# Skin Lesion Analysis

Bruno Sousa
up201604145@fe.up.pt
João Gonçalves
up201604245@fe.up.pt
Pedro Silva
up201604470@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

## Abstract

For this study, we implemented a model that was able to perform an automatic classification of skin lesions from dermoscopic images. This challenge can be divided into two sub-challenges, those being: Malignant or Benign image classification and Multi-class image classification. We were provided, by ISIC Challenges, with data sets with several images to use for training and for testing. For these challenges, multiple strategies were attempted, compared and critically analysed. Among those strategies we used Oriented FAST and Rotated BRIEF (ORB), Bag of Visual Words (BoVW), Support Vector Machines (SVM) and Convolutional Neural Networks (CNN).

## 1 Introduction

Skin cancer, the abnormal growth of skin cells, is one of the most common cancers causing thousands of deaths each year. This disease often starts as changes to the skin, changes that are not cancer but could become cancer over time.

Skin changes can be malignant (overgrowth of abnormal cells that divide without control and order) and cause cancer or they can be benign (normal cells that divide and grow too much, but do not interfere with the function of normal cells around them) and not cancerous. [1]

These changes are usually detected but people do not take the necessary appropriate precautions, since many of them are benign.

Since an early detection of skin cancer gives the greatest chance for successful skin cancer treatment, an application that could distinguish malignant and benign tumors with image classification, would help people to better track the changes they find in their skin and prevent them from turning into cancer if they are malignant.

As the number of cases of skin cancer continue to increase globally each year, many research works are being done using Computer vision and Image processing to detect malicious melanoma. This research brought the development of medical image analysis approaches that can display remarkable accuracy in the recognition of malignant tumors using deep learning techniques. [4] Due to this, various clinics and dermatologists have incorporated these systems to detect skin problems with their patients. [5]

## 2 Methodology

### 2.1 Malign/Benign image classification

For the classification of lesions in dermoscopy images, two different approaches were studied and implemented: dictionary-based representation, using Bag of Visual Words, along with a Support Vector Machine (SVM) classifier; deep learning method using a Convolutional Neural Network (CNN) for both feature extraction and classification.

#### 2.1.1 Bag of Visual Words (BoVW)

OpenCV was used for reading the images and converting them to grayscale. Afterwards, a feature extraction algorithm was used to retrieve the image descriptors (ORB). The descriptors of the image are then added to the BoVW. Finally, we use the BoVW to train an SVM which is then used to classify new images, after they have been transformed into BoVWs. Before settling with this architecture, multiple feature extraction algorithms were tried: ORB, SIFT, FAST and BRISK. After testing these algorithms, ORB was chosen because it had the best results. In this process, there are two parameters that can be optimized: the number of clusters used to calculate different BoVWs and the number of iterations of the SVM training. The number of training iterations was chosen to allow the SVM model to converge.

#### 2.1.2 Convolutional Neural Network (CNN)

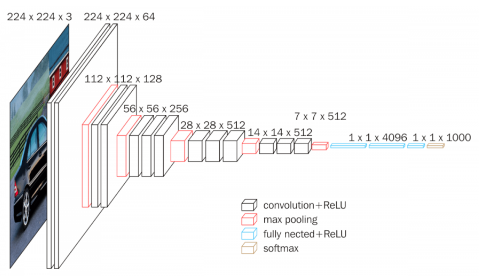

Figure 1: VGG16 Convolutional Network[6]

Due to the time constraints, it was not possible to train a feature extraction CNN for this project. Therefore, we decided to use a well-known architecture, VGG16, with its weights already tuned. This was accomplished by transfer learning the weights of an identical network trained extensively and with great results: ImageNet. The weights from ImageNet are optimized to detect 1000 different classes which are very different from our goal of skin lesions detection. However, this process can be used as a feature detector since its layers have proven to be great at finding nuances in images and translating them to features. Finally, and to seize its capabilities to a maximum, we appended to the end of the VGG16 a smaller CNN which takes as Input the Output of the VGG16, processes its features and then classifies the images as Malign or Benign. Since the VGG16 is already trained, its weights are frozen during the training of the classifier making it a faster and more reliable strategy.

Other strategies were considered: different architectures (GoogLeNet and Residual Network) and using different weights when transfer learning, this includes other generic training weights but also other models that are more focused in skin lesions which would work much better as a feature extractor.

The smaller CNN added to the VGG16 network has the following architecture: one dense layer with 16 nodes using a rectified linear unit (ReLU) as the activation function, one dropout layer to avoid over fitting and a final dense layer with 2 nodes, one for each class, using a sigmoid as the activation function.

Finally, the training should be fine tuned in order to get the best results, this can be achieved by changing various variables such as the classifier architecture, the number of epochs, the optimizer function, its learning rate and the loss function. Moreover, and if possible, the resolution of the network can also be increase to get more detail.

### 2.2 Multi-class image classification

For the classification of image in different multi-class images, an adaption of the CNN used for this malign/beningn classification was used. Since this task required a more varied understanding of the problem, as it contains more classes, the first dense layer was changed to include 256 nodes. Also, the last layer was redesigned to have 7 output nodes with a softmax activation function.

## 3 Datasets

The dataset used for the Malign/Benign image classification was taken from a ISIC 2016 challenge[2], while the dataset used for the Multi-class image classification was taken from a ISIC 2018 challenge[3].

### 3.1 Malign/Benign image classification

| | Malign | Benign | Total |
|---|---|---|---|
| Train Set | 173 | 727 | 900 |
| Test Set | 75 | 304 | 379 |

Table 1: Distribution of classes.

This dataset was highly unbalanced, having an high proportion of benign classes, which became a problem to the training of the CNN. To tackle it, we used data augmentation to increase the number of available image by doing multiple random transformations, such as vertical and horizontal flips, to the training images. Also, to encourage the network to learn both classes, different weights were attributed to both classes in order to balance the learning of them both.

The training dataset was split in training and validation for the training of the CNN, with an 80%/20% distribution.

### 3.2 Multi-class image classification

| AKIEC | BCC | BKL | DF | NV | MEL | VASC | Total |
|---|---|---|---|---|---|---|---|
| 327 | 514 | 1099 | 115 | 6705 | 1113 | 142 | 10015 |

Table 2: Distribution of classes.

This dataset also had problems with the highly unbalance of classes, especially the high number of *Melanocytic nevus* entries. To tackle this problem, the same solutions explained on the previous dataset were also used.

As the dataset did not contain a separation between train and test sets, we split the given dataset into 70% for training and 30% for testing.

## 4 Results

### 4.1 Malign/Benign image classification

#### 4.1.1 Bag of Visual Words (BoVW)

| | | Predicted | |
|---|---|---|---|
| | | Benign | Malign |
| Actual | Benign | 215 | 89 |
| | Malign | 58 | 17 |

Table 3: Confusion Matrix.

**Test Accuracy:** 61.21%

#### 4.1.2 Convolutional Neural Network (CNN)

Along the 100 epochs of training, the accuracy of both the training set and the validation set increased from an initial value of around 30% up to almost 60%.

To make sure the model was not over fitting, the model loss was also taken into account and monitored. In this instance, both the train as well as validation set loss decreased over time approximately at the same rate.

| | | Predicted | |
|---|---|---|---|
| | | Benign | Malign |
| Actual | Benign | 242 | 62 |
| | Malign | 54 | 21 |

Table 4: Confusion Matrix.

**Test accuracy:** 69.39%

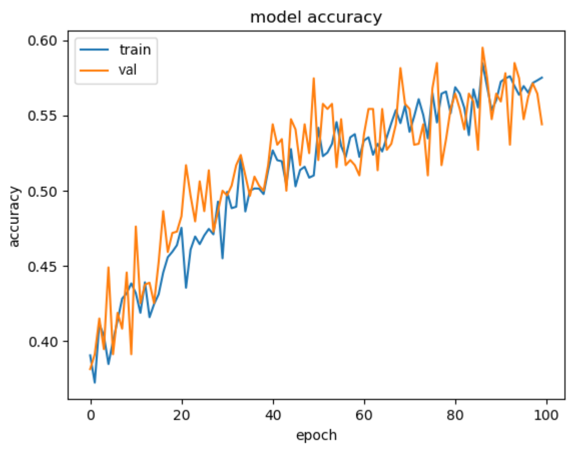

Figure 2: CNN Accuracy Graph

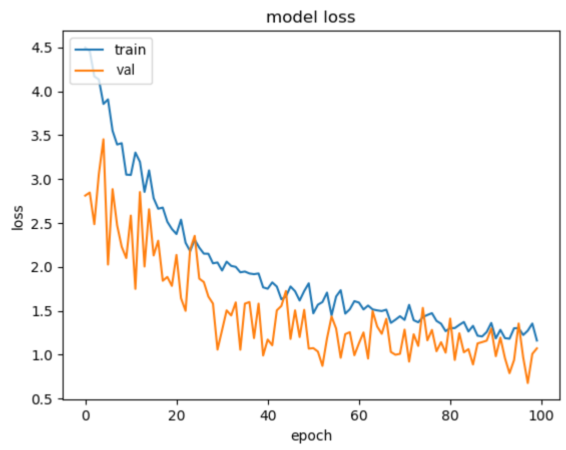

Figure 3: CNN Loss Graph

### 4.2 Multi-class image classification

| | | Predicted | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | AKIEC | BCC | BKL | DF | NV | MEL | VASC |
| Actual | AKIEC | 0 | 1 | 6 | 0 | 78 | 13 | 0 |
| | BCC | 1 | 2 | 7 | 0 | 133 | 11 | 0 |
| | BKL | 4 | 9 | 24 | 0 | 272 | 18 | 3 |
| | DF | 0 | 1 | 2 | 0 | 30 | 2 | 0 |
| | NV | 21 | 32 | 107 | 0 | 1724 | 116 | 12 |
| | MEL | 4 | 9 | 25 | 0 | 276 | 19 | 1 |
| | VASC | 0 | 1 | 3 | 0 | 36 | 3 | 0 |

Table 5: Confusion Matrix.

**Test Accuracy:** 58.85%

## 5 Performance

Generally speaking, our work did not meet our expectations. Despite having a test accuracy of well over 50%, which would beat a coin flip, the confusion matrix shows that this result is mostly a consequence of the disparity in examples of each class.

### 5.1 Malign/Benign image classification

By comparing both architectures we observe that CNN has a better performance than BoVW. Both models learned to predict similarly, but CNN has had better results in part due to the data augmentation applied for this method. Furthermore, CNN is the approach that can easily and more completely be improved.

#### 5.1.1 Bag of Visual Words (BoVW)

While the accuracy seems high for BoVW, we can see that it is mostly predicting the images to be benign, while occasionally predicting them to be malign, which hints that the model isn't correctly able to predict malign and only has an high accuracy because the test classes are unbalanced.

#### 5.1.2 Convolutional Neural Network (CNN)

The graphs show a good tendency during the training, both training and validation accuracy steadily increase showing that the model is improving over time. Furthermore, both training and validation loss decrease over time, showing that the knowledge being learnt using the training data is reflected in the validation set (which the system has never seen before). Moreover, it shows that the system is not being over fitted which would be shown by an increase of validation loss and a decrease of training loss meaning the network would be memorizing the data seen.

However, and despite of the apparent good accuracy of 70%, the confusion matrices, both from the training and from the testing set, show that the main thing that the system is learning is which is the best rate at which it should guess it is a benign image. This can be seen in Table 4: the number of true positives (maligns images being detected as malign) is smaller than the number of benign images being classified as malign. This result hints for an unbalanced training data set even though we used both data augmentation, weight balancing and data sub sampling strategies to try to fix this problem.

To tackle these issues, we attempted to vary all the variables stated in the Methodology section according to the behaviour of the network: decrease of architecture complexity when the system was over fitting (and the architecture was too complex), tune the learning rate when the network was converging either too quickly or too slowly, increase the number of epochs when the network showed it had a good trend, among other strategies.

### 5.2 Multi-class image classification

For this problem, the same analysis can be applied as the one on the previous section. The results show us that the model is mostly trying to predict the most represented class, while sometimes learning to predict other classes seemingly at random.

We can also see that, for this problem, the model never predicted an image to be *Dermatofibroma* and it never correctly predicted it to be *Actinic keratosis* or *Vascular lesion*.

## 6 Conclusions

In this study, we realize this field of work, since its statistics based, is hard to grasp for a beginner. Knowing how to fine tune these systems is something that only an experienced person can do easily in a timely fashion.

To improve the results of our network used for the first task, we aim to increase the quality of the images used as input. The images are rescaled automatically which means that the amount of information being lost before the image even gets to the network is dependent on the size of the points of interest. In a future work, there should be more image preprocessing to make sure these images are more standardized and easier to compare. Be aware that this change could imply that the network would be less resistant to not pre-processed images, if the goal of the system is to work in real time, then the pre-process work should be instant. Furthermore, we aim to tune the architecture of the network by removing the last layer of the VGG16, this would be an attempt to make our feature extractor more abstract since it no longer has the last layer which classifies the image as one of 1000 specific classes. This specification might be making us loose precious information in the feature extraction stage.

To sum up, although we did not achieve the best results, we learned a lot about how to deal with neural networks and how to understand the metrics of both the training and the results in order to improve the overall success of the system.

## References

[1] AIM at Melanoma Foundation. Benign or malignant. https://www.aimatmelanoma.org/about-melanoma/other-lesions/benign-or-malignant/.

[2] ISIC Challenges. Malign/benign image classification, 2016. https://challenge.kitware.com/phase/5667455bcad3a56fac786791.

[3] ISIC Challenges. Multi-class image classification, 2018. https://challenge2018.isic-archive.com/task3/.

[4] Wael A. Mohamed Fatma Sherif and A.S. Mohra. Skin lesion analysis toward melanoma detection using deep learning techniques, 2019. https://www.researchgate.net/publication/336472732_Skin_Lesion_Analysis_towards_melanoma_Detection_using_Deep_Learning_Techniques.

[5] Darrell S Rigel Natalie Tucker Richard R Winkelmann, Gregory Nikolaidis and Laura Speck. Comparison of the distribution of morphological disorganization of pigmented lesions in a community-based practice versus a university-based clinical setting as measured by a multispectral digital skin lesion analysis device: Impact on diagnosis, 2015. https://pubmed.ncbi.nlm.nih.gov/25741398/.

[6] Muneeb ul Hassan. Vgg16 – convolutional network for classification and detection. https://neurohive.io/en/popular-networks/vgg16/.

# 7 Appendix

## 7.1 task1/orb.py

```
import cv2 as cv
import csv
import numpy as np
from scipy.cluster.vq import kmeans, vq
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, average_precision_score
from joblib import dump, load
import os.path
from os import path

orb = cv.ORB_create()

def loadFeatures(datasetPath):
    descriptors = []
    with open(datasetPath + '.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            img = cv.imread(datasetPath + '/' + row[0] + '.jpg')
            gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
            kp, des = orb.detectAndCompute(gray,None)
            if des is None:
                continue
            descriptors.append((des, row[1]))
    return descriptors

def generateImFeatures(dataset):
    descriptors = dataset[0][0]
    for descriptor,malign in dataset[1:]:
        descriptors = np.vstack((descriptors, descriptor))

    descriptors_float = descriptors.astype(float)

    k = 400
    voc, variance = kmeans(descriptors_float, k, 1)

    im_features = np.zeros((len(dataset), k), "float32")
    for i in range(len(dataset)):
        words, distance = vq(dataset[i][0],voc)
        for w in words:
            im_features[i][w] += 1

    nbr_occurences = np.sum( (im_features > 0) * 1, axis = 0)
    idf = np.array(np.log((1.0*len(dataset)+1) / (1.0*nbr_occurences + 1)), 'float32')

    stdSlr = StandardScaler().fit(im_features)
    im_features = stdSlr.transform(im_features)

    return im_features

def generateModel(dataset):
    for descriptor,malign in dataset:
        maligns.append(malign)

    im_features = generateImFeatures(dataset)

    clf = LinearSVC(max_iter=1000000)
    clf.fit(im_features, maligns)

    return clf

if not path.exists('orb.joblib'):
    train = loadFeatures('dataORB/train')

test = loadFeatures('dataORB/test')
maligns = []

if path.exists('orb.joblib'):
    clf = load('orb.joblib')
else:
    clf = generateModel(train)

dump(clf, 'orb.joblib')

im_features = generateImFeatures(test)

result = clf.predict(im_features)

hit = 0
miss = 0

for descriptor,malign in test:
    maligns.append(malign)

for i in range(len(result)):
    if result[i] == maligns[i]:
        hit+=1
    else:
        miss+=1

print(result)
print("Accuracy: " + str(hit * 100 / (hit+miss)) + "%")
```

```
cm = confusion_matrix(maligns, result)
print (cm)
print("Average Precision Score: " + str(average_precision_score(list(map(float,maligns)),list(map(float,result)))))
```

## 7.2 task1/cnn.py

```
# Disable GPU
# import os
# os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

from keras.models import Model
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping, ModelCheckpoint
import sys
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, average_precision_score
from sklearn.utils import class_weight
import matplotlib.pyplot as plt


img_width, img_height = 224, 224

train_data_dir = 'data/train/'
test_data_dir = 'data/test/'
epochs = 100
batch_size = 32

model = applications.VGG16(weights='imagenet', include_top=False,
                           input_shape=(224, 224, 3))

for layer in model.layers:
    layer.trainable = False

print('Model loaded.')

top_model = Sequential()
top_model.add(Flatten(input_shape=model.output_shape[1:]))
top_model.add(Dense(16, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(2, activation='sigmoid'))

model = Model(inputs= model.input, outputs= top_model(model.output))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.SGD(lr=1e-7, momentum=0.9),
              metrics=['accuracy'])

model.summary()

train_datagen = ImageDataGenerator(
    # shear_range=0.2,
    # zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2)

test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

class_weights = class_weight.compute_class_weight(
        'balanced',
        np.unique(train_generator.classes),
        train_generator.classes)

history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples// batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    class_weight=class_weights)#,
    #callbacks=[EarlyStopping(monitor='val_accuracy', patience=5)])

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

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

X_pred = model.predict_generator(train_generator, train_generator.samples // batch_size + 1)
Y_pred = model.predict_generator(test_generator, test_generator.samples // batch_size + 1)
x_pred = np.argmax(X_pred, axis=1)
y_pred = np.argmax(Y_pred, axis=1)

print('Train Confusion Matrix')
print(confusion_matrix(train_generator.classes, x_pred))
print('Classification Report')
target_names = ['Benign', 'Malign']
print(classification_report(train_generator.classes, x_pred, target_names=target_names))
print(x_pred)


print('Test Accuracy: ' + (str) (100 * accuracy_score(test_generator.classes, y_pred, normalize=True)) + '%')
print('Test Average Precision: ' + (str) (average_precision_score(test_generator.classes, y_pred)) + '%')
print('Test Confusion Matrix')
print(confusion_matrix(test_generator.classes, y_pred))
print('Classification Report')
target_names = ['Benign', 'Malign']
print(classification_report(test_generator.classes, y_pred, target_names=target_names))
print(y_pred)

print('saving model...')
model.save('model.h5')
```

## 7.3 task2/cnn.py

```
# Disable GPU
# import os
# os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

from keras.models import Model
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping, ModelCheckpoint
import sys
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.utils import class_weight
import matplotlib.pyplot as plt


img_width, img_height = 224, 224

train_data_dir = 'data/train/'
test_data_dir = 'data/test/'
epochs = 100
batch_size = 16

model = applications.VGG16(weights='imagenet', include_top=False,
                           input_shape=(224, 224, 3))

for layer in model.layers:
    layer.trainable = False

print('Model loaded.')

top_model = Sequential()
top_model.add(Flatten(input_shape=model.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(7, activation='softmax'))

model = Model(inputs= model.input, outputs= top_model(model.output))

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=1e-5, momentum=0.9),
              metrics=['accuracy'])

model.summary()

train_datagen = ImageDataGenerator(
    #shear_range=0.2,
    #zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2)

test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

class_weights = class_weight.compute_class_weight(
        'balanced',
        np.unique(train_generator.classes),
        train_generator.classes)

history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples// batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    class_weight=class_weights)#,
    # callbacks=[EarlyStopping(monitor='val_accuracy', patience=5)])

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

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

X_pred = model.predict_generator(train_generator, train_generator.samples // batch_size + 1)
Y_pred = model.predict_generator(test_generator, test_generator.samples // batch_size + 1)
x_pred = np.argmax(X_pred, axis=1)
y_pred = np.argmax(Y_pred, axis=1)

target_names = ['MEL','NV','BCC','AKIEC','BKL','DF','VASC']

print('Train Confusion Matrix')
print(confusion_matrix(train_generator.classes, x_pred))
print('Classification Report')
print(classification_report(train_generator.classes, x_pred, target_names=target_names))
print(x_pred)

print('Test Accuracy: ' + (str) (100 * accuracy_score(test_generator.classes, y_pred, normalize=True)) + '%')
print('Test Confusion Matrix')
print(confusion_matrix(test_generator.classes, y_pred))
print('Classification Report')
print(classification_report(test_generator.classes, y_pred, target_names=target_names))
print(y_pred)

print('saving model...')
model.save('model.h5')
```