CS551G_Assessment1_ZHIXI_TANG_52097136

STUDENT NAME: ZHIXI TANG

STUDENT ID: 52097136

Index

Abstract

This file is the report of CS551G Assessment 1 experiment. This experiment includes two tasks. Task 1 is that generating 50 synthetic images of COVID-19 based on the provided COVID-19 X-ray images and cgan python file. Task 2 is that using the pre-trained model to classify COVID-19 and normal CXRs. All the experiments are operated on Google Colab.

The original images are saved in Google Drive, its directory in Colab is '/content/Mydrive/cgan_Assessment/Covid-19(Normal)', the 50 synthetic images are also saved in this directory. If the instructor gets the error when running my ipynb file, please kindly check if the directory path is correct.

All procedures of the experiment were runing on Google Colab. I only shown the code snippets I think it is essential and important to interpret my solution, for complete source code, please kindly check my .ipynb file.

Task 1:

Subtask 1.1:

Step 1: Preprocess Data

The sizes of provided images of Covid-19 are not equal. In this case, we have to re-size the images before we feed the image to the network. We will add a new function in the cgan.py file to preprocess the data. Each image will be reshaped to $(150, 150, 1)$, at the same time, we will also normalize the data of images (rescale the range of image array from $(-1, 1)$). When we load the training data and labels for the network, we will call this function. The code snippet for this function is shown as below:

```
def preprocess_data(self):
  img_dic = os.path.join(gdrive_path, 'MyDrive', 'cgan_Assessment', 'Covid-19')
  imgList = []
  imgList.extend(glob.glob(os.path.join(img_dic, '*')))

  y_train = np.array([0] * len(imgList))
  x_train  = np.zeros(shape=[len(imgList), 150, 150, 1])

  for i in range(len(imgList)):
    img = load_img(imgList[i], color_mode='grayscale', target_size=(150, 150))
    img_array = img_to_array(img)
    img_array = img_array.astype(np.float64)
    x_train[i] = img_array

  # configure input
  x_train = (x_train.astype(np.float32) - 127.5) / 127.5
  y_train = y_train.reshape(-1, 1)

  return x_train, y_train
```

Step 2: Re-write original cgan.py -- sample_images function

In the original CGAN file, this structure is for the training of MNIST dataset. To generate the required images, we will re-write the sample_images function as the following:

```
def sample_images(self, epoch, epochs, img_dir):
  if epoch == epochs-1:
  # save 50 images
    final_img_dir = os.path.join(gdrive_path, 'MyDrive', 'cgan_Assessment', 'Generated_covid_imgs')
    if not os.path.exists(final_img_dir):
      os.mkdir(final_img_dir)

    noise = np.random.normal(0, 1, (50, self.latent_dim))
    gen_labels = np.zeros(50).reshape(-1, 1)
    gen_images = self.generator.predict([noise, gen_labels])

    img_nums = len(gen_images)

    for i in range(img_nums):
      imgFileName = os.path.join(final_img_dir, '_' + str(i) + '.png')
      save_image = (gen_images[i])
      save_img(imgFileName, save_image)
    else:
      r, c = 2, 5
      noise = np.random.normal(0, 1, (10, self.latent_dim))
      gen_labels = np.zeros(10).reshape(-1, 1)
      gen_imgs = self.generator.predict([noise, gen_labels])

      gen_imgs = 0.5 * gen_imgs + 0.5
      fig, axs = plt.subplots(r, c)
      cnt = 0
      for i in range(r):
        for j in range(c):
          axs[i][j].imshow(gen_imgs[cnt,:,:,0], cmap='gray')
          axs[i][j].set_title("Digit: %d" % gen_labels[cnt])
          axs[i][j].axis('off')
          cnt+= 1
      fig.savefig("images/%d.png" % epoch)
      plt.close()
```

In the last epoch, the function will generate 50 synthetic images saving them into Google drive, the directory is "generated_covid_imges". If the epoch is not the last one, then the function will generate 10 images into one plot to let the user see the quality of the generated images in the current epoch.

Step 3: Re-write original cgan.py -- other hyperparameters

To let the cgan.py adopt our task, we also have change some parameters. Firstly, we have to change the input shape of CGAN class as the following:

```
class CGAN():
  # re-wrotten part
  def __init__(self):
    self.img_rows = 150 # height of image is 150
    self.img_cols = 150 # width of image is 150
    self.channels = 1   # channel of image is 1
    self.img_shape = (self.img_rows, self.img_cols, self.channels)
    self.num_classes = 2   # real Covid and synthetic Covid image, only 2 classes
    self.latent_dim = 256  # noise dimensionality

    optimizer = Adam(0.0002, 0.5) # optimizer for discriminator
    optimizer_g = Adam(0.002, 0.5) # optimizer for generator

# Other parts weren't re-wrotten so we omit it ....
```

In the train function, we will add some codes to create a directory to save the image.

```
def train(self, epochs, batch_size=128, sample_interval=50):
  # create a directory to save the images
  img_dir = r'/content/images'
  if not os.path.exists(img_dir):
    os.makedirs(img_dir)
  else:
    shutil.rmtree(img_dir)
    os.makedirs(img_dir)
  # Adversarial ground truths
  # .....
```

Step 4: Mount the Google Drive, upload cgan.py

Because this experiment was operating on Google Colab and we will use the data already saved in Google Drive, so we have to write a code script to mount Google Drive, and import the cgan.py we re-wroten on the above.

```
# mount Google Drive
from google.colab import drive, files
gdrive_path = '/content/gdrive'
drive.mount(gdrive_path)
files.upload()
```

```
# import re-written CGAN class
from re_written_cgan import CGAN
```

Step 5: Create a CGAN object and start training

On the Google Colab, we will create a CGAN object called cgan, then call its internal function train to start the training process. We will set the epochs = 20000, batch_size = 64, and every 1000 epochs we will draw 10 images into one plot to see the quality of generated images.

```
cgan = CGAN()
cgan.train(epochs=20000, batch_size=64, sample_interval=1000)
```

After executing the above command the network start training. We can see in the 0th epoch, 1500th epoch, 19000 epoch, the generated images are below:

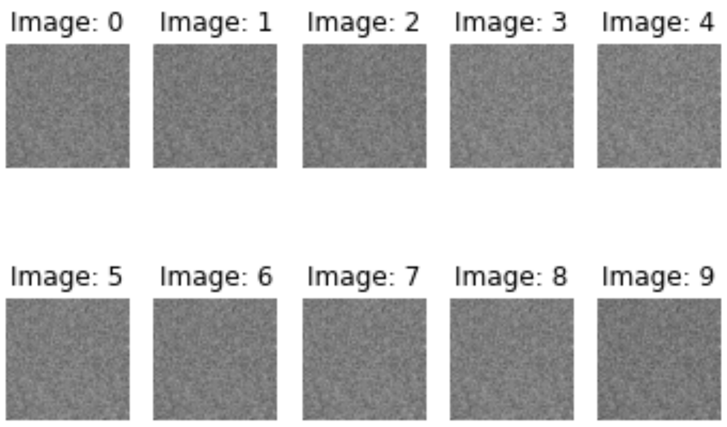


Image: 0
Image: 1
Image: 2
Image: 3
Image: 4
Image: 5
Image: 6
Image: 7
Image: 8
Image: 9

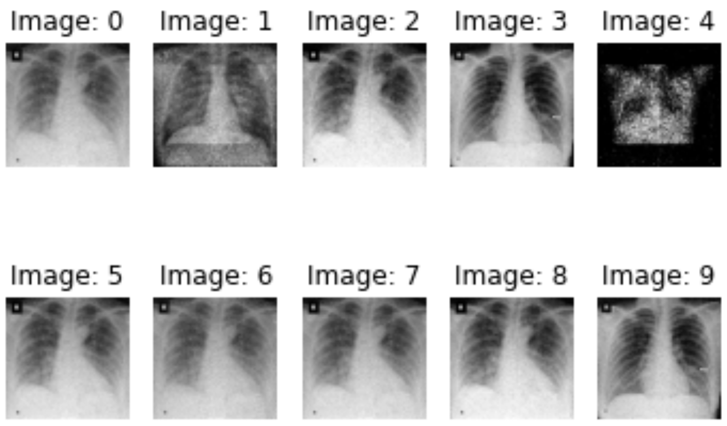


Image: 0
Image: 1
Image: 2
Image: 3
Image: 4
Image: 5
Image: 6
Image: 7
Image: 8
Image: 9

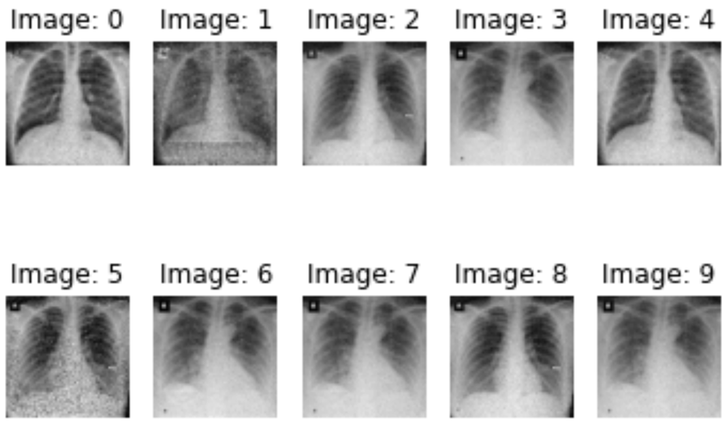
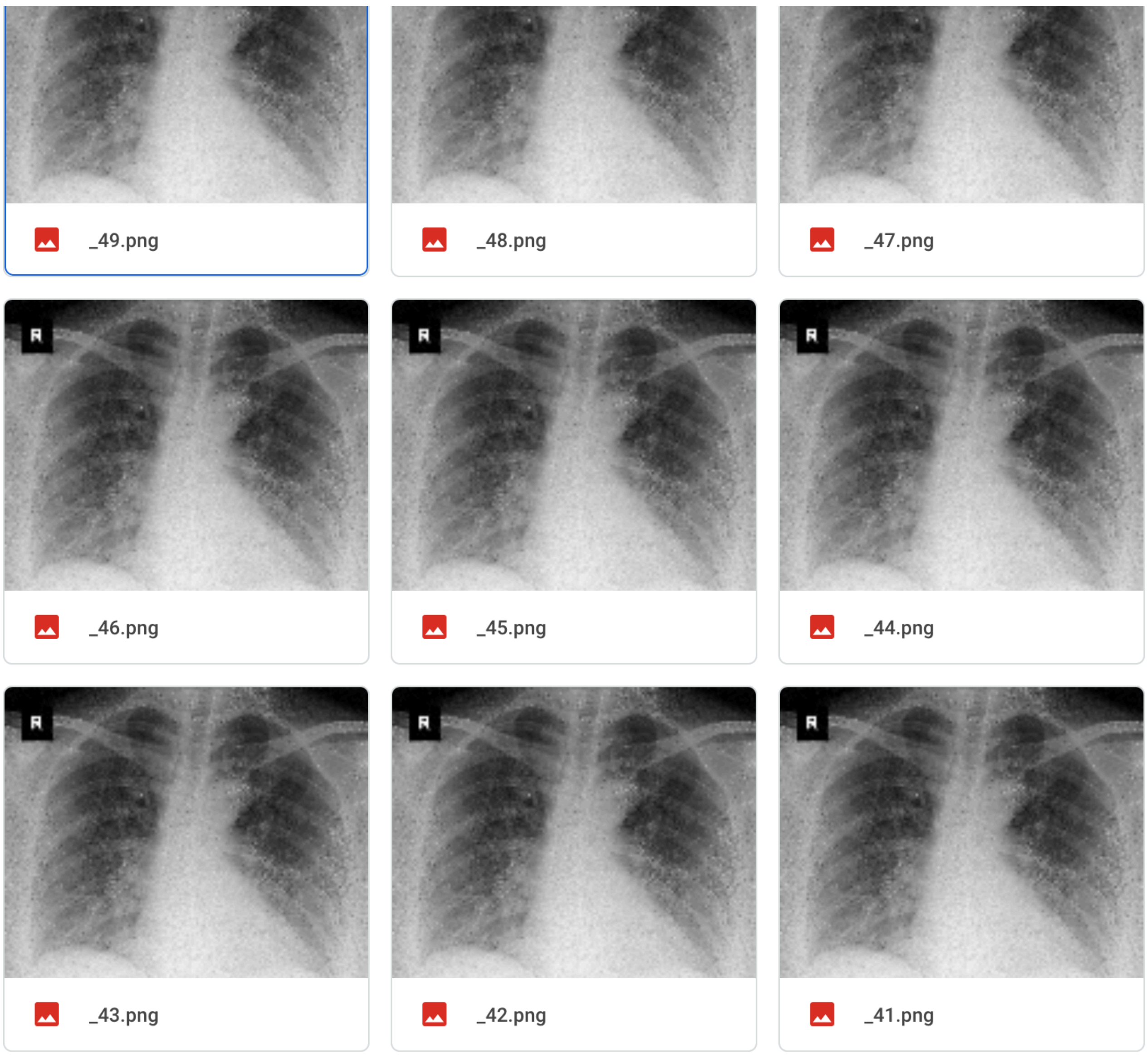


Image: 0
Image: 1
Image: 2
Image: 3
Image: 4
Image: 5
Image: 6
Image: 7
Image: 8
Image: 9

After the process of training, we check the Google drive, we will found a directory called "Generated_covid_imgs" containing 50 synthetic images of Covid-19 X-rays, part of images are shown as below:



Subtask 1.2:

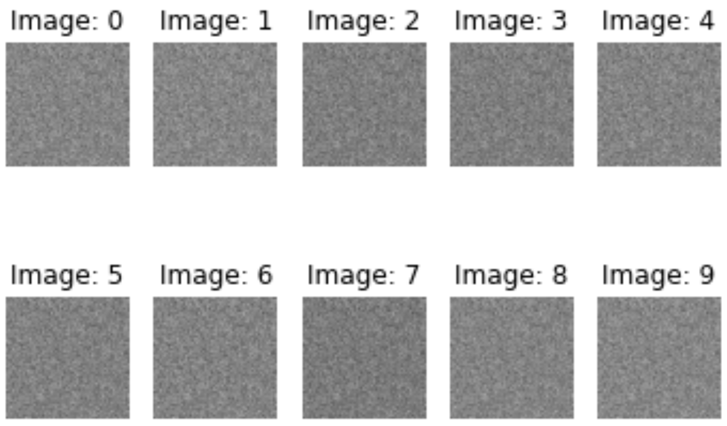
As we have done all the preparations in subtask 1.1, so in this task, we will change the hyperparameters directly to see the training result. Based on the hyperparameters in subtask 1.1, we will do three groups of change as below *(please check the details of revision in the files of "change-1.py", "change-2.py", "change-3.py".)*:

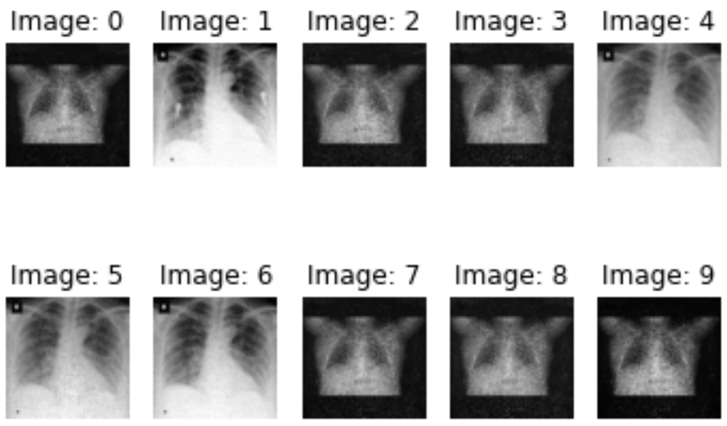
- revise the learning rate of the generator from 0.002 to 0.0002, epochs from 20000 to 10000. In this change, we will plot 10 images in one plot after every 2000 epochs and draw 50 individual images in the last epoch. These images will be saved into '/content/images_change1', and '/content/gdrive/MyDrive/cgan_Assessment/Generated_covid_imgs_change1'.
- revise epochs from 20000 to 15000, batch size from 64 to 32, change the neurals of each layer of both generator and discriminator, we will plot 10 images in one plot after every 3000 epochs and draw 50 individual images in the last epoch. These images will be saved into

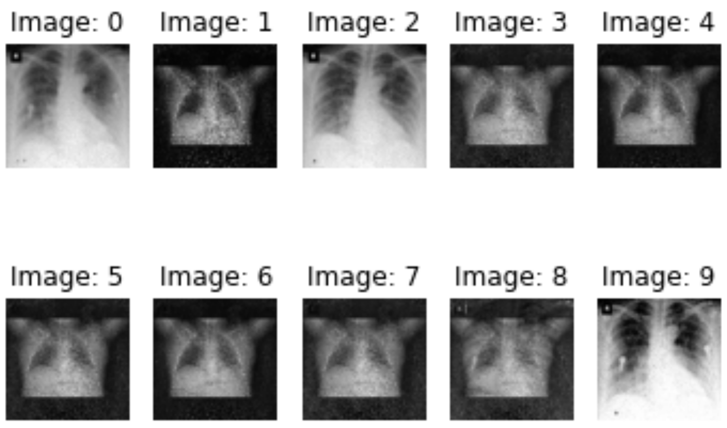
'/content/images_change2', and
'/content/gdrive/MyDrive/cgan_Assessment/Generated_covid_imgs_change2'.

- Add a new hidden dense layer(512 neurals) for the generator, a new hidden dense layer(512 neurals) for the discriminator. we will plot 10 images in one plot after every 2000 epochs and draw 50 individual images in the last epoch. These images will be saved into '/content/images_change3', and '/content/gdrive/MyDrive/cgan_Assessment/Generated_covid_imgs_change3'.

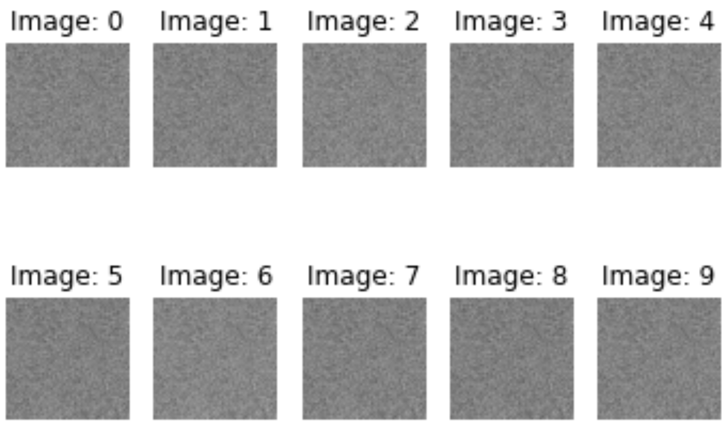
In the first CGAN class, the images generated in the 0th epoch, 5000th epoch, 9000th epoch are shown as below:

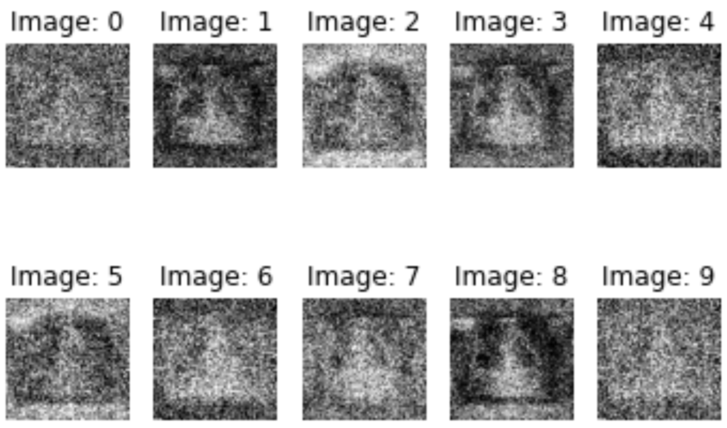


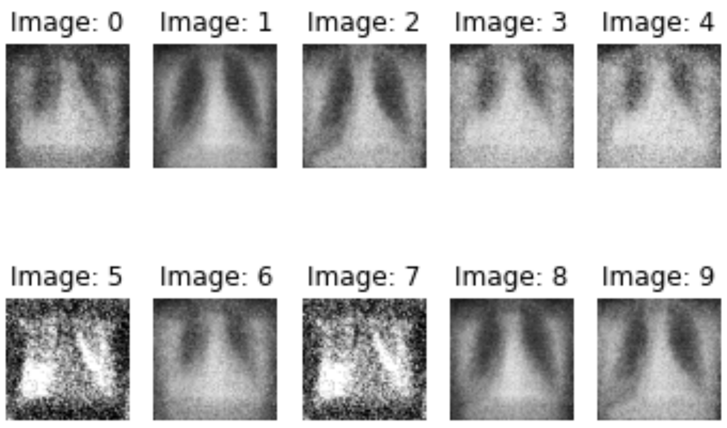




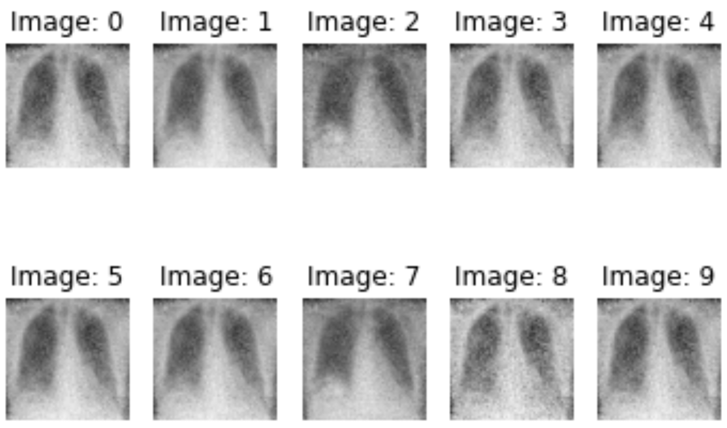
In the second CGAN class, the images generated in the 0th epoch, 6000th epoch, 12000th epoch are shown as below:







In the third CGAN class, the images generated in the 0th, 10000th epoch, 19000th epoch are shown as below:



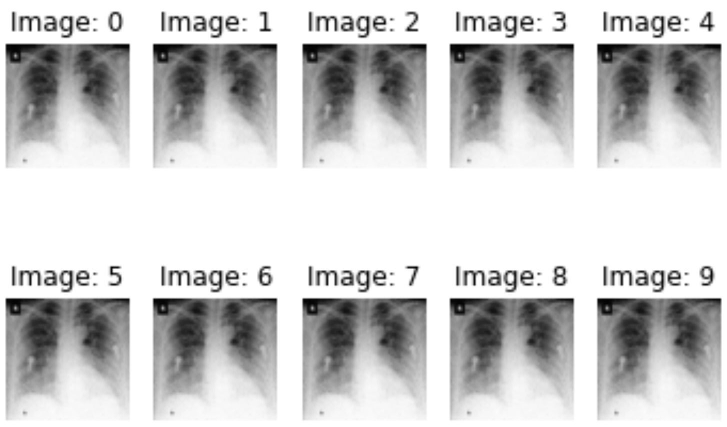


Image: 0
Image: 1
Image: 2
Image: 3
Image: 4
Image: 5
Image: 6
Image: 7
Image: 8
Image: 9

Task 2:

Subtask 2.1:

Step 1: Creating directories

Firstly, we will write a code snippet to create directories to save the test(120 images), validation(40 images), test images (40 images). The architecture of directories are shown below, for the complete code please kindly check the ipynb file.

```
/content/base_dir
├── test
│    ├── covid
│    └── normal
├── train
│    ├── covid
│    └── normal
└── validation
     ├── covid
     └── normal
```

Step 2: Copy images from Google Drive

After we created the directories, we will copy the images from Google Drive, 120 images for training (60 Covid-19 images, 60 normal images) for training, 40 images for validation (20 Covid-19 images, 20 normal images), finally using 40 images for testing as required. (20 Covid-19 images, 20 normal images).

Step 3: Import ResNet50

We import the ResNet50 pre-trained model, and build a base model and set up the related hyperparameters. We set the weights of imagenet, the input shape of this network will be (150, 150, 3), as we are going to add the classfication on our own, so we set "include_top" as False. The code snippet is shown below:

```python
from keras.applications.resnet import ResNet50  # import the model
# create the base model
conv_base = ResNet50(weights='imagenet', include_top=False, input_shape=
(150,150,3))
# show the architecture of base model
conv_base.summary()
```

Step 4: Add Dense Layers

In this step, we will add two Dense layers on the top of the ResNet50 model. the final Dense layer uses Sigmoid as the activation function, as the network is a binary classification, those two Dense layers are shown below:

```
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

then we will freeze the ResNet50 model, so in this case, only the weight and bias of Dense layers will be changed with the process of training. This step is very important, if we don't freeze ResNet50, the weights and biases will also be changed with the training process, then it will be pointless to import the pre-trained model.

```
conv_base.trainable = Flase # freeze ResNet50
```

Step 5: Data Argumentation

In this task, the dataset is very small (160 images for training, 40 images for test) so it may cause overfitting especially we are going to use some images for validation. To avoid this problem, we will use data argumentation via Kreas class ImageDataGenerator, this class is specialized to process deep learning data on the image field. Therefore, we will use imageDataGenerator object to increase the scale of our dataset.

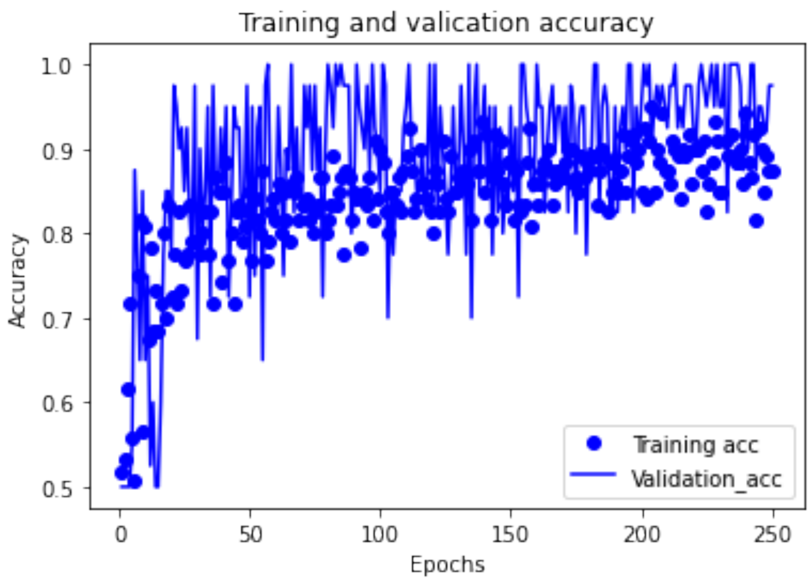
```
# create ImageDataGenerator object for increasing the scale of test dataset
from keras.preprocessing.image import ImageDataGenerator
# create train_datagen
train_datagen = ImageDataGenerator(
  rescale = ./255,  # data normalization
  rotation_range = 45,  # the range of rotation angel
  width_shift_range = 0.2,  # range of horizontal moving
  height_shift_range = 0.2, # range of vertical moving
  shear_range = 0.2,  # Shear Intensity
  zoom_range = 0.2, # Range for random zoom
  horizontal_flip = True, # Randomly flip inputs horizontally
  fill_mode = 'nearest' # Points outside the boundaries of the input are
filled with nearest pixels
)
# create test datagen, but we don't have to increase the scale of test data
test_datagen = ImageDataGenerator(rescale=1./255)
```

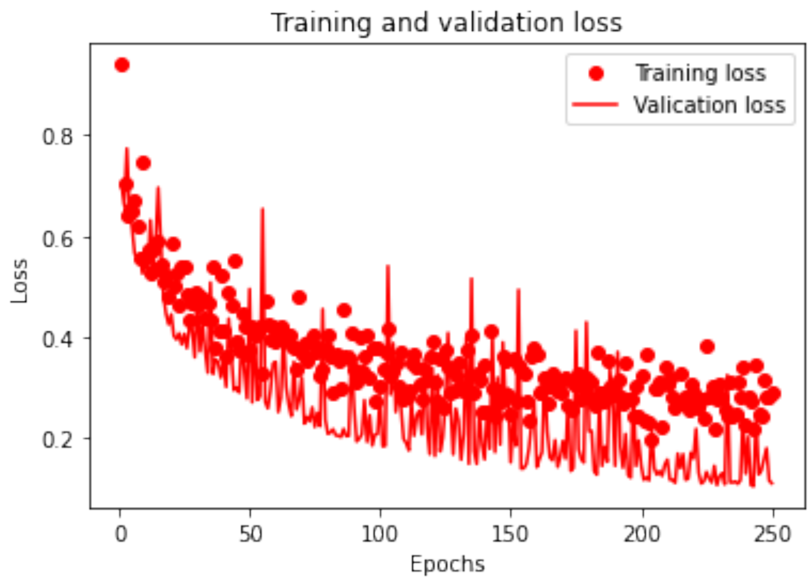
Step 6: Train the model

In this step, we will set up the loss function as "binary_crossentropy", optimizer as "RMSprop", epochs = 250, and other hyperparameters of the network, then use the method `model.fit_generator` to train the network. The method will return a history object, we will use this object to evaluate the network later.

Step 7: Draw the images of the results of the training

After the training, we will draw the image of loss and accuracy, the images are shown as following:

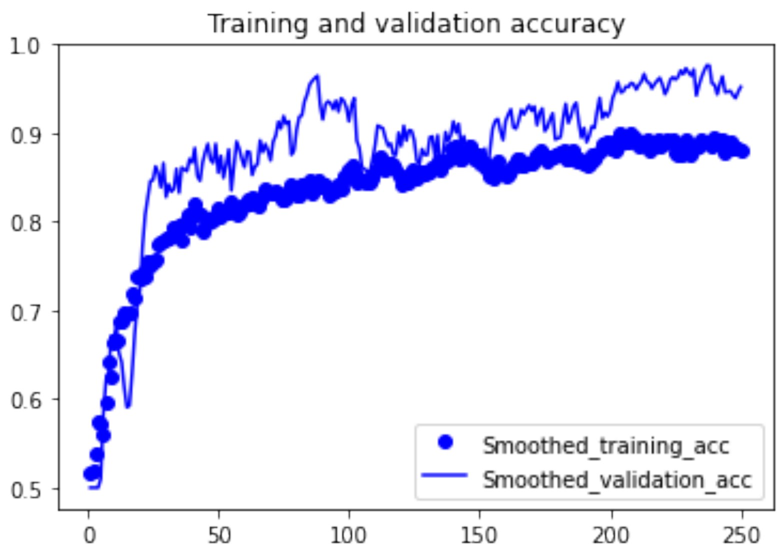


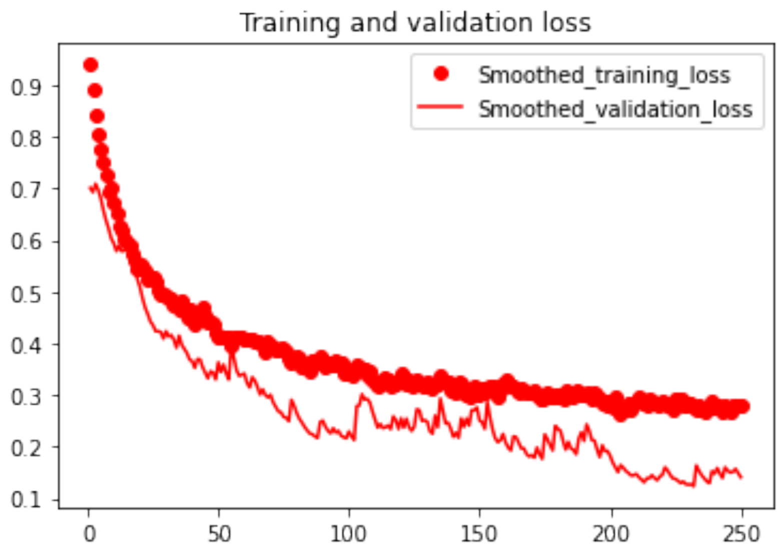


From the above images, we can tell there are some noises in the images. To make the images become more readable and intuitive, we will write a function to smooth the images. The code of the function is shown as below:

```python
def smooth_curve(points, factor=0.8):
    smoothed_points=[]
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point*(1-factor))
        else:
            smoothed_points.append(point)
    return smoothed_points
```

The smoothed images are shown as below:





From the above-smoothed images, we can tell that when the 220th epoch - 235 epoch, the accuracy is around 97%. After this epoch, the accuracy started decreasing, which means the model started becoming overfitting.

Step 8: Train a new model and evaluate it with test dataset.

According to the conclusion we got in the last step, we will train a new model over again with 235 epochs. Then after the training process, we will evaluate the model with the test dataset. We will print the test loss and test accuracy. The results are shown below:

```
Test loss:  0.09341797977685928
Test acc:  1.0
```

Subtask 2.2:

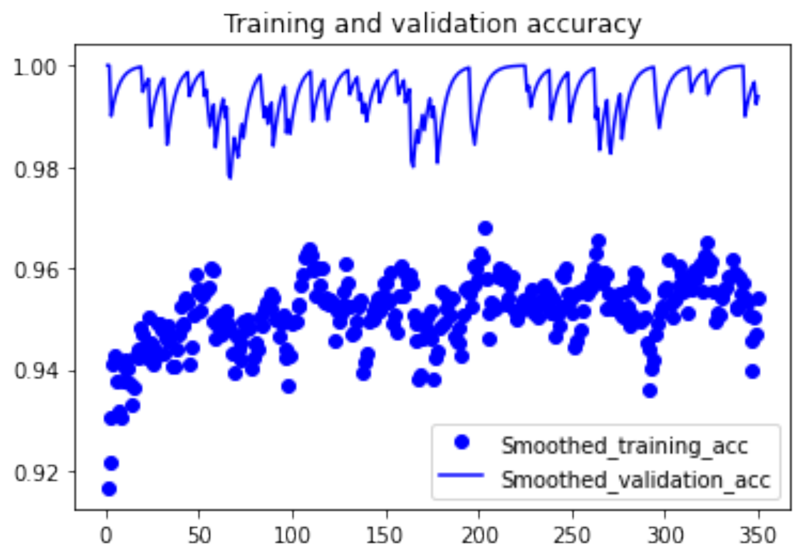
The procedure of this task is almost the same as the last one. The only difference is that we have to use 50 synthetic images generated from Task 1 in the training.

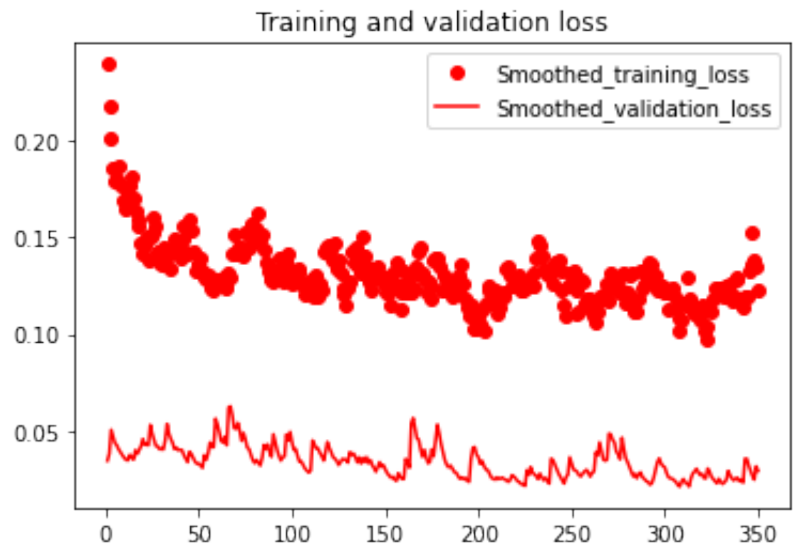
Step 1: Load 50 synthetic images

Based on the directory architecture in Task 2.1, we will amend the script a little bit to load the synthetic images. We will still use 120 images for training (30 synthetic images, 30 original covid-19 images, 60 normal images), 40 images for validation (10 synthetic images, 10 original covid-19 images, 20 normal images), 40 images for testing(10 synthetic images, 10 original images, 20 normal images). The architecture of directories are same as Task 2.1

Step 2: Train the network and evaluate

Similarly, we will train the network with 350 epochs, then draw the smoothed accuracy and loss images for observation. The smoothed images are shown as below:





From the above images, we can tell in the 200 epochs the accuracy at the top, so we will train a new model over again with 200 epochs. Finally, we will evaluate the network with the test dataset and print the results as following:

```
test loss:  0.3409997522830963
test acc:  0.925000011920929
```

Conclusion, Supplementary, or Argumentation

On the above are all the contents of my experiment report, however, I do know my solution wasn't ideal enough. As the following are some defects I think I can improve in the future:

1. In Task 1.1, the quality of synthetic images are not very good. I personally think they have a very big difference from the original pictures. I think if I can build a convolutional neural network the quality could be improved (but I haven't learnt that).
2. In Task 2.2, still, because of the bad quality of synthetic images, the noise of input is very huge, which caused the accuracy curve is very unstable, although the overall trend is decreasing.
3. An idea to improve the quality of synthetic images is to rescale the image to 28×28 or 32×32, One day before the due, I noticed that many image datasets on Kaggle and Keras inside image datasets with this size. So maybe this idea could make the accuracy curve of Task 2.2 be better but it will be too small to check intuitively.

4. Another idea to improve the quality of synthetic images, I noticed that the channel of generated image is 1, the mode is grayscale, however, the channel of original images is 3 or more, the mode is rgb. In this case, I think if I stack 3 or more same images into 1 image. I think this way is not very logical (Because there are definitely other ways to increase the generator to improve the quality of image) but still could be a method to improve the quality of image.