019) as deep learning packages.

CNN is the state of the art deep learning class that excellent for image classification/recognition based on high resolution image. Moreover, by using Keras it will be easier to build layer model. CNN also provide numbers of alternative methods which one can try arbitrary and compare the result. Bacause of these reason, I choose to only use CNN but more detail about it.

0.2.1 Problem set

Dataset for this problem consists of 10015 skin lesion images including the label and several additional feature (i.e. age, gender and image body location). All images have the same resolution 600x450 pixel (large image), therefore it will be resized to 1/3 of original resolution become 200x150.

This project consists of four types trials/experiments and compare each result in the end, they are: * CNN with standar layer * CNN with standar layer and data augmentation * CNN with multiple input. The input not only images but values as well (i.e. ages, gender, localization) * CNN with transfer learning and data augmentation. Use state of the art pre-trained model.

0.2.2 Data preparation

There are seven labels or classification of skin lesions, they are Melanocytic nevi, Melanoma, Benign keratosis-like lesions, Basal cell carcinoma, Actinic keratoses, Vascular lesions, Dermatofibroma.

```
In [3]: imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                             for x in glob(os.path.join(base_skin_dir, '*', '*.jpg'))}

        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'
        }
```

After create the dictionary for all seven class, store all image path and any other features into one dataframe to be used later on training and validation.

```
In [4]: tile_df = pd.read_csv(os.path.join(base_skin_dir, 'HAM10000_metadata.csv'))
        #tile_df = tile_df.sample(10000)
        tile_df['path'] = tile_df['image_id'].map(imageid_path_dict.get)
        tile_df['cell_type'] = tile_df['dx'].map(lesion_type_dict.get)
        tile_df['cell_type_idx'] = pd.Categorical(tile_df['cell_type']).codes
        tile_df['localization'] = pd.Categorical(tile_df['localization']).codes
        tile_df['sex'] = pd.Categorical(tile_df['sex']).codes
```

The original image is too large, so it should be resize into 1/3 original size.

```
In [5]: tile_df['image'] = tile_df['path'].map(lambda x: np.asarray(Image.open(x).resize((200,
```

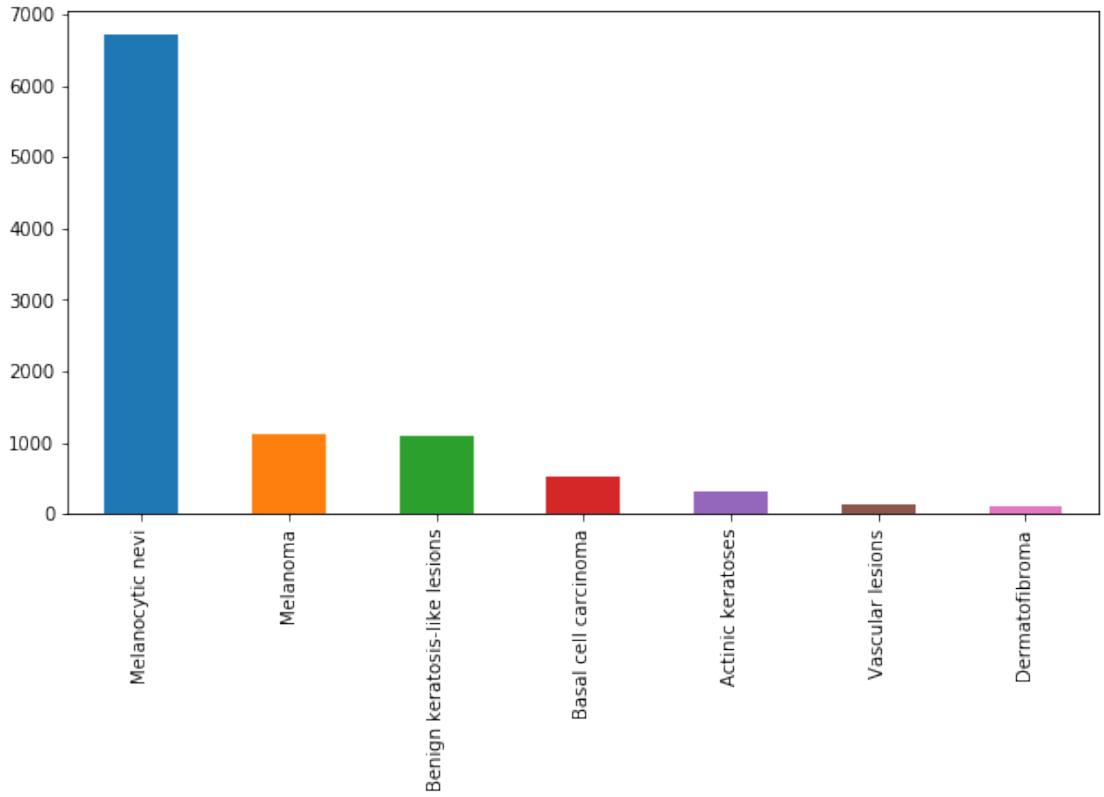
0.2.3 Exploratory data analysis

The image distribution is highly imbalance

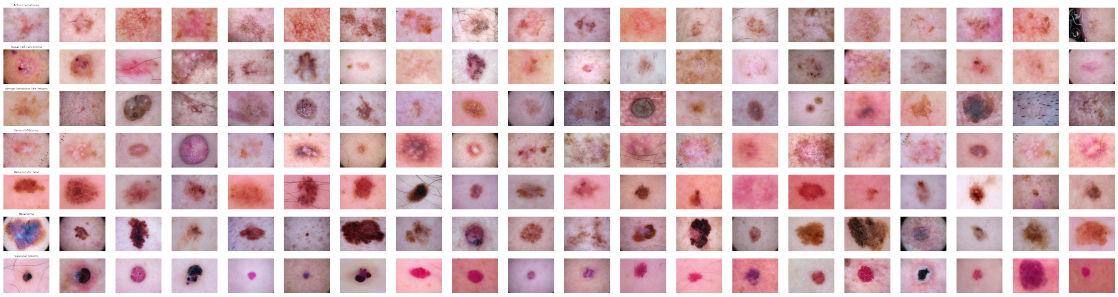
```
In [7]: tile_df['cell_type'].value_counts()

Out[7]: Melanocytic nevi                6705
        Melanoma                        1113
        Benign keratosis-like lesions   1099
        Basal cell carcinoma             514
        Actinic keratoses                327
        Vascular lesions                 142
        Dermatofibroma                   115
        Name: cell_type, dtype: int64

In [7]: # see the image size distribution
        tile_df['image'].map(lambda x: x.shape).value_counts()

Out[7]: (225, 300, 3)    10015
        Name: image, dtype: int64
```

In [21]:
```
fig, ax1 = plt.subplots(1, 1, figsize = (10, 5))
tile_df['cell_type'].value_counts().plot(kind='bar', ax=ax1)
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x2e27f99f128>



In [9]:
```
n_samples = 20
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,
                                         tile_df.sort_values(['cell_type']).groupby('ce
    n_axs[0].set_title(type_name)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples, random_state=2018).i
        c_ax.imshow(c_row['image'])
        c_ax.axis('off')
fig.savefig('category_samples.png', dpi=300)
```



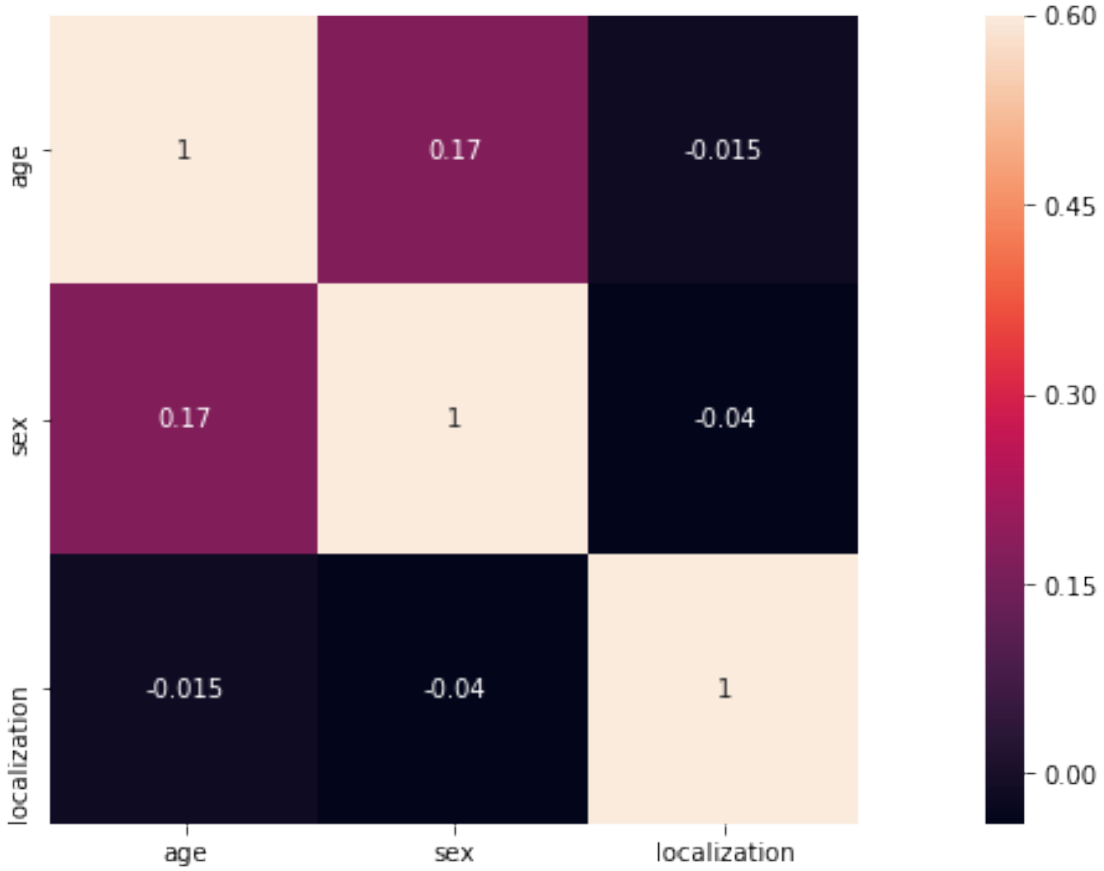
0.2.4 Correlation between additional features

The correlations between all three additional features are very week, the correlation very close to zero. All three features have insignificant effect between each other.

```
In [14]: plot_df = tile_df.iloc[:, [4, 5, 6]]

         a4_dims = (18, 6)
         fig, ax = plt.subplots(figsize=a4_dims)
         sns.heatmap(plot_df.corr(), vmax=0.6, square=True, annot=True)
         plot_df.corr()
```

Out[14]:

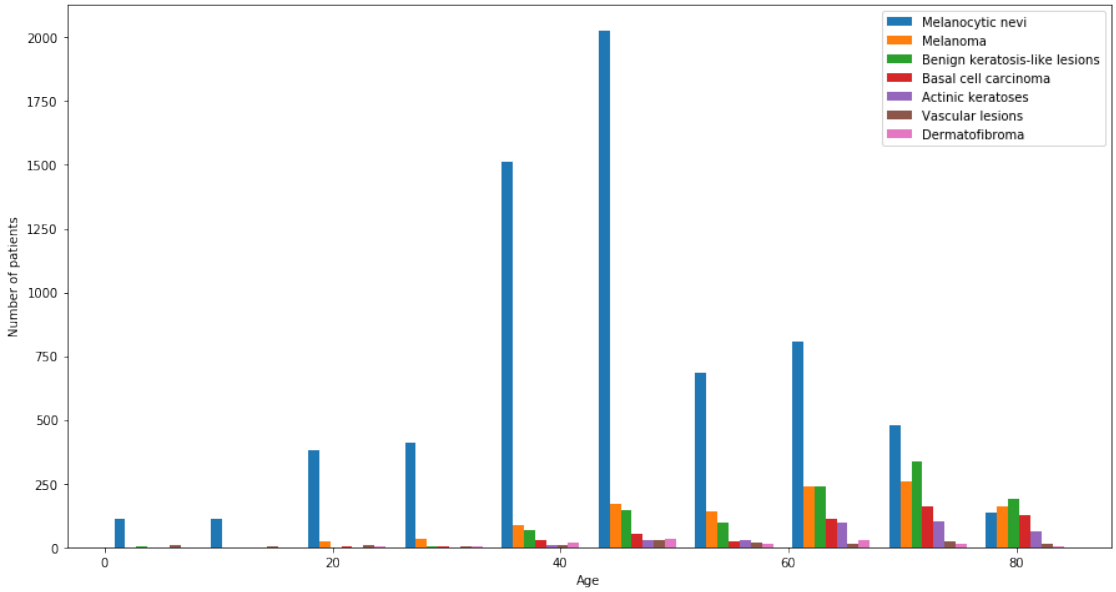
	age	sex	localization
age	1.000000	0.167984	-0.014926
sex	0.167984	1.000000	-0.039845
localization	-0.014926	-0.039845	1.000000



Graph below shows comparison and distribution between the skin type lesion and age of the patients. Except for melanocytic nevi, all other skin lesion increase in accordance to age.

```
In [15]: figure = plt.figure(figsize=(15,8))
         plt.hist([
                  tile_df[tile_df['cell_type']=='Melanocytic nevi']['age'],
                  tile_df[tile_df['cell_type']=='Melanoma']['age'],
                  tile_df[tile_df['cell_type']=='Benign keratosis-like lesions']['age'],
                  tile_df[tile_df['cell_type']=='Basal cell carcinoma']['age'],
                  tile_df[tile_df['cell_type']=='Actinic keratoses']['age'],
                  tile_df[tile_df['cell_type']=='Vascular lesions']['age'],
                  tile_df[tile_df['cell_type']=='Dermatofibroma']['age']],
                  bins = 10,label = ['Melanocytic nevi','Melanoma', 'Benign keratosis-like lesi
         plt.xlabel('Age')
         plt.ylabel('Number of patients')
         plt.legend()

C:\Users\b8060480\Documents\anaconda3\envs\csc8635_project\lib\site-packages\numpy\lib\histogra
  keep = (tmp_a >= first_edge)
C:\Users\b8060480\Documents\anaconda3\envs\csc8635_project\lib\site-packages\numpy\lib\histogra
  keep &= (tmp_a <= last_edge)
```

Out[15]: <matplotlib.legend.Legend at 0x2e27f6d09e8>




```
In [8]: from sklearn.utils import resample

        tile_df_min1 = tile_df[tile_df['cell_type']=='Melanoma']
        tile_df_min2 = tile_df[tile_df['cell_type']=='Benign keratosis-like lesions']
        tile_df_min3 = tile_df[tile_df['cell_type']=='Basal cell carcinoma']
        tile_df_min4 = tile_df[tile_df['cell_type']=='Actinic keratoses']
```

```
tile_df_min5 = tile_df[tile_df['cell_type']=='Vascular lesions']
tile_df_min6 = tile_df[tile_df['cell_type']=='Dermatofibroma']
tile_df_maj = tile_df[tile_df['cell_type']=='Melanocytic nevi']

# Upsample minority class
df_minority_upsampled1 = resample(tile_df_min1,
                                  replace=True,     # sample with replacement
                                  n_samples=2000,    # to match majority class
                                  random_state=42) # reproducible results
df_minority_upsampled2 = resample(tile_df_min2,
                                  replace=True,     # sample with replacement
                                  n_samples=2000,    # to match majority class
                                  random_state=42) # reproducible results
df_minority_upsampled3 = resample(tile_df_min3,
                                  replace=True,     # sample with replacement
                                  n_samples=2000,    # to match majority class
                                  random_state=42) # reproducible results
df_minority_upsampled4 = resample(tile_df_min4,
                                  replace=True,     # sample with replacement
                                  n_samples=2000,    # to match majority class
                                  random_state=42) # reproducible results
df_minority_upsampled5 = resample(tile_df_min5,
                                  replace=True,     # sample with replacement
                                  n_samples=2000,    # to match majority class
                                  random_state=42) # reproducible results
df_minority_upsampled6 = resample(tile_df_min6,
                                  replace=True,     # sample with replacement
                                  n_samples=2000,    # to match majority class
                                  random_state=42) # reproducible results
df_majority_downsampled6 = resample(tile_df_maj,
                                  replace=False,     # sample with replacement
                                  n_samples=2500,    # to match majority class
                                  random_state=42) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority_downsampled6, df_minority_upsampled1, df_minorit
                          df_minority_upsampled3, df_minority_upsampled4, df_minority_u
                          df_minority_upsampled6])
```

In [16]:
```
y = tile_df.cell_type_idx

from sklearn.model_selection import train_test_split
x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(tile_df, y, test_size=0.2

x_train = np.asarray(x_train_o['image'].tolist())
x_test = np.asarray(x_test_o['image'].tolist())

#x_train_add = np.asarray([x_train_o['localization'], x_train_o['sex'], x_train_o['ag
```

```
#x_test_add = np.asarray([x_test_o['localization'], x_test_o['sex'], x_test_o['age']]

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

# 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)
```

0.2.5 CNN

CNN contains sequence of layers, which has Convolutional Layer, Pooling Layer, and Fully-Connected Layer as it's general architecture. CNN performs great because the weight sharing capability that provide by the convolution layers, it also faster than general multi layer perceptron because pooling capability to reduce dimension of image.

For generalisation and decrease the overconfidence of the model, drop-out layer is also used ini this development.

Gradient Decent optimization algorithm use "adam" without any customization.

0.2.6 CNN with customized layer for multiple input

The idea of using other features in addition to images came out while doing EDA. Probably there is opportunity to make the model more robust. I personally very glad for figure this out eventhough it is not common in CNN, and I struggled so much to make it work because cannot find a working example. In order to use this model, the data features need to be cleaned-up first to avoid N/A value and also one-hot encoding for non integer input (i.e. gender)

```
In [17]: img_input = Input(shape=(150, 200,  3))
         conv1 = Conv2D(32, kernel_size = (3, 3), activation='relu')(img_input)
         pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
         batch1 = BatchNormalization()(pool1)
         conv2 = Conv2D(64, kernel_size = (3, 3), activation='relu')(batch1)
         pool2 = MaxPooling2D(pool_size=(2,2))(conv2)
         batch2 = BatchNormalization()(pool2)
         conv3 = Conv2D(96, kernel_size = (3, 3), activation='relu')(batch2)
         pool3 = MaxPooling2D(pool_size=(2,2))(conv3)
         batch3 = BatchNormalization()(pool3)
         conv4 = Conv2D(96, kernel_size = (3, 3), activation='relu')(batch3)
         pool4 = MaxPooling2D(pool_size=(2,2))(conv4)
         batch4 = BatchNormalization()(pool4)
         drop1 = Dropout(0.2)(batch4)
         flat1 = Flatten()(drop1)
```

```
        dense1 = Dense(128, activation='relu')(flat1)
        drop2 = Dropout(0.3)(dense1)
        dense2 = Dense(64, activation='relu')(drop2)
        drop3 = Dropout(0.4)(dense2)
        #dense2 = Dense(64, activation='relu')(drop3)
        #drop3 = Dropout(0.5)(dense2)


        additional_features = Input(shape=(1,))
        add_dense1 = Dense(64, activation='relu')(additional_features)

        merged1 = concatenate([drop3, add_dense1], axis=-1)
        out1 = Dense(64, activation='relu')(merged1)

        additional_features2 = Input(shape=(1,))
        add_dense2 = Dense(64, activation='relu')(additional_features2)

        merged2 = concatenate([out1, add_dense2], axis=-1)
        out2 = Dense(64, activation='relu')(merged2)

        additional_features3 = Input(shape=(1,))
        add_dense3 = Dense(64, activation='relu')(additional_features3)

        merged3 = concatenate([out2, add_dense3], axis=-1)
        out3 = Dense(7, activation='softmax')(merged3)

        model = Model([img_input, additional_features, additional_features2, additional_featu

        model.compile(optimizer='adam', loss=keras.losses.categorical_crossentropy, metrics=[
        model.summary()
```

```
_________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
input_1 (InputLayer)         (None, 150, 200, 3)       0
_________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, 148, 198, 32)      896         input_1[0][0]
_________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 74, 99, 32)     0           conv2d_1[0][0]
_________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 74, 99, 32)     128         max_pooling2d_1[0][0]
_________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 97, 64)        18496       batch_normalization_1[0][0]
_________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 36, 48, 64)     0           conv2d_2[0][0]
_________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 36, 48, 64)     256         max_pooling2d_2[0][0]
_________________________________________________________________________________________
```

```
conv2d_3 (Conv2D)               (None, 34, 46, 96)   55392       batch_normalization_2[0][0]
_________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 17, 23, 96)   0           conv2d_3[0][0]
_________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 17, 23, 96)   384         max_pooling2d_3[0][0]
_________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 15, 21, 96)   83040       batch_normalization_3[0][0]
_________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 7, 10, 96)    0           conv2d_4[0][0]
_________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 7, 10, 96)    384         max_pooling2d_4[0][0]
_________________________________________________________________________________________
dropout_1 (Dropout)             (None, 7, 10, 96)    0           batch_normalization_4[0][0]
_________________________________________________________________________________________
flatten_1 (Flatten)             (None, 6720)         0           dropout_1[0][0]
_________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          860288      flatten_1[0][0]
_________________________________________________________________________________________
dropout_2 (Dropout)             (None, 128)          0           dense_1[0][0]
_________________________________________________________________________________________
dense_2 (Dense)                 (None, 64)           8256        dropout_2[0][0]
_________________________________________________________________________________________
input_2 (InputLayer)            (None, 1)            0
_________________________________________________________________________________________
dropout_3 (Dropout)             (None, 64)           0           dense_2[0][0]
_________________________________________________________________________________________
dense_3 (Dense)                 (None, 64)           128         input_2[0][0]
_________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 128)          0           dropout_3[0][0]
                                                                 dense_3[0][0]
_________________________________________________________________________________________
input_3 (InputLayer)            (None, 1)            0
_________________________________________________________________________________________
dense_4 (Dense)                 (None, 64)           8256        concatenate_1[0][0]
_________________________________________________________________________________________
dense_5 (Dense)                 (None, 64)           128         input_3[0][0]
_________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 128)          0           dense_4[0][0]
                                                                 dense_5[0][0]
_________________________________________________________________________________________
input_4 (InputLayer)            (None, 1)            0
_________________________________________________________________________________________
dense_6 (Dense)                 (None, 64)           8256        concatenate_2[0][0]
_________________________________________________________________________________________
dense_7 (Dense)                 (None, 64)           128         input_4[0][0]
_________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 128)          0           dense_6[0][0]
                                                                 dense_7[0][0]
```

```
dense_8 (Dense)                 (None, 7)            903         concatenate_3[0][0]
==================================================================================================
Total params: 1,045,319
Trainable params: 1,044,743
Non-trainable params: 576
__________________________________________________________________________________________________
```

0.2.7 CNN with standard layer

This model develop using combination of five Convolution 2d layers, five polling layers, two fully connected layers and output layer using softmax as activation to get output as percentages of confidence for all class.

0.2.8 CNN with standard layer and data augmentation

With the same layer as previous model, this model use data augmentation to enhance learning capability and also help to create more sample. Data augmentation that used in this project are Rotate, Flip (Horizontal and Vertical), Zoom and etc.

```
In [ ]: model = Sequential()
        model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(150, 200,  3
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(BatchNormalization())
        model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(BatchNormalization())
        model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(BatchNormalization())
        model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(BatchNormalization())
        model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(BatchNormalization())
        model.add(Dropout(0.3))
        model.add(Flatten())
        model.add(Dense(256, activation='relu'))
        model.add(Dropout(0.3))
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(7, activation = 'softmax'))


        model.compile(optimizer='adam', loss=keras.losses.categorical_crossentropy , metrics=[
        model.summary()
```

0.2.9 CNN with transfer learning and data augmentation

Transfer learning is to use predefined or pre-trained model and configured it to fit the input and output. It can accelerate the learning curve. For this project, the pre-trained model is DenseNet121 (Huang, G.).

```
In [7]: densenet = DenseNet121(
            weights='imagenet',
            include_top=False,
            input_shape=(150,200,3)
        )

        model = Sequential()
        model.add(densenet)
        model.add(layers.GlobalAveragePooling2D())
        model.add(layers.Dense(7, activation='softmax'))
        model.summary()


        model.compile(optimizer='adam', loss=keras.losses.categorical_crossentropy , metrics=[
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 4, 6, 1024)        7037504
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 7175
=================================================================
Total params: 7,044,679
Trainable params: 6,961,031
Non-trainable params: 83,648
_________________________________________________________________
```

0.2.10 Problems

The hardest technical problem for this project was the imbalance dataset, the Melanocytic nevi sample dominates other sample (more than 60% of all sample) mean while several class have only have 1% member sample. There are several way to deal with this situation some of them are: * oversampling - increase the smaller sample by duplicate it until it has similar number with the highest sample number * undersampling - delete some image from the higher sample class closer to the smallest number * augmentation - similar with oversampling but not only duplicate but change the image (flip, rotate, zoom-in, etc.) * generate synthetic model - SMOTE: Synthetic Minority Over-sampling Technique * class weight or model penalty - use different weight for each class according to sample numbers

Over sampling and under sampling was not a good choice because, the difference between the highest sample and the lowest sample is to far away at 1:60, either many information will lost

(undersampling) or overfitting occur because duplicated so many times.

Augmentation also not a very good option, almost similar with oversampling and it will create huge fake image.

While synthetic model can be one of good option, but the concept looks difficult, so will become one area of future development.

Set the sensitivity of each class seems the most plausible options and will be executed as solution.

```
In [19]: class_weight = compute_class_weight('balanced', [0,1,2,3,4,5,6], tile_df.cell_type_idx
```

0.3 Results

Out of four model Transfer learning with data augmentation gives the best result arround 83%-85% accuracy followed by model with standard layer and data augmentation with 78%-80% accuracy. On the other hand CNN with multiple input gives the worst result between all four model with 70%-76% accuracy and CNN with only standard layer scored 75-76%.

```
In [20]: batch_size = 32
         epochs = 70

         history = model.fit([x_train, x_train_o['localization'], x_train_o['sex'], x_train_o[
                             batch_size=batch_size,
                             epochs=epochs,
                             verbose=1,
                             class_weight= class_weight,
                             validation_data=([x_test, x_test_o['localization'],  x_test_o['sex

         score = model.evaluate([x_test, x_test_o['localization'],  x_test_o['sex'], x_test_o[
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
Train on 7511 samples, validate on 2504 samples
Epoch 1/70
7511/7511 [==============================] - 17s 2ms/step - loss: 1.0313 - acc: 0.6596 - val_lo
Epoch 2/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.9135 - acc: 0.6777 - val_lo
Epoch 3/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.8470 - acc: 0.6930 - val_lo
Epoch 4/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.8295 - acc: 0.7032 - val_lo
Epoch 5/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.8024 - acc: 0.7027 - val_lo
Epoch 6/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.7771 - acc: 0.7211 - val_lo
Epoch 7/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.7466 - acc: 0.7297 - val_lo
Epoch 8/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.7257 - acc: 0.7311 - val_lo
Epoch 9/70
```

```
7511/7511 [==============================] - 15s 2ms/step - loss: 0.7151 - acc: 0.7466 - val_lo
Epoch 10/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.6836 - acc: 0.7550 - val_lo
Epoch 11/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.6521 - acc: 0.7650 - val_lo
Epoch 12/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.6432 - acc: 0.7678 - val_lo
Epoch 13/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.6271 - acc: 0.7729 - val_lo
Epoch 14/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.6038 - acc: 0.7830 - val_lo
Epoch 15/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.5830 - acc: 0.7938 - val_lo
Epoch 16/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.5624 - acc: 0.7963 - val_lo
Epoch 17/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.5259 - acc: 0.8080 - val_lo
Epoch 18/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.5176 - acc: 0.8141 - val_lo
Epoch 19/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.4800 - acc: 0.8276 - val_lo
Epoch 20/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.4539 - acc: 0.8397 - val_lo
Epoch 21/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.4400 - acc: 0.8417 - val_lo
Epoch 22/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.4103 - acc: 0.8562 - val_lo
Epoch 23/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.3824 - acc: 0.8590 - val_lo
Epoch 24/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.3598 - acc: 0.8677 - val_lo
Epoch 25/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.3123 - acc: 0.8883 - val_lo
Epoch 26/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.3034 - acc: 0.8940 - val_lo
Epoch 27/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.2759 - acc: 0.9035 - val_lo
Epoch 28/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.2603 - acc: 0.9108 - val_lo
Epoch 29/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.2473 - acc: 0.9084 - val_lo
Epoch 30/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.2306 - acc: 0.9187 - val_lo
Epoch 31/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.2093 - acc: 0.9258 - val_lo
Epoch 32/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.2172 - acc: 0.9242 - val_lo
Epoch 33/70
```

```
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1882 - acc: 0.9334 - val_lo
Epoch 34/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1746 - acc: 0.9426 - val_lo
Epoch 35/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1858 - acc: 0.9361 - val_lo
Epoch 36/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1545 - acc: 0.9485 - val_lo
Epoch 37/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1528 - acc: 0.9499 - val_lo
Epoch 38/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1457 - acc: 0.9519 - val_lo
Epoch 39/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1400 - acc: 0.9545 - val_lo
Epoch 40/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1231 - acc: 0.9593 - val_lo
Epoch 41/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1259 - acc: 0.9581 - val_lo
Epoch 42/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1196 - acc: 0.9622 - val_lo
Epoch 43/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1280 - acc: 0.9578 - val_lo
Epoch 44/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1116 - acc: 0.9649 - val_lo
Epoch 45/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1186 - acc: 0.9645 - val_lo
Epoch 46/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1017 - acc: 0.9674 - val_lo
Epoch 47/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0972 - acc: 0.9683 - val_lo
Epoch 48/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1204 - acc: 0.9610 - val_lo
Epoch 49/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.1014 - acc: 0.9680 - val_lo
Epoch 50/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0963 - acc: 0.9675 - val_lo
Epoch 51/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0910 - acc: 0.9687 - val_lo
Epoch 52/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0983 - acc: 0.9694 - val_lo
Epoch 53/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0927 - acc: 0.9686 - val_lo
Epoch 54/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0752 - acc: 0.9759 - val_lo
Epoch 55/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0931 - acc: 0.9706 - val_lo
Epoch 56/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0799 - acc: 0.9759 - val_lo
Epoch 57/70
```

```
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0827 - acc: 0.9728 - val_lo
Epoch 58/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0802 - acc: 0.9756 - val_lo
Epoch 59/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0720 - acc: 0.9770 - val_lo
Epoch 60/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0736 - acc: 0.9772 - val_lo
Epoch 61/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0693 - acc: 0.9760 - val_lo
Epoch 62/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0668 - acc: 0.9775 - val_lo
Epoch 63/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0704 - acc: 0.9774 - val_lo
Epoch 64/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0821 - acc: 0.9756 - val_lo
Epoch 65/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0703 - acc: 0.9770 - val_lo
Epoch 66/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0682 - acc: 0.9779 - val_lo
Epoch 67/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0721 - acc: 0.9778 - val_lo
Epoch 68/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0632 - acc: 0.9800 - val_lo
Epoch 69/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0687 - acc: 0.9811 - val_lo
Epoch 70/70
7511/7511 [==============================] - 15s 2ms/step - loss: 0.0782 - acc: 0.9760 - val_lo
Test loss: 1.1074325989800902
Test accuracy: 0.7671725239616614
```

```
In [10]: batch_size = 32
         epochs = 100

         history = model.fit([x_train] , y_train,
                             batch_size=batch_size,
                             epochs=epochs,
                             verbose=1,
                             class_weight= class_weight,
                             validation_data=([x_test], y_test))

         score = model.evaluate([x_test], y_test, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
Train on 7511 samples, validate on 2504 samples
Epoch 1/100
7511/7511 [==============================] - 14s 2ms/step - loss: 1.0676 - acc: 0.6532 - val_lo
```

```
Epoch 2/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.9069 - acc: 0.6850 - val_lo
Epoch 3/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.8629 - acc: 0.6936 - val_lo
Epoch 4/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.8086 - acc: 0.7063 - val_lo
Epoch 5/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.7988 - acc: 0.7084 - val_lo
Epoch 6/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.7589 - acc: 0.7200 - val_lo
Epoch 7/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.7522 - acc: 0.7292 - val_lo
Epoch 8/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.7263 - acc: 0.7349 - val_lo
Epoch 9/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6998 - acc: 0.7524 - val_lo
Epoch 10/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6941 - acc: 0.7493 - val_lo
Epoch 11/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6870 - acc: 0.7506 - val_lo
Epoch 12/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6672 - acc: 0.7585 - val_lo
Epoch 13/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6591 - acc: 0.7615 - val_lo
Epoch 14/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6432 - acc: 0.7646 - val_lo
Epoch 15/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6305 - acc: 0.7735 - val_lo
Epoch 16/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6074 - acc: 0.7765 - val_lo
Epoch 17/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.6002 - acc: 0.7814 - val_lo
Epoch 18/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5933 - acc: 0.7837 - val_lo
Epoch 19/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5724 - acc: 0.7910 - val_lo
Epoch 20/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5601 - acc: 0.7947 - val_lo
Epoch 21/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5383 - acc: 0.8040 - val_lo
Epoch 22/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5277 - acc: 0.8096 - val_lo
Epoch 23/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5032 - acc: 0.8188 - val_lo
Epoch 24/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.5064 - acc: 0.8123 - val_lo
Epoch 25/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.4794 - acc: 0.8213 - val_lo
```

```
Epoch 26/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.4686 - acc: 0.8291 - val_lo
Epoch 27/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.4550 - acc: 0.8324 - val_lo
Epoch 28/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.4295 - acc: 0.8388 - val_lo
Epoch 29/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.4030 - acc: 0.8502 - val_lo
Epoch 30/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.3820 - acc: 0.8591 - val_lo
Epoch 31/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.3674 - acc: 0.8639 - val_lo
Epoch 32/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.3462 - acc: 0.8715 - val_lo
Epoch 33/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.3497 - acc: 0.8713 - val_lo
Epoch 34/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.3206 - acc: 0.8848 - val_lo
Epoch 35/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.3129 - acc: 0.8908 - val_lo
Epoch 36/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2960 - acc: 0.8886 - val_lo
Epoch 37/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2846 - acc: 0.8955 - val_lo
Epoch 38/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2668 - acc: 0.9045 - val_lo
Epoch 39/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2450 - acc: 0.9125 - val_lo
Epoch 40/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2715 - acc: 0.9045 - val_lo
Epoch 41/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2280 - acc: 0.9200 - val_lo
Epoch 42/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2344 - acc: 0.9188 - val_lo
Epoch 43/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2072 - acc: 0.9260 - val_lo
Epoch 44/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.2068 - acc: 0.9276 - val_lo
Epoch 45/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1948 - acc: 0.9274 - val_lo
Epoch 46/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1870 - acc: 0.9329 - val_lo
Epoch 47/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1904 - acc: 0.9312 - val_lo
Epoch 48/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1638 - acc: 0.9401 - val_lo
Epoch 49/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1572 - acc: 0.9441 - val_lo
```

```
Epoch 50/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1700 - acc: 0.9405 - val_lo
Epoch 51/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1591 - acc: 0.9434 - val_lo
Epoch 52/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1358 - acc: 0.9522 - val_lo
Epoch 53/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1623 - acc: 0.9479 - val_lo
Epoch 54/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1505 - acc: 0.9494 - val_lo
Epoch 55/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1331 - acc: 0.9559 - val_lo
Epoch 56/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1228 - acc: 0.9582 - val_lo
Epoch 57/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1312 - acc: 0.9557 - val_lo
Epoch 58/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1353 - acc: 0.9535 - val_lo
Epoch 59/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1222 - acc: 0.9599 - val_lo
Epoch 60/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1218 - acc: 0.9578 - val_lo
Epoch 61/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1204 - acc: 0.9606 - val_lo
Epoch 62/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1142 - acc: 0.9626 - val_lo
Epoch 63/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1146 - acc: 0.9638 - val_lo
Epoch 64/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1036 - acc: 0.9660 - val_lo
Epoch 65/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1065 - acc: 0.9666 - val_lo
Epoch 66/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0974 - acc: 0.9657 - val_lo
Epoch 67/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1166 - acc: 0.9601 - val_lo
Epoch 68/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1005 - acc: 0.9684 - val_lo
Epoch 69/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0923 - acc: 0.9688 - val_lo
Epoch 70/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1137 - acc: 0.9621 - val_lo
Epoch 71/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1007 - acc: 0.9678 - val_lo
Epoch 72/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0858 - acc: 0.9750 - val_lo
Epoch 73/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0952 - acc: 0.9722 - val_lo
```

```
Epoch 74/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0788 - acc: 0.9758 - val_lo
Epoch 75/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0793 - acc: 0.9719 - val_lo
Epoch 76/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0712 - acc: 0.9746 - val_lo
Epoch 77/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0857 - acc: 0.9730 - val_lo
Epoch 78/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0724 - acc: 0.9772 - val_lo
Epoch 79/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0916 - acc: 0.9694 - val_lo
Epoch 80/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0795 - acc: 0.9762 - val_lo
Epoch 81/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0742 - acc: 0.9771 - val_lo
Epoch 82/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1083 - acc: 0.9687 - val_lo
Epoch 83/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0672 - acc: 0.9783 - val_lo
Epoch 84/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0766 - acc: 0.9772 - val_lo
Epoch 85/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.1057 - acc: 0.9672 - val_lo
Epoch 86/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0737 - acc: 0.9766 - val_lo
Epoch 87/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0619 - acc: 0.9798 - val_lo
Epoch 88/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0655 - acc: 0.9776 - val_lo
Epoch 89/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0745 - acc: 0.9774 - val_lo
Epoch 90/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0697 - acc: 0.9794 - val_lo
Epoch 91/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0594 - acc: 0.9802 - val_lo
Epoch 92/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0899 - acc: 0.9734 - val_lo
Epoch 93/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0807 - acc: 0.9748 - val_lo
Epoch 94/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0589 - acc: 0.9816 - val_lo
Epoch 95/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0745 - acc: 0.9775 - val_lo
Epoch 96/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0667 - acc: 0.9776 - val_lo
Epoch 97/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0560 - acc: 0.9824 - val_lo
```

```
Epoch 98/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0673 - acc: 0.9779 - val_lo
Epoch 99/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0695 - acc: 0.9772 - val_lo
Epoch 100/100
7511/7511 [==============================] - 12s 2ms/step - loss: 0.0788 - acc: 0.9747 - val_lo
Test loss: 1.3927374640211891
Test accuracy: 0.7687699680511182
```

```
In [ ]: batch_size = 32
        epochs = 60


        datagen = ImageDataGenerator(
                rotation_range=180,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
                horizontal_flip=True,
                vertical_flip=True,
                fill_mode='nearest')

        history = model.fit_generator(generator=datagen.flow(x_train, y_train, batch_size=batch
                              epochs=epochs,
                              steps_per_epoch=x_train.shape[0] // batch_size * 2,
                              verbose=1,
                              workers=8,
                              class_weight= class_weight,
                              validation_data=(x_test, y_test))

        score = model.evaluate([x_test], y_test, verbose=0)
        print('Test loss:', score[0])
        print('Test accuracy:', score[1])
```

```
Epoch 1/60
468/468 [==============================] - 125s 267ms/step - loss: 0.7596 - acc: 0.7315 - val_l
Epoch 2/60
468/468 [==============================] - 99s 212ms/step - loss: 0.6424 - acc: 0.7613 - val_lo
Epoch 3/60
468/468 [==============================] - 99s 212ms/step - loss: 0.5911 - acc: 0.7837 - val_lo
Epoch 4/60
468/468 [==============================] - 99s 212ms/step - loss: 0.6445 - acc: 0.7644 - val_lo
Epoch 5/60
468/468 [==============================] - 99s 212ms/step - loss: 0.5774 - acc: 0.7833 - val_lo
Epoch 6/60
468/468 [==============================] - 99s 212ms/step - loss: 0.5840 - acc: 0.7834 - val_lo
```

```
Epoch 7/60
468/468 [==============================] - 99s 212ms/step - loss: 0.5234 - acc: 0.8026 - val_lo
Epoch 8/60
468/468 [==============================] - 99s 211ms/step - loss: 0.4940 - acc: 0.8180 - val_lo
Epoch 9/60
468/468 [==============================] - 99s 211ms/step - loss: 0.4720 - acc: 0.8198 - val_lo
Epoch 10/60
468/468 [==============================] - 99s 211ms/step - loss: 0.4529 - acc: 0.8299 - val_lo
Epoch 11/60
468/468 [==============================] - 99s 211ms/step - loss: 0.4394 - acc: 0.8323 - val_lo
Epoch 12/60
468/468 [==============================] - 99s 211ms/step - loss: 0.4228 - acc: 0.8389 - val_lo
Epoch 13/60
468/468 [==============================] - 99s 211ms/step - loss: 0.4052 - acc: 0.8457 - val_lo
Epoch 14/60
468/468 [==============================] - 99s 211ms/step - loss: 0.3935 - acc: 0.8506 - val_lo
Epoch 15/60
468/468 [==============================] - 99s 211ms/step - loss: 0.3762 - acc: 0.8568 - val_lo
Epoch 16/60
468/468 [==============================] - 99s 211ms/step - loss: 0.3612 - acc: 0.8608 - val_lo
Epoch 17/60
468/468 [==============================] - 99s 212ms/step - loss: 0.3471 - acc: 0.8669 - val_lo
Epoch 18/60
468/468 [==============================] - 99s 211ms/step - loss: 0.3289 - acc: 0.8745 - val_lo
Epoch 19/60
468/468 [==============================] - 99s 211ms/step - loss: 0.3201 - acc: 0.8775 - val_lo
Epoch 20/60
468/468 [==============================] - 99s 211ms/step - loss: 0.3045 - acc: 0.8840 - val_lo
Epoch 21/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2912 - acc: 0.8888 - val_lo
Epoch 22/60
468/468 [==============================] - 99s 212ms/step - loss: 0.2826 - acc: 0.8940 - val_lo
Epoch 23/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2617 - acc: 0.8963 - val_lo
Epoch 24/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2561 - acc: 0.9017 - val_lo
Epoch 25/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2424 - acc: 0.9092 - val_lo
Epoch 26/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2291 - acc: 0.9117 - val_lo
Epoch 27/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2186 - acc: 0.9164 - val_lo
Epoch 28/60
468/468 [==============================] - 99s 211ms/step - loss: 0.2149 - acc: 0.9178 - val_lo
Epoch 29/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1993 - acc: 0.9230 - val_lo
Epoch 30/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1951 - acc: 0.9253 - val_lo
```

```
Epoch 31/60
468/468 [==============================] - 99s 212ms/step - loss: 0.1832 - acc: 0.9288 - val_lo
Epoch 32/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1795 - acc: 0.9312 - val_lo
Epoch 33/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1687 - acc: 0.9374 - val_lo
Epoch 34/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1676 - acc: 0.9371 - val_lo
Epoch 35/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1522 - acc: 0.9429 - val_lo
Epoch 36/60
468/468 [==============================] - 99s 212ms/step - loss: 0.1398 - acc: 0.9482 - val_lo
Epoch 37/60
468/468 [==============================] - 99s 212ms/step - loss: 0.1553 - acc: 0.9428 - val_lo
Epoch 38/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1340 - acc: 0.9494 - val_lo
Epoch 39/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1282 - acc: 0.9502 - val_lo
Epoch 40/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1225 - acc: 0.9535 - val_lo
Epoch 41/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1214 - acc: 0.9556 - val_lo
Epoch 42/60
468/468 [==============================] - 99s 212ms/step - loss: 0.1255 - acc: 0.9537 - val_lo
Epoch 43/60
468/468 [==============================] - 99s 212ms/step - loss: 0.1143 - acc: 0.9566 - val_lo
Epoch 44/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1143 - acc: 0.9554 - val_lo
Epoch 45/60
468/468 [==============================] - 99s 212ms/step - loss: 0.1087 - acc: 0.9601 - val_lo
Epoch 46/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0972 - acc: 0.9643 - val_lo
Epoch 47/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1020 - acc: 0.9627 - val_lo
Epoch 48/60
468/468 [==============================] - 99s 211ms/step - loss: 0.1031 - acc: 0.9637 - val_lo
Epoch 49/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0998 - acc: 0.9626 - val_lo
Epoch 50/60
468/468 [==============================] - 99s 212ms/step - loss: 0.0933 - acc: 0.9648 - val_lo
Epoch 51/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0909 - acc: 0.9676 - val_lo
Epoch 52/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0877 - acc: 0.9669 - val_lo
Epoch 53/60
468/468 [==============================] - 99s 212ms/step - loss: 0.0982 - acc: 0.9649 - val_lo
Epoch 54/60
468/468 [==============================] - 99s 212ms/step - loss: 0.0894 - acc: 0.9681 - val_lo
```

```
Epoch 55/60
468/468 [==============================] - 99s 212ms/step - loss: 0.0752 - acc: 0.9730 - val_lo
Epoch 56/60
468/468 [==============================] - 99s 212ms/step - loss: 0.0721 - acc: 0.9739 - val_lo
Epoch 57/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0758 - acc: 0.9712 - val_lo
Epoch 58/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0755 - acc: 0.9700 - val_lo
Epoch 59/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0703 - acc: 0.9747 - val_lo
Epoch 60/60
468/468 [==============================] - 99s 211ms/step - loss: 0.0695 - acc: 0.9728 - val_lo
```

```
In [10]: # serialize model to JSON
         model_json = model.to_json()
         with open("model.json", "w") as json_file:
             json_file.write(model_json)
         # serialize weights to HDF5
         model.save_weights("model.h5")
         print("Saved model to disk")
```

```
Saved model to disk
```

```
In [10]: #y_score = model.predict([x_test, x_test_o['localization'],  x_test_o['sex']])
         y_score = model.predict(x_test)
```

0.3.1 Evaluation

The best result evaluated using confusion matrix to see how the prediction distributed over the classes. The second heat map belows shows the percentage of correct/incorrect answer. The lowest percentage score is "dermatofibroma", the percentage of correct answer is only 44% and evenmore the model misclassify "dermatofibroma" as "Melanocytic nevi" 25% of time. Another missclassification is 24% from total predictions of melanoma predicted as "Melanocytic nevi" which in this case can be fatal because if melanoma is not treated at early staged it can be deadly.

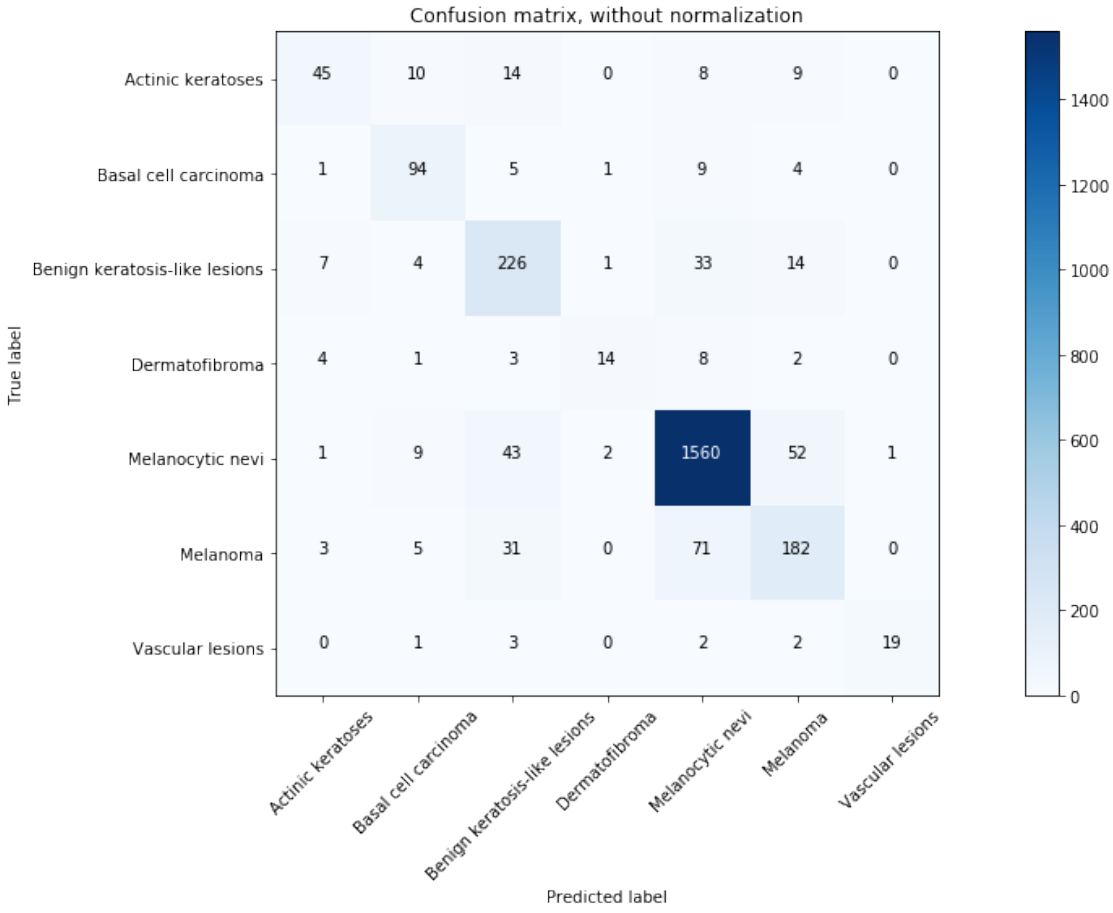
From these two evaluation can be interpreted that this is still an unsuccesful model and still far from good and can be considered as fail model.

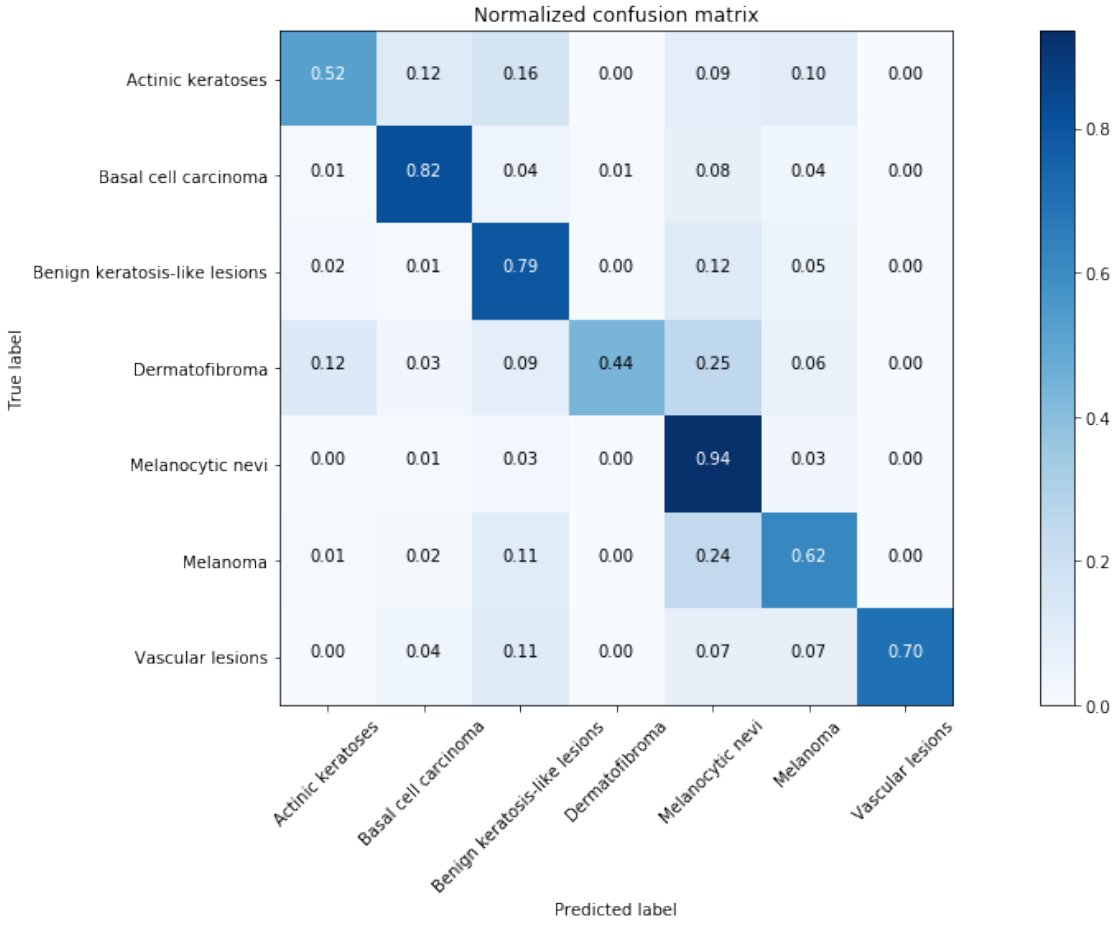
```
In [11]: def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             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()


    predict_class = np.argmax(y_score, axis=1)
    test_class = np.argmax(y_test, axis=1)
    # Compute confusion matrix
    cnf_matrix = confusion_matrix(test_class, predict_class)
    np.set_printoptions(precision=2)
    class_name = ['Actinic keratoses', 'Basal cell carcinoma', 'Benign keratosis-like lesi
                  'Melanocytic nevi', 'Melanoma', 'Vascular lesions']

    # Plot non-normalized confusion matrix
    plt.figure(figsize=(15,8))
    plot_confusion_matrix(cnf_matrix, classes=class_name,
                          title='Confusion matrix, without normalization')

    # Plot normalized confusion matrix
    plt.figure(figsize=(15,8))
    plot_confusion_matrix(cnf_matrix, classes=class_name, normalize=True, title='Normalize

    plt.show()
```

```
Confusion matrix, without normalization
[[ 45  10  14   0   8   9   0]
 [  1  94   5   1   9   4   0]
 [  7   4 226   1  33  14   0]
```

```
 [   4    1    3   14    8    2    0]
 [   1    9   43    2 1560   52    1]
 [   3    5   31    0   71  182    0]
 [   0    1    3    0    2    2   19]]
Normalized confusion matrix
[[5.23e-01 1.16e-01 1.63e-01 0.00e+00 9.30e-02 1.05e-01 0.00e+00]
 [8.77e-03 8.25e-01 4.39e-02 8.77e-03 7.89e-02 3.51e-02 0.00e+00]
 [2.46e-02 1.40e-02 7.93e-01 3.51e-03 1.16e-01 4.91e-02 0.00e+00]
 [1.25e-01 3.12e-02 9.38e-02 4.38e-01 2.50e-01 6.25e-02 0.00e+00]
 [6.00e-04 5.40e-03 2.58e-02 1.20e-03 9.35e-01 3.12e-02 6.00e-04]
 [1.03e-02 1.71e-02 1.06e-01 0.00e+00 2.43e-01 6.23e-01 0.00e+00]
 [0.00e+00 3.70e-02 1.11e-01 0.00e+00 7.41e-02 7.41e-02 7.04e-01]]
```





In [13]:
```
from sklearn.metrics import balanced_accuracy_score
```

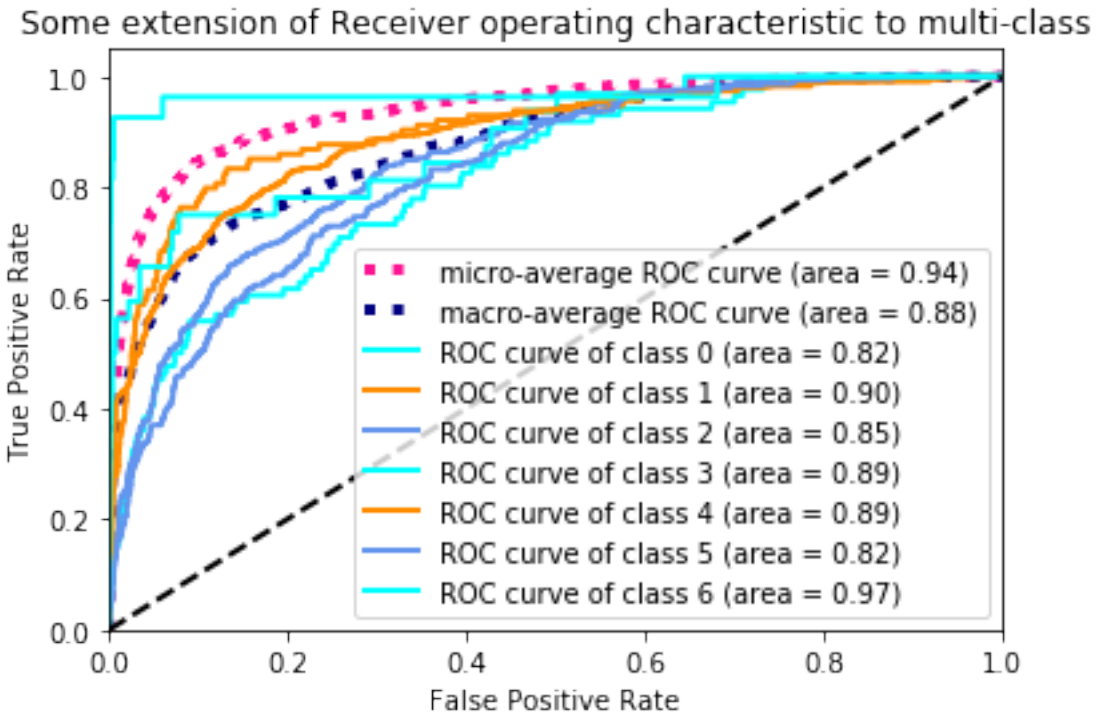
```
balanced_accuracy_score(test_class, predict_class)
```

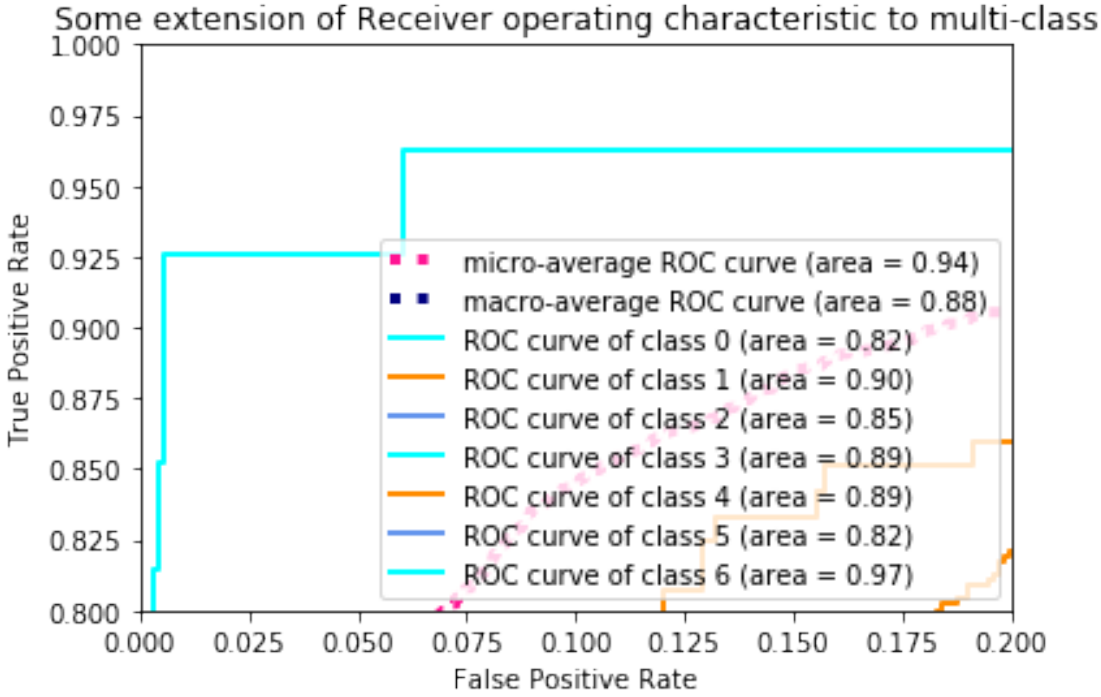
Out[13]: 0.5516761672441042

In [14]:
```
import numpy as np
from scipy import interp
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn.metrics import roc_curve, auc

# Plot linewidth.
lw = 2
n_classes = 7

# Compute ROC curve and ROC area for each class
fpr = dict()
```

```
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(1)
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()


# Zoom in view of the upper left corner.
plt.figure(2)
plt.xlim(0, 0.2)
plt.ylim(0.8, 1)
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```


Some extension of Receiver operating characteristic to multi-class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.94)
macro-average ROC curve (area = 0.88)
ROC curve of class 0 (area = 0.82)
ROC curve of class 1 (area = 0.90)
ROC curve of class 2 (area = 0.85)
ROC curve of class 3 (area = 0.89)
ROC curve of class 4 (area = 0.89)
ROC curve of class 5 (area = 0.82)
ROC curve of class 6 (area = 0.97)


Some extension of Receiver operating characteristic to multi-class
True Positive Rate
False Positive Rate
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
micro-average ROC curve (area = 0.94)
macro-average ROC curve (area = 0.88)
ROC curve of class 0 (area = 0.82)
ROC curve of class 1 (area = 0.90)
ROC curve of class 2 (area = 0.85)
ROC curve of class 3 (area = 0.89)
ROC curve of class 4 (area = 0.89)
ROC curve of class 5 (area = 0.82)
ROC curve of class 6 (area = 0.97)

```
In [ ]: from collections import Counter
        from sklearn.datasets import make_classification
        from imblearn.over_sampling import SMOTE # doctest: +NORMALIZE_WHITESPACE
        X, y = make_classification(n_classes=7, class_sep=2, weights=[0.1, 0.9], n_informative=
        print('Original dataset shape %s' % Counter(y))
        Original dataset shape Counter({1: 900, 0: 100})
        sm = SMOTE(random_state=42)
        X_res, y_res = sm.fit_resample(X, y)
        print('Resampled dataset shape %s' % Counter(y_res))
        Resampled dataset shape Counter({0: 900, 1: 900})
```

0.4 Discussion

Although 83% is generally good score but it is not the case with health care. In this industry the accuracy need to be far more accurate the current result namely 98%-99%

Probably the issue is not on the CNN itself but more to the data sample. Data sample very highly imbalanced and some class didn't have enough sample to make a good prediction

Also several skin lesion types are similar which even an expert can hardly recognise it correctly.

0.4.1 Future opportunity and implications

So much room for improvement especially for the data sample, with more sample the model can learn better. On the other hand the model can be enhanced to classify another type of healthy issue.

With integration with mobile device (smartphone camera), people don't need to come to hospital to get first evaluation about their disease.

0.4.2 Reflection

This project help me so much to understand one of CNN and ways to implement it.

The biggest issue on developing machine learning model (especially deep learning), it is resources hungry process (memory, cpu, gpu) so it become so slow when running on laptop or old computer.

Also Deep learning like CNN make me care more about the data quality, sometimes I missed how traditional machine learning works. Feature engineering can give exposure about the problem domain it self.

0.5 Conclusion

The CNN perform very well with the image classification although the result still below the standard and expectation.

0.6 Reference

Tschandl, P. et al. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5:180161 doi: 10.1038/ sdata.2018.161 (2018).

Esteva, A., Kuprel, B., Novoa, R., Ko, J., Swetter, S., Blau, H. and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), pp.115-118.

Huang, G., Liu, Z., Van der Maaten, L., & Weinberger, K. (2016). Densely Connected Convolutional Networks.

Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2011). SMOTE: Synthetic Minority Oversampling Technique. 16, 321-357.

Challenge2018.isic-archive.com. (2019). ISIC 2018 | ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection. [online] Available at: https://challenge2018.isic-archive.com/ [Accessed 25 Jan. 2019].

Kaggle.com. (2019). Dermatology Image Classification | Kaggle. [online] Available at: https://www.kaggle.com/yuningalexliu/dermatology-image-classification [Accessed 25 Jan. 2019].

Kaggle.com. (2019). Dermatology MNIST: Loading and Processing | Kaggle. [online] Available at: https://www.kaggle.com/kmader/dermatology-mnist-loading-and-processing [Accessed 25 Jan. 2019].

Bhf.org.uk. (2019). BHF-Turing Cardiovascular Data Science Awards. [online] Available at: https://www.bhf.org.uk/for-professionals/information-for-researchers/what-we-fund/bhf-turing-cardiovascular-data-science-awards [Accessed 25 Jan. 2019].

Cs231n.github.io. (2019). CS231n Convolutional Neural Networks for Visual Recognition. [online] Available at: http://cs231n.github.io/convolutional-networks/ [Accessed 25 Jan. 2019].