Homework 5

2015129053 김형철

1.

(a) (1) The function is used for augmenting the images. It is quite useful because you can use it to augment the image in real-time. (2) The function has various arguments. It has "samplewise_center". This adjusts the sample mean to be zero. It is a Boolean factor. It also has "featurewise_std_normalization". This is also in boolean factor. It literally divides the input value with the standard deviation of the data. There is also "rescale". This is basically used to rescale the data. The data is multiplied by the input given to the factor. (If it is given "None" or 0, rescaling does not happen.) Another argument is "vertical_flip". It flips the input data in a vertical direction. The flip is done randomly. Lastly, "preprocessing_function" is an argument that is used to specify the function that will be used for the input. The function will take the image input and return some Numpy tensor that contains the same size as the input.

(b) [3]: [3] is divided into two slot. First slot is used to describe the image data that we will use. That is, we are using plt function to plot the uninfected and infected cell tissues. The second slot of [3] is the part where we rescale and divide the data set. Our usual image data has RGB from 0 to 255. These values are too big for the CNN model to process. So we rescale the value to go between 0 and 1 by multiplying it by 1/255. Also, by using validation_split argument, we split the data into 80-20 sets. This is done so that we can use 80% of the data set as training data and 20% as test data. The width and height is also written so that the input shape of the data is specified.

[4]: This part is used to load the data for training. The code is used to pick up and load the data set that will be used for training. Since we specified that 80% of the data set will be allocated to the training set, this gives us 22048 images with 2 classes (infected or not).

[5]: Similar to [4], this part is used to load the data for testing. The code is used to pick up and load the data set that will be used for testing. Since we specified that 20% of the data set will be allocated to testing set, this gives us 5510 images with 2 classes (infected or not).

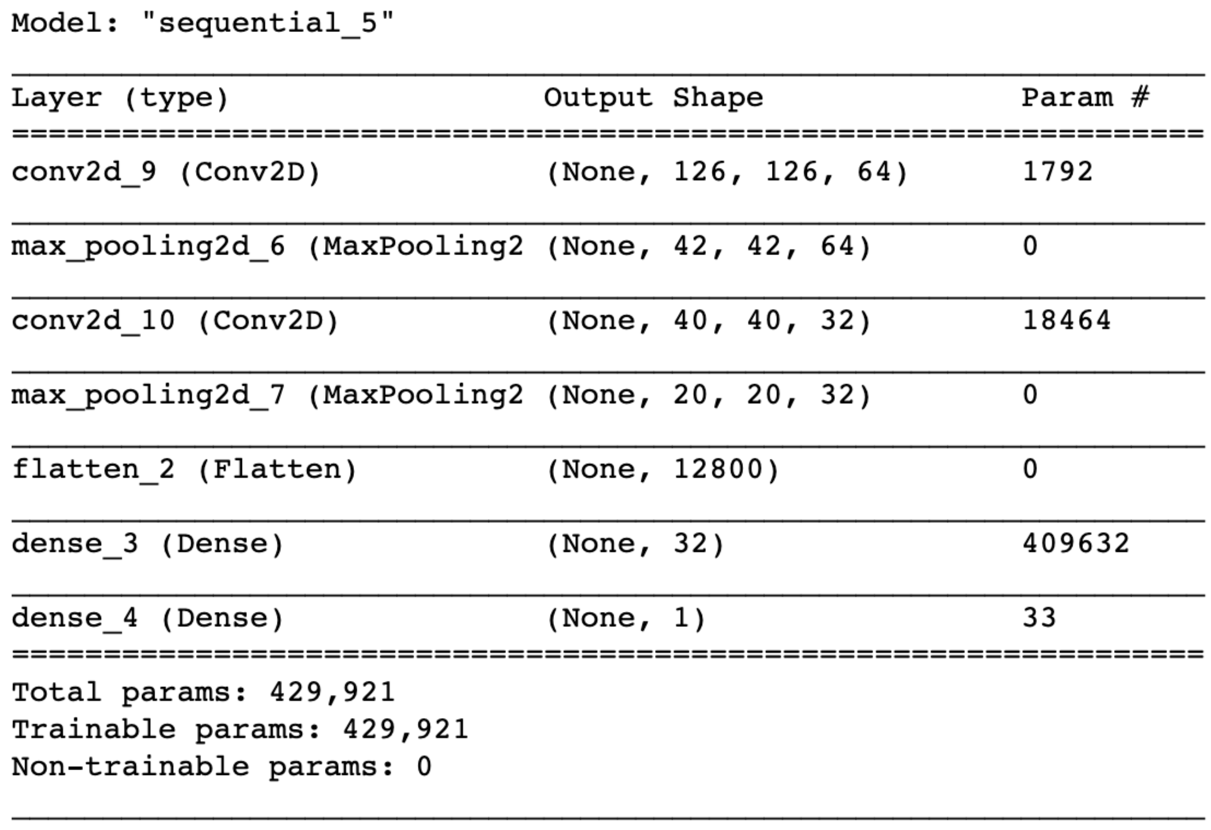
(c) I built CNN with the following codes:

```
# define the keras model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(128, 128, 3)))
model.add(tf.keras.layers.MaxPooling2D((3,3)))
model.add(tf.keras.layers.Conv2D(32, (3,3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2,2)))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Using model.summary function, I show the structure of the CNN:

```
model.summary()
```

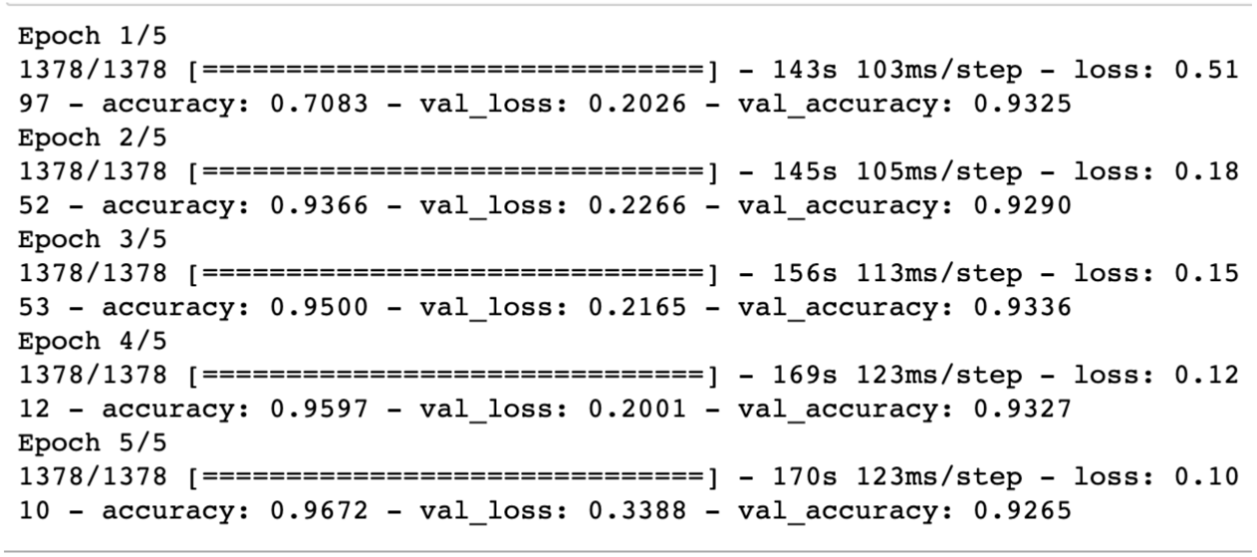


```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 126, 126, 64)      1792
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 42, 42, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 40, 40, 32)        18464
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 20, 20, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                409632
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 429,921
Trainable params: 429,921
Non-trainable params: 0
_________________________________________________________________
```

(d) Now I compiled the model and fitted the model using the codes model.compile and model.fit:

```
# model fit
model.compile(optimizer='adam',
          loss='binary_crossentropy',
          metrics=['accuracy'])

history=model.fit(trainDatagen, epochs = 5,
            validation_data=valDatagen)
```

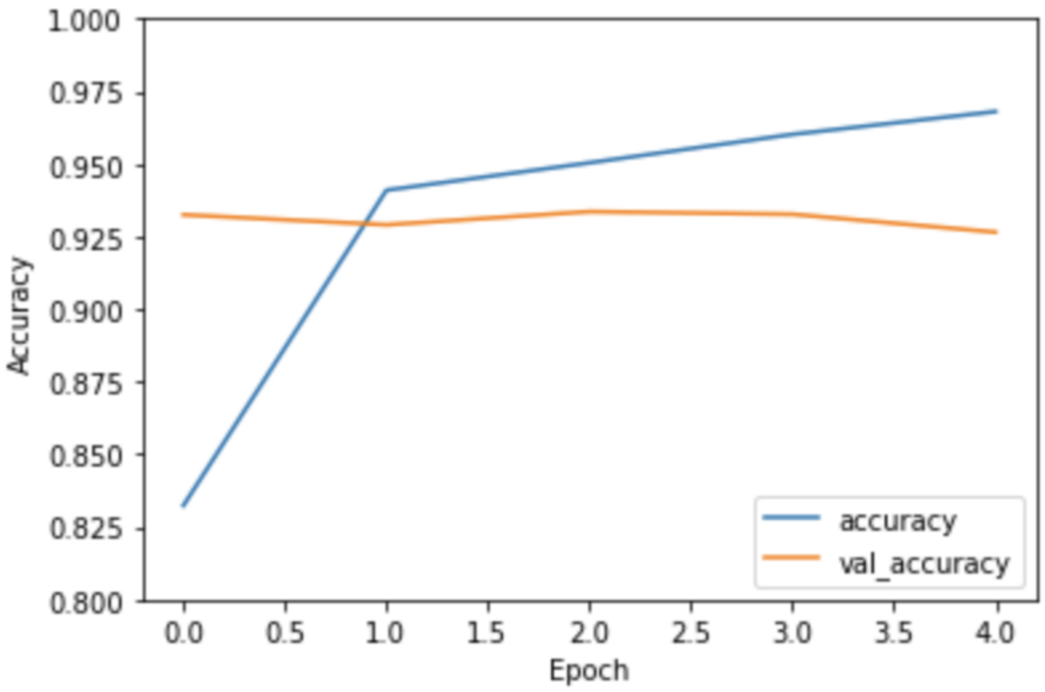


```
Epoch 1/5
1378/1378 [==============================] - 143s 103ms/step - loss: 0.51
97 - accuracy: 0.7083 - val_loss: 0.2026 - val_accuracy: 0.9325
Epoch 2/5
1378/1378 [==============================] - 145s 105ms/step - loss: 0.18
52 - accuracy: 0.9366 - val_loss: 0.2266 - val_accuracy: 0.9290
Epoch 3/5
1378/1378 [==============================] - 156s 113ms/step - loss: 0.15
53 - accuracy: 0.9500 - val_loss: 0.2165 - val_accuracy: 0.9336
Epoch 4/5
1378/1378 [==============================] - 169s 123ms/step - loss: 0.12
12 - accuracy: 0.9597 - val_loss: 0.2001 - val_accuracy: 0.9327
Epoch 5/5
1378/1378 [==============================] - 170s 123ms/step - loss: 0.10
10 - accuracy: 0.9672 - val_loss: 0.3388 - val_accuracy: 0.9265
```

It shows that accuracy of the training set is around 97% and the accuracy of the validation set is around 93%.

I also plotted the accuracy:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.8,1])
plt.legend(loc='lower right')
```



The accuracy of both training and data sets are well above 80%.