Ecole supérieure en Informitique

Sidi Bel Abbes

Option : ISI

Transfer-Learning and CNN for image classification

Authors
Khebchi Abdallah
Bendaoud Mohammed Amine

November 17, 2021

Abstract

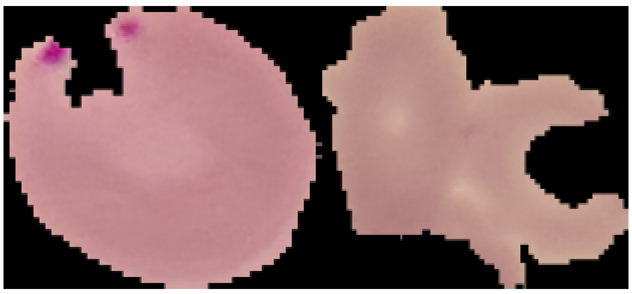
Malaria is a deadly disease which claims the lives of hundreds of thousands of people every year. Computational methods have been proven to be useful in the medical industry by providing effective means of classification of diagnostic imaging and disease identification. This paper examines different deep learning methods in the context of classifying the presence of malaria in cell images. Numerous deep learning methods can be applied to the same problem; the question is which deep learning method is better-suited to a problem relies heavily on the problem itself and the implementation of a model. In particular, convolutional neural networks and VGG 19 are both analyzed and contrasted in regards to their application to classifying the presence of malaria and each model's empirical performance. Here, we implement two models of classification; a convolutional neural network, and the VGG19 transfer learning algorithm . These two algorithms are compared based on validation accuracy. For our implementation, CNN v1 (59 percent) - VGG19 95 percent - CNN v2 95 percent

Contents

1 Dataset

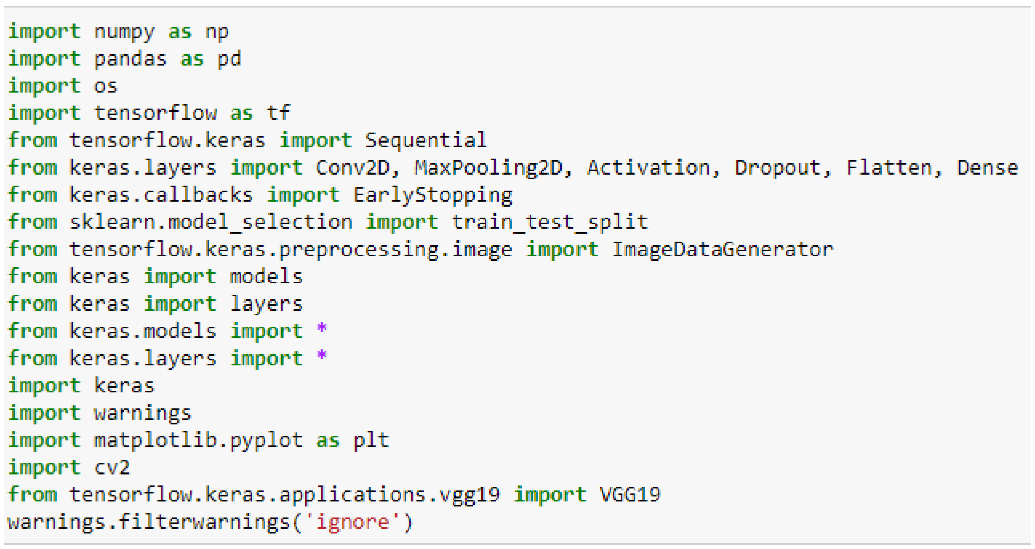
1.1 Malaria cell dataset

This Dataset is taken from the official NIH Website: https://ceb.nlm.nih.gov/repositories/malaria-datasets/
The dataset contains 2 folders :
1- Infected
2- Uninfected
And a total of 27,558 images (133 * 133) .
Example of cell images (infected on the left side)



2 libraries used in the implementation

2.1 Importing the libraries needed



```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tensorflow.keras import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras import models
from keras import layers
from keras.models import *
from keras.layers import *
import keras
import warnings
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.applications.vgg19 import VGG19
warnings.filterwarnings('ignore')
```

Using numpy for matrix manipulation and pandas for data preparation , tensorflow and tensorflow.keras for model building , matplotlib for ploting graphs , cv2 from openCV to handle image data .

2.2 Data preparation

```
    malaria = ["./dataset/Malaria" + '/' +  m for m in malaria]
    normal  = ["./dataset/Normal" + '/' +  n for n  in normal]

    labels = len(malaria)*['malaria'] + len(normal)*['normal']
    data = malaria + normal
    return pd.DataFrame({'Image_Path': data , 'Labels': labels})

df = data_prep(os.listdir('./dataset/Malaria/'), os.listdir('./dataset/Normal/'))
df.head()
```

	Image_Path	Labels
0	./dataset/Malaria/C100P61ThinF_IMG_20150918_14...	malaria
1	./dataset/Malaria/C100P61ThinF_IMG_20150918_14...	malaria
2	./dataset/Malaria/C100P61ThinF_IMG_20150918_14...	malaria
3	./dataset/Malaria/C100P61ThinF_IMG_20150918_14...	malaria
4	./dataset/Malaria/C100P61ThinF_IMG_20150918_14...	malaria

Using pandas we created a data frame for each folder for better manupulation of our dataset.

2.3 Data Augmentation

```
img_datagen = ImageDataGenerator(rotation_range=20,
                                width_shift_range=0.1,
                                height_shift_range=0.1,
                                shear_range=0.1,
                                zoom_range=0.1,
                                horizontal_flip=True,
                                fill_mode='nearest', validation_split=0.2)
```

Using ImageDataGenerator from tf.keras.preprocessing.image we generated more image data with different angels/size duo to the small size of our dataset and also to make the models performance better .

3 CNN and VGG19 Implementations

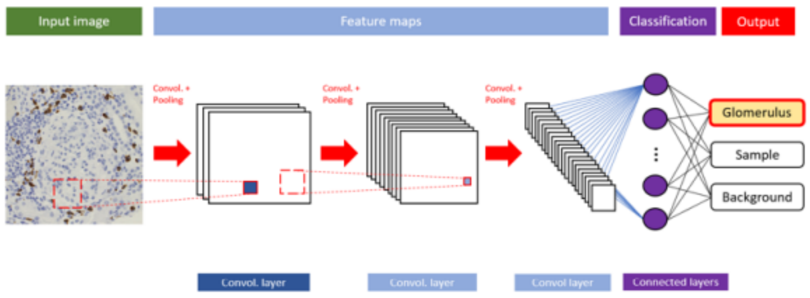
3.1 CNN Implementation

3.1.1 Definition

A convolutional neural network, or CNN, is a deep learning neural network designed for processing structured arrays of data such as images. Convolutional neural networks are widely used in computer vision and have become the state of the art for many visual applications such as image classification

3.1.2 CNN Design

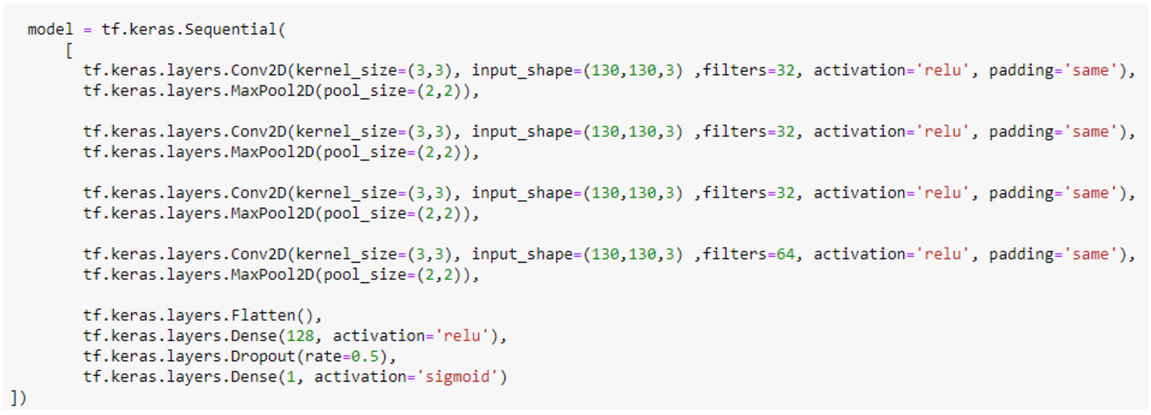
The architecture of a ConvNet is analogous to that of the connectivity pattern of neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.



A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better

3.1.3 Basic CNN Model

Code

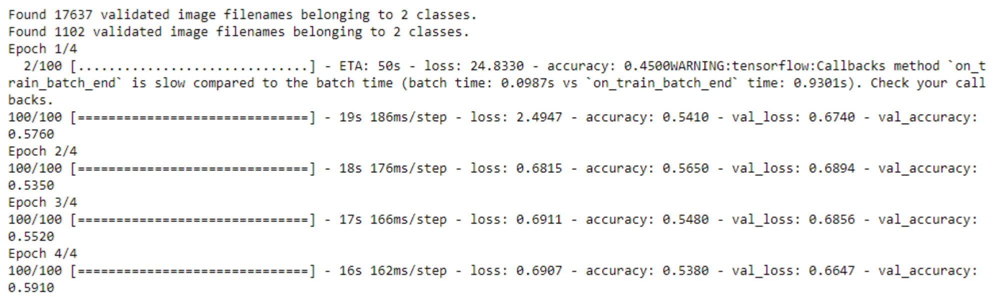


```
model = tf.keras.Sequential(
    [
      tf.keras.layers.Conv2D(kernel_size=(3,3), input_shape=(130,130,3) ,filters=32, activation='relu', padding='same'),
      tf.keras.layers.MaxPool2D(pool_size=(2,2)),

      tf.keras.layers.Conv2D(kernel_size=(3,3), input_shape=(130,130,3) ,filters=32, activation='relu', padding='same'),
      tf.keras.layers.MaxPool2D(pool_size=(2,2)),

      tf.keras.layers.Conv2D(kernel_size=(3,3), input_shape=(130,130,3) ,filters=32, activation='relu', padding='same'),
      tf.keras.layers.MaxPool2D(pool_size=(2,2)),

      tf.keras.layers.Conv2D(kernel_size=(3,3), input_shape=(130,130,3) ,filters=64, activation='relu', padding='same'),
      tf.keras.layers.MaxPool2D(pool_size=(2,2)),

      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dropout(rate=0.5),
      tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Our model consists of four Conv2D kernels of size 3x3, which 'walk over' the input image each one of them follwed by A subsampling layer, also known as an average pooling layer using MaxPool2D with size 2x2 , in each Conv2D layer we are using the relu activation function with a total of 32 filters except for the last Conv2D layer we used 64 for features extraction .
Flatten Layer :
Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer.
Dense Layer :
We used a dense layer to to classify image based on output from convolutional layers with a relu activation function with output of shape (none,128).
To avoid a model from overfitting we used Droput layer with 0.5 rate .
Finnaly we used another dense layer with an output shapr of (none,1) using a sigmoid activation function.

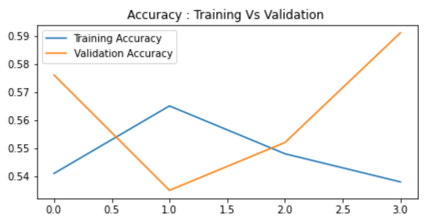
3.1.4 Basic CNN Model Results

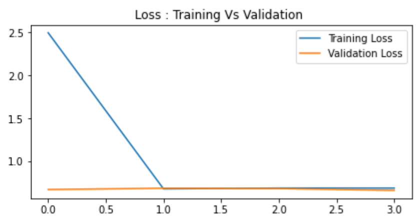
Model evaluation during batches



```
Found 17637 validated image filenames belonging to 2 classes.
Found 1102 validated image filenames belonging to 2 classes.
Epoch 1/4
  2/100 [..............................] - ETA: 50s - loss: 24.8330 - accuracy: 0.4500WARNING:tensorflow:Callbacks method `on_t
rain_batch_end` is slow compared to the batch time (batch time: 0.0987s vs `on_train_batch_end` time: 0.9301s). Check your call
backs.
100/100 [==============================] - 19s 186ms/step - loss: 2.4947 - accuracy: 0.5410 - val_loss: 0.6740 - val_accuracy:
0.5760
Epoch 2/4
100/100 [==============================] - 18s 176ms/step - loss: 0.6815 - accuracy: 0.5650 - val_loss: 0.6894 - val_accuracy:
0.5350
Epoch 3/4
100/100 [==============================] - 17s 166ms/step - loss: 0.6911 - accuracy: 0.5480 - val_loss: 0.6856 - val_accuracy:
0.5520
Epoch 4/4
100/100 [==============================] - 16s 162ms/step - loss: 0.6907 - accuracy: 0.5380 - val_loss: 0.6647 - val_accuracy:
0.5910
```

Ploting the accuracy and loss of the model



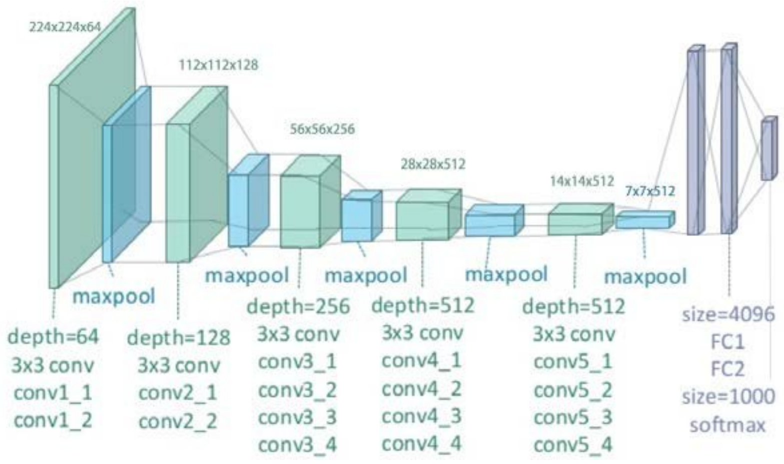


3.2 VGG19 Implementation

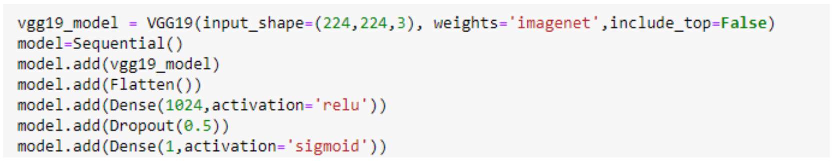
3.2.1 Definition

VGG-19 is a convolutional neural network that is 19 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database . The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

3.2.2 VGG19 Design



3.2.3 VGG19 Model

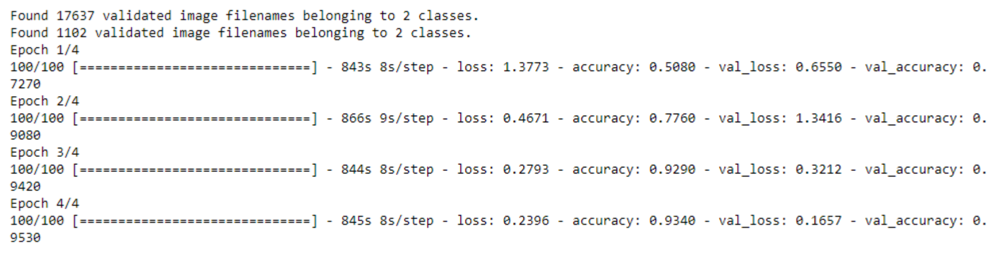


```
vgg19_model = VGG19(input_shape=(224,224,3), weights='imagenet',include_top=False)
model=Sequential()
model.add(vgg19_model)
model.add(Flatten())
model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))
```

In the VGG19 implementation we used a vgg19 model with the default input shape (224,244,3) and we used the imagnet database for weights ,setting the include top to False allow us to mount our dataset .
Along with the vgg19 model we added 4 Layers :
The flatten layer to form a vector of input .
The Dense Layer with the relu activation function for classification and an output shape (none,1024).
To avoid a model from overfitting we used Droput layer with 0.5 rate .
Finnaly we used another dense layer with an output shapr of (none,1) using a sigmoid activation function.

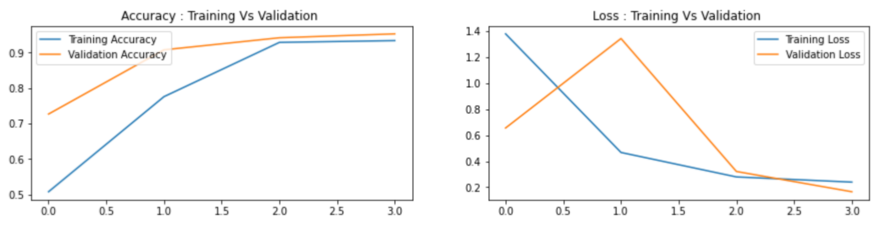
3.2.4 VGG19 Results

Vgg19 model evaluation :



```
Found 17637 validated image filenames belonging to 2 classes.
Found 1102 validated image filenames belonging to 2 classes.
Epoch 1/4
100/100 [==============================] - 843s 8s/step - loss: 1.3773 - accuracy: 0.5080 - val_loss: 0.6550 - val_accuracy: 0.
7270
Epoch 2/4
100/100 [==============================] - 866s 9s/step - loss: 0.4671 - accuracy: 0.7760 - val_loss: 1.3416 - val_accuracy: 0.
9080
Epoch 3/4
100/100 [==============================] - 844s 8s/step - loss: 0.2793 - accuracy: 0.9290 - val_loss: 0.3212 - val_accuracy: 0.
9420
Epoch 4/4
100/100 [==============================] - 845s 8s/step - loss: 0.2396 - accuracy: 0.9340 - val_loss: 0.1657 - val_accuracy: 0.
9530
```

Ploting the accuracy and loss of the vgg19 model :

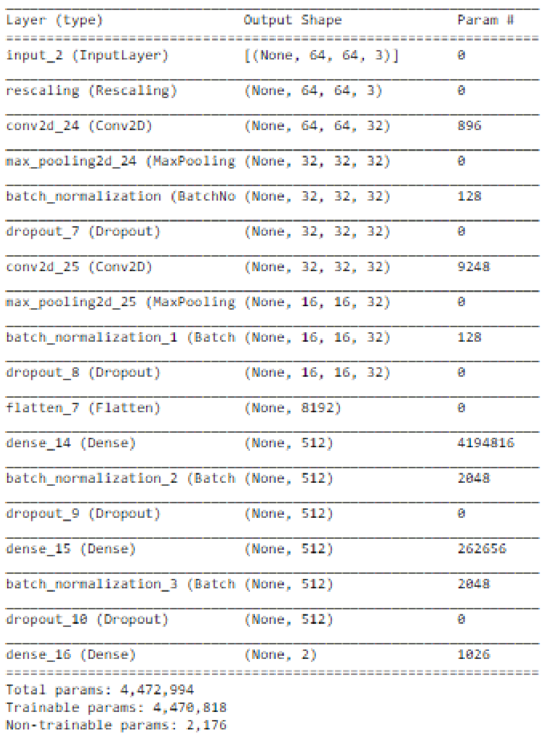


With the vgg19 model we got a higher accuracy score but with a lot of computation resources .
In our case we used a gpu card with 6gb and 16 gb ram the total amount of time taken by the vgg19 us 50 min in the other side CNN model took 1 minute and 10 seconds .
In the next section we will try to get a better accuracy score by enhancing the cnn model .

3.3 Enhancing the CNN model

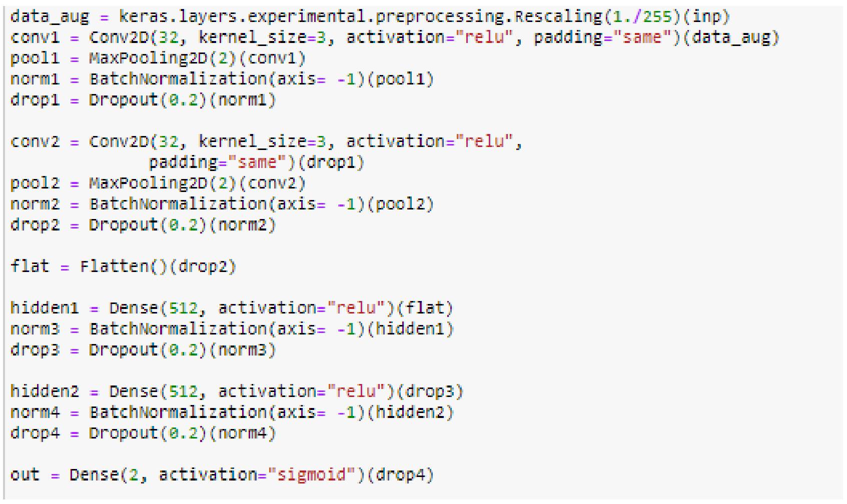
In this section we will try to enhance our cnn model by adding layers and tuning the parameter's.

3.3.1 Proposed new CNN Model



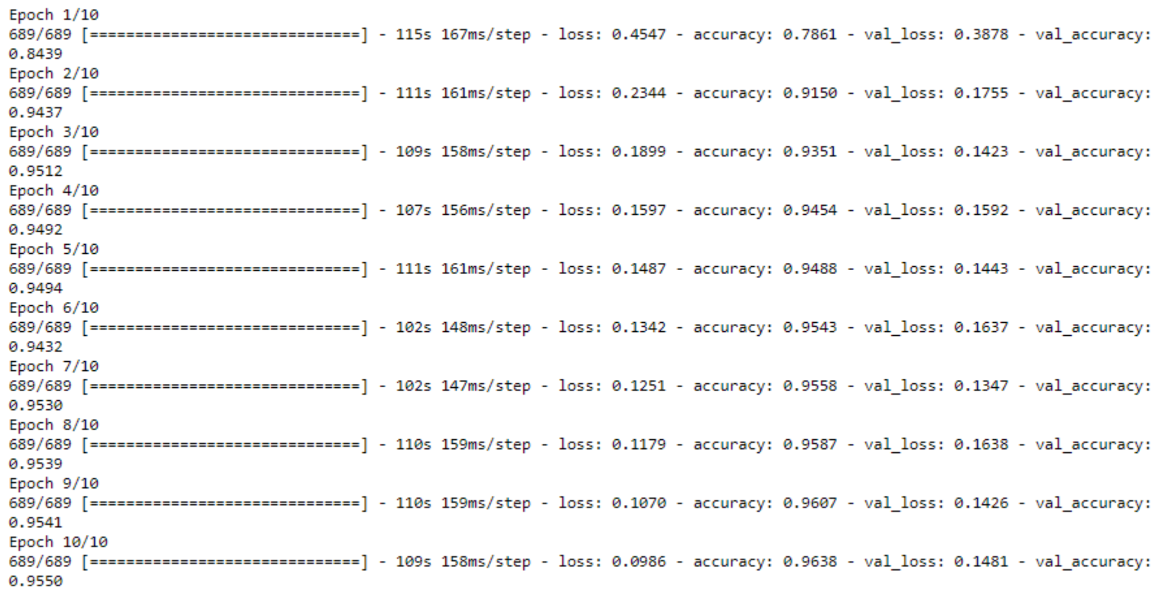
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
rescaling (Rescaling)        (None, 64, 64, 3)         0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               4194816
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 512)               262656
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_10 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 2)                 1026
=================================================================
Total params: 4,472,994
Trainable params: 4,470,818
Non-trainable params: 2,176
```

3.3.2 Code



```
data_aug = keras.layers.experimental.preprocessing.Rescaling(1./255)(inp)
conv1 = Conv2D(32, kernel_size=3, activation="relu", padding="same")(data_aug)
pool1 = MaxPooling2D(2)(conv1)
norm1 = BatchNormalization(axis= -1)(pool1)
drop1 = Dropout(0.2)(norm1)

conv2 = Conv2D(32, kernel_size=3, activation="relu",
              padding="same")(drop1)
pool2 = MaxPooling2D(2)(conv2)
norm2 = BatchNormalization(axis= -1)(pool2)
drop2 = Dropout(0.2)(norm2)

flat = Flatten()(drop2)

hidden1 = Dense(512, activation="relu")(flat)
norm3 = BatchNormalization(axis= -1)(hidden1)
drop3 = Dropout(0.2)(norm3)

hidden2 = Dense(512, activation="relu")(drop3)
norm4 = BatchNormalization(axis= -1)(hidden2)
drop4 = Dropout(0.2)(norm4)

out = Dense(2, activation="sigmoid")(drop4)
```

For the second implementation of CNN model we used the same layers as before additionally we added drop layer and batchNormalisation layer after each Conv2D layer for avoiding the overfitting of the model .
We also changed epochs number to 10 instead of 4 .



```
Epoch 1/10
689/689 [==============================] - 115s 167ms/step - loss: 0.4547 - accuracy: 0.7861 - val_loss: 0.3878 - val_accuracy:
0.8439
Epoch 2/10
689/689 [==============================] - 111s 161ms/step - loss: 0.2344 - accuracy: 0.9150 - val_loss: 0.1755 - val_accuracy:
0.9437
Epoch 3/10
689/689 [==============================] - 109s 158ms/step - loss: 0.1899 - accuracy: 0.9351 - val_loss: 0.1423 - val_accuracy:
0.9512
Epoch 4/10
689/689 [==============================] - 107s 156ms/step - loss: 0.1597 - accuracy: 0.9454 - val_loss: 0.1592 - val_accuracy:
0.9492
Epoch 5/10
689/689 [==============================] - 111s 161ms/step - loss: 0.1487 - accuracy: 0.9488 - val_loss: 0.1443 - val_accuracy:
0.9494
Epoch 6/10
689/689 [==============================] - 102s 148ms/step - loss: 0.1342 - accuracy: 0.9543 - val_loss: 0.1637 - val_accuracy:
0.9432
Epoch 7/10
689/689 [==============================] - 102s 147ms/step - loss: 0.1251 - accuracy: 0.9558 - val_loss: 0.1347 - val_accuracy:
0.9530
Epoch 8/10
689/689 [==============================] - 110s 159ms/step - loss: 0.1179 - accuracy: 0.9587 - val_loss: 0.1638 - val_accuracy:
0.9539
Epoch 9/10
689/689 [==============================] - 110s 159ms/step - loss: 0.1070 - accuracy: 0.9607 - val_loss: 0.1426 - val_accuracy:
0.9541
Epoch 10/10
689/689 [==============================] - 109s 158ms/step - loss: 0.0986 - accuracy: 0.9638 - val_loss: 0.1481 - val_accuracy:
0.9550
```

The results of the second implementation of CNN model are better than our basic CNN model with Accuracy score of 0.955 .
The model took about 16 minutes

3.4 Conclusion

Finally this case study showed up how can model building and parameter tuning can change the accuracy score and the computation resources .
This is a table that summaries our 3 implementations :

results table			
Model	Accuracy score	Time(min)	Difficulty
Basic CNN	0.5910	1.10	Easy
VGG19	0.9530	59	Easy
Enhanced CNN	0.9550	16	Hard

We can say that using vgg19 is a great choice and it is easy to implements only if you have a good computation resources (GPU card , CPU power , Ram)
Otherwise using CNN model can be beneficial in matter of time and without the need of important computation resources but demands a lot of work in implementation .