National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Krishnachander Rajendran
Student ID: x18122779

School of Computing
National College of Ireland

Supervisor: Dr. Bahman Honari

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Krishnachander Rajendran		
Student ID:	X18122779		
Programme:	Data Analytics	**Year:**	2019/2020
Module:	MSc Research Project		
Lecturer:	Dr. Bahman Honari		
Submission Due Date:	12/12/2019		
Project Title:	Performance optimization on skin lesion image classification using pre-trained models		
Word Count:	363	**Page Count:**	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

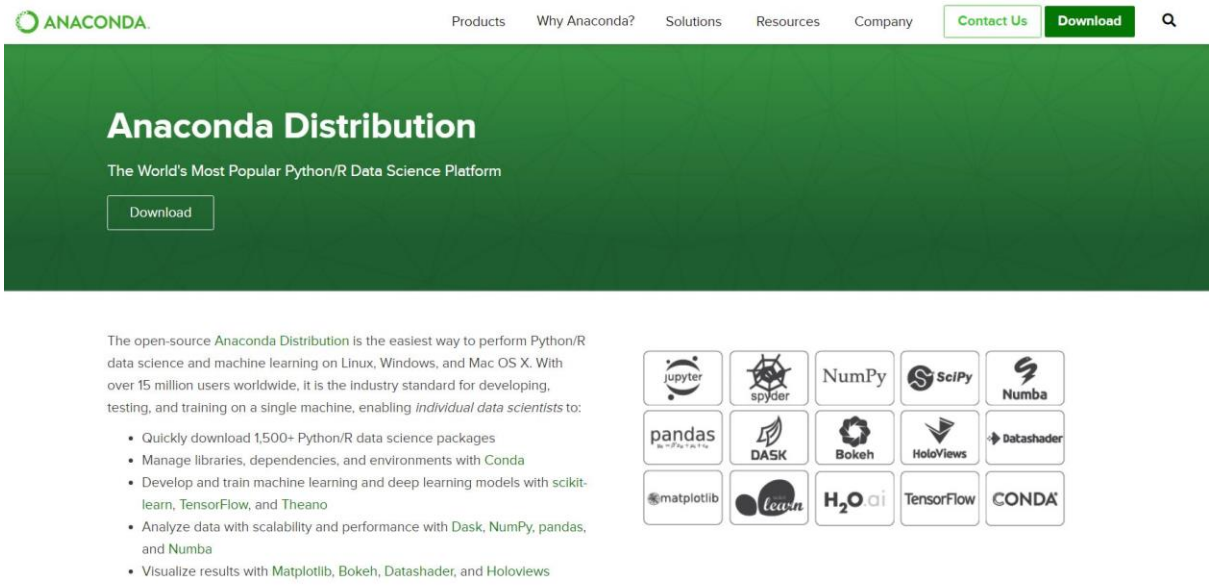
Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

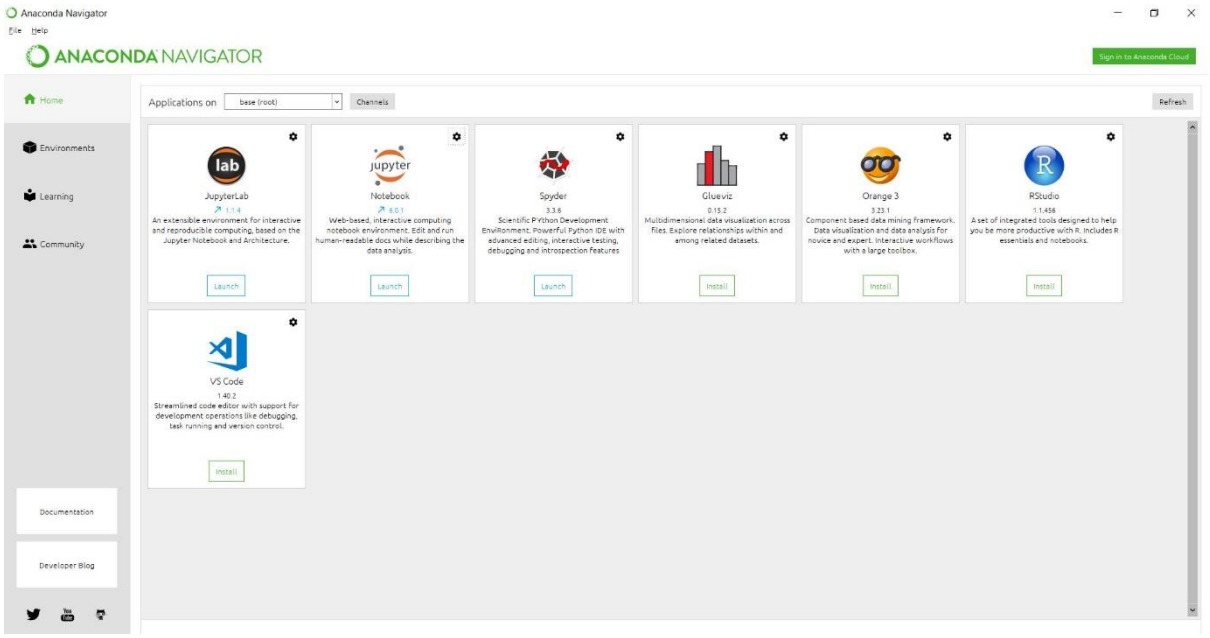
Krishnachander Rajendran
Student ID: x18122779

1 Software Requirements

Your first section. Change the header and label to something appropriate.
The codes for the research study were written entirely using Python 3 version. Python codes were implemented using the Jupyter Notebook on Anaconda Navigator platform. The first step that needs to be done will be in installing the Anaconda Software. As requirements go by, the 64-bit version of the setup file is downloaded to be in compatible with the 64-bit Windows OS.



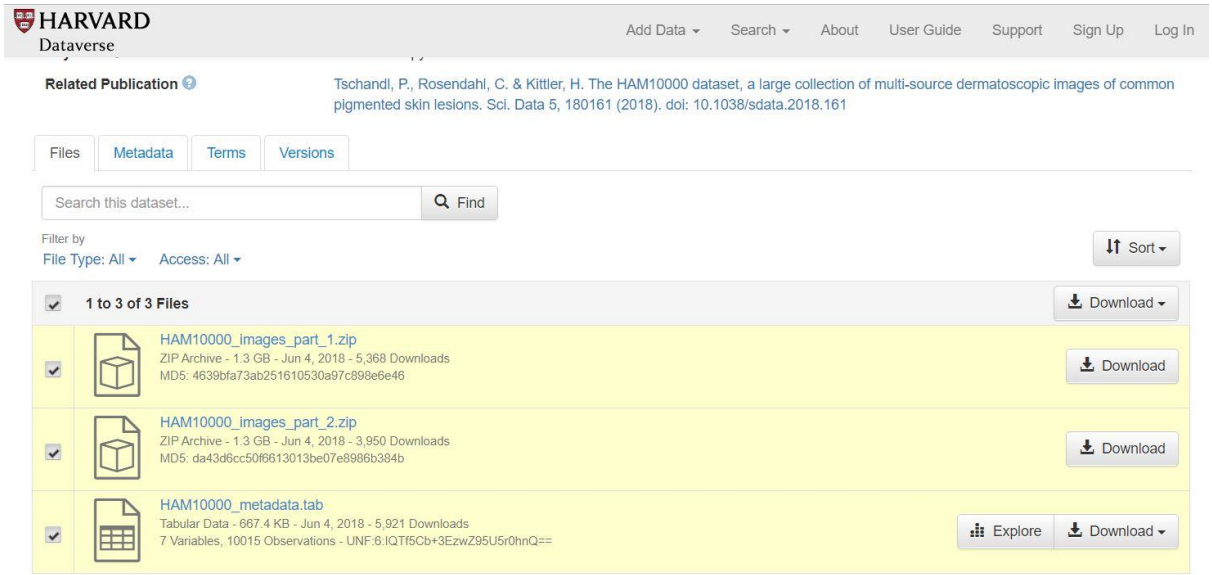
After successful installation, the Anaconda Navigator is opened from **Start -> Anaconda Navigator**



Jupyter Notebook is launched and is launched via google chrome browser as its base platform.

2 Data Extraction

The skin lesion image dataset of HAM10000 (Human-Against-Machine 10000) was downloaded from the Dataverse website of Harvard University (Tschandl et al., 2018).



As the dataset consists of a skewed representation of images in multiple classes, image augmentation becomes necessary to reduce the degree of skewness in image class representation.

3 Data Preprocessing

Launch Jupyter Notebook from **Start -> Anaconda Navigator** to open up the browser based python platform. Execute the python file with the pre-processing and data augmentation code.

4 Library Installation

A few libraries need to be installed as part of the requirements for the CNN for image classification task. Tensorflow is installed to begin satisfying the environment for CNN architecture.

```
conda create -n tf tensorflow
conda activate tf
```

Tensorflow includes the packages required for building the models discussed in this research.

5 Code Listing

Data pre-processing and augmentation

```
from numpy.random import seed
seed(101)
from tensorflow import set_random_seed
set_random_seed(101)

import pandas as pd
import numpy as np
#import keras
#from keras import backend as K

import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

import os

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline

#Libraries for building the model
import keras
from keras import backend as K
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense,Flatten
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import *
from matplotlib import pyplot as plt
import itertools
import matplotlib.pyplot as plt
%matplotlib inline
from keras import optimizers
```

```
Using TensorFlow backend.
```

```
# Create a new directory
base_dir = 'G:\\Skin Cancer image dataset\\base_dir'
os.mkdir(base_dir)


#[CREATE FOLDERS INSIDE THE BASE DIRECTORY]

# now we create 7 folders inside 'base_dir':

# train_dir
    # nevi
    # melanoma
    # bkerat
    # basalcarc
    # akerat
    # vascular
    # defibr

# val_dir
    # nevi
    # melanoma
    # bkerat
    # basalcarc
    # akerat
    # vascular
    # defibr

# create a path to 'base_dir' to which we will join the names of the new folders
# train_dir
train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)

# val_dir
val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)
```

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

# create new folders inside train_dir
nv = os.path.join(train_dir, 'nv')
os.mkdir(nv)
mel = os.path.join(train_dir, 'mel')
os.mkdir(mel)
bkl = os.path.join(train_dir, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(train_dir, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(train_dir, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(train_dir, 'vasc')
os.mkdir(vasc)
df = os.path.join(train_dir, 'df')
os.mkdir(df)

# create new folders inside val_dir
nv = os.path.join(val_dir, 'nv')
os.mkdir(nv)
mel = os.path.join(val_dir, 'mel')
os.mkdir(mel)
bkl = os.path.join(val_dir, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(val_dir, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(val_dir, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(val_dir, 'vasc')
os.mkdir(vasc)
df = os.path.join(val_dir, 'df')
os.mkdir(df)
```

```
df_data = pd.read_csv('HAM10000_metadata.csv')

df_data.head()
```

```
# this will tell us how many images are associated with each lesion_id
df = df_data.groupby('lesion_id').count()

# now we filter out lesion_id's that have only one image associated with it
df = df[df['image_id'] == 1]

df.reset_index(inplace=True)

df.head()
```

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000001	1	1	1	1	1	1
1	HAM_0000003	1	1	1	1	1	1
2	HAM_0000004	1	1	1	1	1	1
3	HAM_0000007	1	1	1	1	1	1
4	HAM_0000008	1	1	1	1	1	1

```
# here we identify lesion_id's that have duplicate images and those that have only
# one image.

def identify_duplicates(x):

    unique_list = list(df['lesion_id'])

    if x in unique_list:
        return 'no_duplicates'
    else:
        return 'has_duplicates'

# create a new colum that is a copy of the lesion_id column
df_data['duplicates'] = df_data['lesion_id']
# apply the function to this new column
df_data['duplicates'] = df_data['duplicates'].apply(identify_duplicates)

df_data.head()
```

```
df_data['duplicates'].value_counts()
```

```
no_duplicates     5514
has_duplicates    4501
Name: duplicates, dtype: int64
```

```
# now we filter out images that don't have duplicates
df = df_data[df_data['duplicates'] == 'no_duplicates']

df.shape
```

```
(5514, 8)
```

```
# now we create a val set using df because we are sure that none of these images
# have augmented duplicates in the train set
y = df['dx']

_, df_val = train_test_split(df, test_size=0.17, random_state=101, stratify=y)

df_val.shape
```

```
(938, 8)
```

```
df_val['dx'].value_counts()
```

```
nv       751
bkl       75
mel       39
bcc       30
akiec     26
vasc      11
df         6
Name: dx, dtype: int64
```

```
# This set will be df_data excluding all rows that are in the val set

# This function identifies if an image is part of the train
# or val set.
def identify_val_rows(x):
    # create a list of all the lesion_id's in the val set
    val_list = list(df_val['image_id'])

    if str(x) in val_list:
        return 'val'
    else:
        return 'train'

# identify train and val rows

# create a new colum that is a copy of the image_id column
df_data['train_or_val'] = df_data['image_id']
# apply the function to this new column
df_data['train_or_val'] = df_data['train_or_val'].apply(identify_val_rows)

# filter out train rows
df_train = df_data[df_data['train_or_val'] == 'train']


print(len(df_train))
print(len(df_val))
```

```
9077
938
```

```
df_train['dx'].value_counts()
```

```
nv       5954
mel      1074
bkl      1024
bcc       484
akiec     301
vasc      131
df        109
Name: dx, dtype: int64
```

```
df_val['dx'].value_counts()
```

```
nv       751
bkl       75
mel       39
bcc       30
akiec     26
vasc      11
df         6
Name: dx, dtype: int64
```

```
# Set the image_id as the index in df_data
df_data.set_index('image_id', inplace=True)
```

```
#train_dir = os.path.join(base_dir, 'train_dir')

# Get a list of images in each of the two folders
folder_1 = os.listdir('ham10000_images_part_1')
folder_2 = os.listdir('ham10000_images_part_2')

# Get a list of train and val images
train_list = list(df_train['image_id'])
val_list = list(df_val['image_id'])



# Transfer the train images

for image in train_list:

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

    if fname in folder_1:
        # source path to image

        src = os.path.join('ham10000_images_part_1', fname)
        # destination path to image
        dst = os.path.join(train_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

```
    if fname in folder_2:
        # source path to image
        src = os.path.join('ham10000_images_part_2', fname)
        # destination path to image
        dst = os.path.join(train_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)


# Transfer the val images

for image in val_list:

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

    if fname in folder_1:
        # source path to image
        src = os.path.join('ham10000_images_part_1', fname)
        # destination path to image
        dst = os.path.join(val_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)

    if fname in folder_2:
        # source path to image
        src = os.path.join('ham10000_images_part_2', fname)
        # destination path to image
        dst = os.path.join(val_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

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

for item in class_list:

    print(item)
    print("\n")

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

    # Choose a class
    img_class = item

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

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


    # point to a dir containing the images and not to the images themselves
    path = aug_dir
    save_path = 'base_dir/train_dir/' + img_class

    # Create a data generator
    datagen = ImageDataGenerator(
        rotation_range=180,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        vertical_flip=True,
        #brightness_range=(0.9,1.1),
        fill_mode='nearest')
```

```
    batch_size = 50

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



    # Generate the augmented images and add them to the training folders

    ############

    num_aug_images_wanted = 6000 # total number of images we want to have in each class

    ############

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

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

        imgs, labels = next(aug_datagen)

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

```
mel


Found 1074 images belonging to 1 classes.
bkl


Found 1024 images belonging to 1 classes.
bcc


Found 484 images belonging to 1 classes.
akiec


Found 301 images belonging to 1 classes.
vasc
vasc


Found 131 images belonging to 1 classes.
df


Found 109 images belonging to 1 classes.
```

```
# Check how many train images we now have in each folder.
# This is the original images plus the augmented images.


print(len(os.listdir('base_dir/train_dir/nv')))
print(len(os.listdir('base_dir/train_dir/mel')))
print(len(os.listdir('base_dir/train_dir/bkl')))
print(len(os.listdir('base_dir/train_dir/bcc')))
print(len(os.listdir('base_dir/train_dir/akiec')))
print(len(os.listdir('base_dir/train_dir/vasc')))
print(len(os.listdir('base_dir/train_dir/df')))
```

```
5954
5920
5920
5858
5217
5290
4410
```

```
# Check how many val images we have in each folder.

print(len(os.listdir('base_dir/val_dir/nv')))
print(len(os.listdir('base_dir/val_dir/mel')))
print(len(os.listdir('base_dir/val_dir/bkl')))
print(len(os.listdir('base_dir/val_dir/bcc')))
print(len(os.listdir('base_dir/val_dir/akiec')))
print(len(os.listdir('base_dir/val_dir/vasc')))
print(len(os.listdir('base_dir/val_dir/df')))
```

```
751
39
75
30
26
11
6
```

CNN with MobileNet model

```
train_path = 'base_dir/train_dir'
valid_path = 'base_dir/val_dir'

num_train_samples = len(df_train)
num_val_samples = len(df_val)
train_batch_size = 10
val_batch_size = 10
image_size = 224

train_steps = np.ceil(num_train_samples / train_batch_size)
val_steps = np.ceil(num_val_samples / val_batch_size)
```

```
datagen = ImageDataGenerator(
    preprocessing_function= \
    tf.keras.applications.mobilenet.preprocess_input)

train_batches = datagen.flow_from_directory(train_path,
                                            target_size=(image_size,image_size),
                                            batch_size=train_batch_size)

valid_batches = datagen.flow_from_directory(valid_path,
                                            target_size=(image_size,image_size),
                                            batch_size=val_batch_size)

# Note: shuffle=False causes the test dataset to not be shuffled
test_batches = datagen.flow_from_directory(valid_path,
                                            target_size=(image_size,image_size),
                                            batch_size=1,
                                            shuffle=False)
```

```
Found 38569 images belonging to 7 classes.
Found 938 images belonging to 7 classes.
Found 938 images belonging to 7 classes.
```

```
# create a copy of a InceptionV3 model

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

```
mobile.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, 225, 225, 3)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
_________________________________________________________________
conv1_relu (ReLU)            (None, 112, 112, 32)      0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)      288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32)      128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, 112, 112, 32)      0
```

```
type(mobile.layers)
```

```
list
```

```
# To find out how many layers MobileNet model has.
len(mobile.layers)
```

```
93
```

```
# CREATE THE MODEL ARCHITECTURE

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

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

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

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

```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, 225, 225, 3)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
_________________________________________________________________
conv1_relu (ReLU)            (None, 112, 112, 32)      0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)      288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32)      128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, 112, 112, 32)      0
```

```
# We need to choose how many layers we actually want to be trained.

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

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

```
# Define Top2 and Top3 Accuracy

from tensorflow.keras.metrics import categorical_accuracy, top_k_categorical_accuracy

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

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

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

```
# Get the labels that are associated with each index
print(valid_batches.class_indices)
```

```
{'akiec': 0, 'bcc': 1, 'bkl': 2, 'df': 3, 'mel': 4, 'nv': 5, 'vasc': 6}
```

```
# Add weights to try to make the model more sensitive to melanoma

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

```
filepath = "tf_mobilenet_model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_top_3_accuracy', verbose=1,
                             save_best_only=True, mode='max')

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

earlyStopping = EarlyStopping(monitor='val_top_3_accuracy', min_delta=0, patience=20, verbose=1,
                              mode='auto', baseline=None)

#callbacks_list = [checkpoint, reduce_lr]

callbacks_list = [checkpoint, reduce_lr, earlyStopping]

history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
                              class_weight=class_weights,
                    validation_data=valid_batches,
                    validation_steps=val_steps,
                    epochs=30, verbose=1,
                   callbacks=callbacks_list)

# history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
#                               class_weight=class_weights,
#                     validation_data=valid_batches,
#                     validation_steps=val_steps,
#                     epochs=30, verbose=1,
#                     callbacks=callbacks_list)
```

```
Epoch 1/30
907/908 [============================>.] - ETA: 30s - loss: 1.7417 - categorical_accuracy: 0.5125 - top_2_accuracy: 0.7187 -
top_3_accuracy: 0.8393 Epoch 1/30
 94/908 [==>...........................] - ETA: 6:15 - loss: 4.0213 - categorical_accuracy: 0.0554 - top_2_accuracy: 0.6397 -
top_3_accuracy: 0.8475
Epoch 00001: val_top_3_accuracy improved from -inf to 0.84755, saving model to tf_mobilenet_model.h5
908/908 [==============================] - 27484s 30s/step - loss: 1.7413 - categorical_accuracy: 0.5127 - top_2_accuracy: 0.
7189 - top_3_accuracy: 0.8395 - val_loss: 4.0213 - val_categorical_accuracy: 0.0554 - val_top_2_accuracy: 0.6397 - val_top_3_
accuracy: 0.8475
Epoch 2/30
907/908 [============================>.] - ETA: 0s - loss: 1.2743 - categorical_accuracy: 0.6094 - top_2_accuracy: 0.8033 - t
op_3_accuracy: 0.9077Epoch 1/30
 94/908 [==>...........................] - ETA: 6:45 - loss: 1.0943 - categorical_accuracy: 0.6173 - top_2_accuracy: 0.8284 -
top_3_accuracy: 0.9062
Epoch 00002: val_top_3_accuracy improved from 0.84755 to 0.90618, saving model to tf_mobilenet_model.h5
908/908 [==============================] - 586s 645ms/step - loss: 1.2744 - categorical_accuracy: 0.6091 - top_2_accuracy: 0.
8033 - top_3_accuracy: 0.9077 - val_loss: 1.0943 - val_categorical_accuracy: 0.6173 - val_top_2_accuracy: 0.8284 - val_top_3_
accuracy: 0.9062
```

```
# Here the the last epoch will be used.

val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = \
model.evaluate_generator(test_batches,
                         steps=len(df_val))

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

```
val_loss: 0.5601274718238453
val_cat_acc: 0.8240938
val_top_2_acc: 0.9157783
val_top_3_acc: 0.9637527
```

```
# Here the best epoch will be used.

model.load_weights('tf_mobilenet_model.h5')

val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = \
model.evaluate_generator(test_batches,
                         steps=len(df_val))

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

```
val_loss: 0.4359079395809308
val_cat_acc: 0.8507463
val_top_2_acc: 0.93070364
val_top_3_acc: 0.97121537
```

```
# display the loss and accuracy curves

import matplotlib.pyplot as plt

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

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

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


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


plt.show()
```

```
#Optimiser
sgd = tf.keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
```

```
#Compile the model
model.compile(loss='categorical_crossentropy', optimizer='sgd',
              metrics=[categorical_accuracy, top_2_accuracy, top_3_accuracy])
```

CNN with DenseNet201 model

```
datagen = ImageDataGenerator(
    preprocessing_function= \
    tf.keras.applications.densenet.preprocess_input)

train_batches = datagen.flow_from_directory(train_path,
                                            target_size=(image_size,image_size),
                                            batch_size=train_batch_size)

valid_batches = datagen.flow_from_directory(valid_path,
                                            target_size=(image_size,image_size),
                                            batch_size=val_batch_size)

# Note: shuffle=False causes the test dataset to not be shuffled
test_batches = datagen.flow_from_directory(valid_path,
                                            target_size=(image_size,image_size),
                                            batch_size=1,
                                            shuffle=False)
```

```
Found 38569 images belonging to 7 classes.
Found 938 images belonging to 7 classes.
Found 938 images belonging to 7 classes.
```

```
# create a copy of a InceptionV3 model

# densenet = tf.keras.applications.DenseNet201.DenseNet201()

densenet = tf.compat.v1.keras.applications.densenet.DenseNet201()
```

```
Downloading data from https://github.com/keras-team/keras-applications/releases/download/densenet/densenet201_weights_tf_dim_ordering_tf_kernels.h5
82526208/82524592 [==============================] - 53s 1us/step
```

```
densenet.summary()
```

```
Model: "densenet201"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
zero_padding2d (ZeroPadding2D)  (None, 230, 230, 3)  0           input_3[0][0]
__________________________________________________________________________________________________
conv1/conv (Conv2D)             (None, 112, 112, 64) 9408        zero_padding2d[0][0]
__________________________________________________________________________________________________
conv1/bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1/conv[0][0]
__________________________________________________________________________________________________
conv1/relu (Activation)         (None, 112, 112, 64) 0           conv1/bn[0][0]
__________________________________________________________________________________________________
zero_padding2d_1 (ZeroPadding2D (None, 114, 114, 64) 0           conv1/relu[0][0]
__________________________________________________________________________________________________
pool1 (MaxPooling2D)            (None, 56, 56, 64)   0           zero_padding2d_1[0][0]
__________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 64)   256         pool1[0][0]
```

```
type(densenet.layers)
```

```
list
```

```
# How many layers does InceptionV3 have?
len(densenet.layers)
```

```
709
```

```
# CREATE THE MODEL ARCHITECTURE

# Exclude the last 2 layers of the above model.
# This will include all layers up to and including global_average_pooling2d_1
x = densenet.layers[-2].output



# Create a new dense layer for predictions
# 7 corresponds to the number of classes
x = Dropout(0.25)(x)
predictions = Dense(7, activation='softmax')(x)

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

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

```
model.summary()
```

```
Model: "model_17"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
zero_padding2d (ZeroPadding2D)  (None, 230, 230, 3)  0           input_3[0][0]
__________________________________________________________________________________________________
conv1/conv (Conv2D)             (None, 112, 112, 64) 9408        zero_padding2d[0][0]
__________________________________________________________________________________________________
conv1/bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1/conv[0][0]
__________________________________________________________________________________________________
conv1/relu (Activation)         (None, 112, 112, 64) 0           conv1/bn[0][0]
__________________________________________________________________________________________________
zero_padding2d_1 (ZeroPadding2D (None, 114, 114, 64) 0           conv1/relu[0][0]
__________________________________________________________________________________________________
pool1 (MaxPooling2D)            (None, 56, 56, 64)   0           zero_padding2d_1[0][0]
__________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 64)   256         pool1[0][0]
```

```
filepath = "tf_densenet2_model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_top_3_accuracy', verbose=1,
                             save_best_only=True, mode='max')

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

earlyStopping = EarlyStopping(monitor='val_top_3_accuracy', min_delta=0, patience=20, verbose=1,
                              mode='auto', baseline=None)

#callbacks_list = [checkpoint, reduce_lr]

callbacks_list = [checkpoint, reduce_lr, earlyStopping]

history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
                              class_weight=class_weights,
                    validation_data=valid_batches,
                    validation_steps=val_steps,
                    epochs=30, verbose=1,
                   callbacks=callbacks_list)

# history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
#                               class_weight=class_weights,
#                     validation_data=valid_batches,
#                     validation_steps=val_steps,
#                     epochs=30, verbose=1,
#                    callbacks=callbacks_list)
```

Custom built CNN

```
import numpy as np
import pandas as pd
import imageio
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support
import scipy.ndimage
from scipy import misc
import skimage
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')

from tqdm import tqdm
from glob import glob
from scipy import stats
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.under_sampling import RandomUnderSampler

import keras
import tensorflow as tf
from keras.utils import to_categorical
from keras.layers import Dense, Input, Flatten, Reshape, Conv2D, MaxPool2D, concatenate, Activation, Dropout
from keras.optimizers import Adam, RMSprop
from keras.models import Model, Sequential, load_model
from keras.losses import binary_crossentropy
from keras.metrics import binary_accuracy
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
```

```
Using TensorFlow backend.
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
np.random.seed(123)

from sklearn.metrics import confusion_matrix
import itertools

import keras
from keras import regularizers
from keras.utils.np_utils import to_categorical # used for converting labels to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
```

```
os.chdir('G:\\Skin cancer dataset')
```

```
base_skin_dir = os.path.join('G:\\Skin cancer dataset')
```

```
# Merge images from both folders into one dictionary
imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join(base_skin_dir, '*', '*.jpg'))}

# This dictionary is useful for displaying more human-friendly labels later on

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'
}
```

```
# Read in the csv of metadata

tile_df = pd.read_csv(os.path.join(base_skin_dir, 'HAM10000_metadata.csv'))
tile_df.tail()
```

```
# Create some new columns (path to image, human-readable name) and review them

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.sample(5)
```

	lesion_id	image_id	dx	dx_type	age	sex	localization	path	cell_type	cell_type_idx
9725	HAM_0004376	ISIC_0024843	akiec	histo	70.0	female	face	G:\Skin cancer dataset\HAM10000_images_part_1\...	Actinic keratoses	0
6059	HAM_0003024	ISIC_0024768	nv	follow_up	35.0	female	trunk	G:\Skin cancer dataset\HAM10000_images_part_1\...	Melanocytic nevi	4
4540	HAM_0001659	ISIC_0026564	nv	follow_up	35.0	male	lower extremity	G:\Skin cancer dataset\HAM10000_images_part_1\...	Melanocytic nevi	4
3817	HAM_0004625	ISIC_0029346	nv	follow_up	40.0	male	upper extremity	G:\Skin cancer dataset\HAM10000_images_part_2\...	Melanocytic nevi	4
7914	HAM_0000443	ISIC_0034271	nv	histo	35.0	female	back	G:\Skin cancer dataset\HAM10000_images_part_2\...	Melanocytic nevi	4

```
# Get general statistics for the dataset

tile_df.describe(exclude=[np.number])
```

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

```
(32, 32, 3)    10015
Name: image, dtype: int64
```

```
y = tile_df.cell_type_idx
x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(tile_df, y, test_size=0.25)
```

```
#Model Architecture
model = Sequential()
model.add(Conv2D(96, (9, 9), input_shape = (232, 232, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (3, 3)))
model.add(Conv2D(64, (5, 5), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (3, 3)))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (3, 3)))
model.add(MaxPooling2D(pool_size = (3, 3)))
model.add(Flatten())
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dense(units = 7, activation = 'softmax'))
model.summary()
```

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

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

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

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

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

```
input_shape = (32, 32, 3)
num_classes = 7
batch_size = 32
epochs = 50
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu',kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
dropout_25 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_26 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 2304)              0
_________________________________________________________________
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

model.summary()
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
dropout_25 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_26 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_17 (Dense)             (None, 128)               295040
_________________________________________________________________
dropout_27 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_18 (Dense)             (None, 7)                 903
=================================================================
Total params: 352,263
Trainable params: 352,263
Non-trainable params: 0
_________________________________________________________________
```

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

```
Train on 7511 samples, validate on 2504 samples
Epoch 1/50
```

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

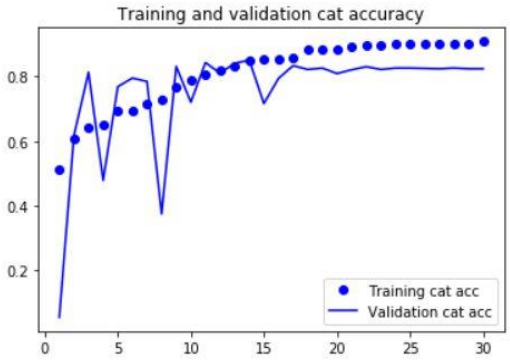
```
Test loss: 0.8282977995781091
Test accuracy: 0.7567891373801917
```

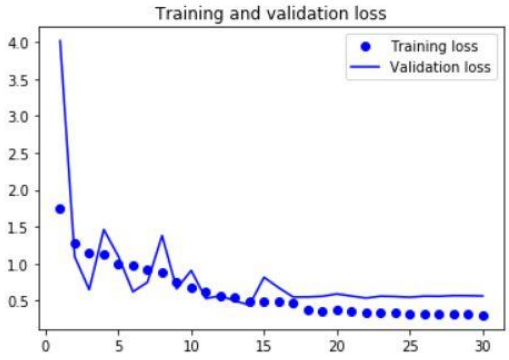
```
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="Validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['acc'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_acc'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.001),
              metrics=['accuracy'])

model.summary()
```

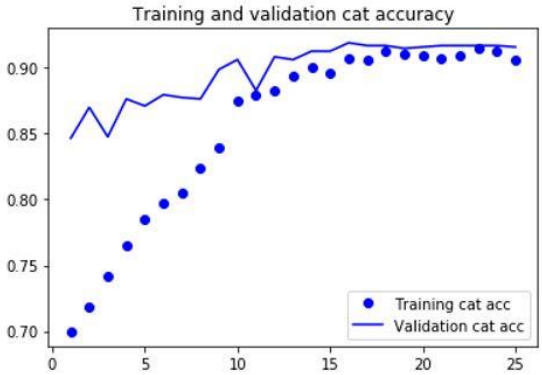
6 Results

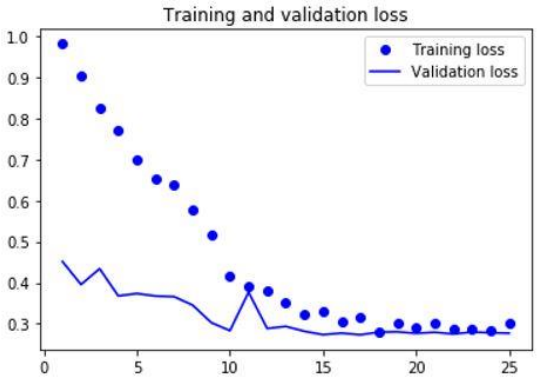
MobileNet model using Adam parameter



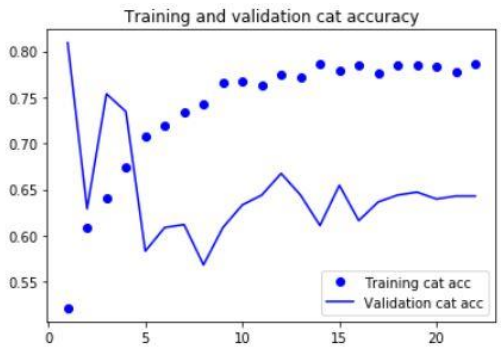


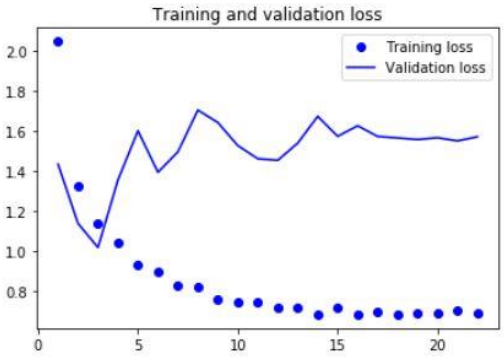
Optimization of MobileNet model using SGD parameter



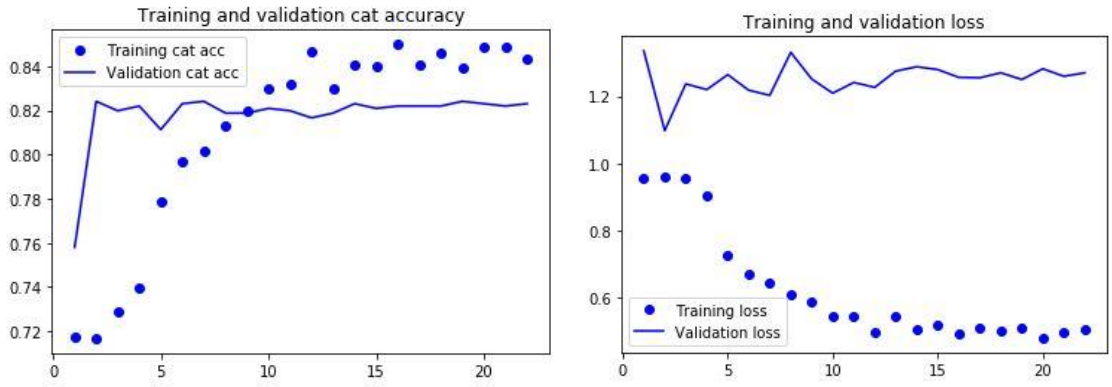


DenseNet201 using Adam parameter

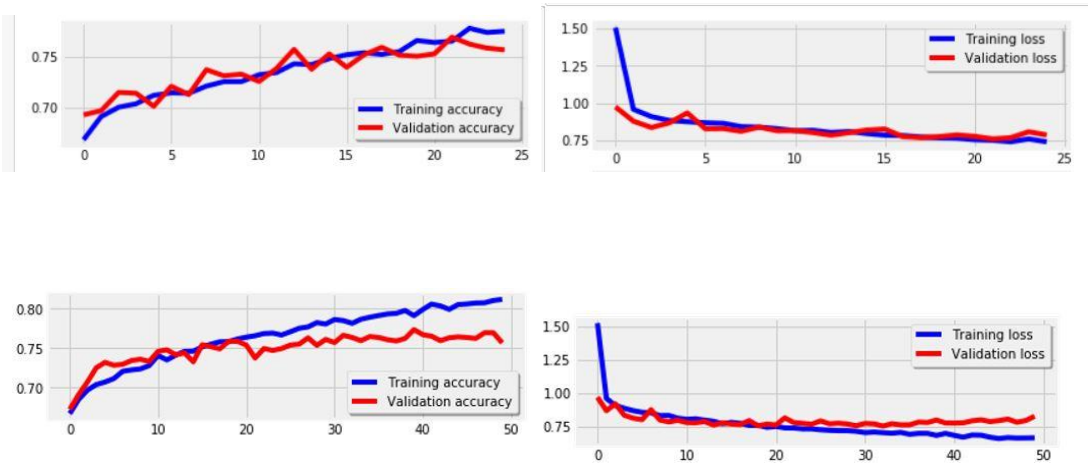




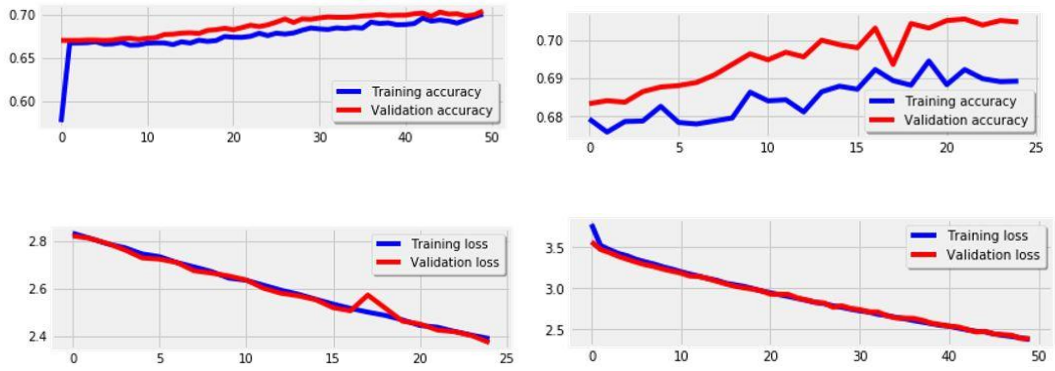
DenseNet201 using SGD parameter



Custom built CNN using Adam parameter



Custom built CNN using SGD parameter



References

Matplotlib: Python plotting Matplotlib 3.1.1 documentation (n.d.).
URL: https://matplotlib.org/

Tschandl, P., Rosendahl, C. and Kittler, H., 2018. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific data, 5, p.180161.
URL: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T

NumPy NumPy (n.d.).
URL: https://www.numpy.org/

Python Data Analysis Library pandas: Python Data Analysis Library (n.d.).
URL: https://pandas.pydata.org/

scikit-learn: machine learning in Python scikit-learn 0.21.3 documentation (n.d.).
URL: https://scikit-learn.org/stable/

SciPy.org SciPy.org (n.d.).
URL: https://www.scipy.org/