Training a SRGAN and Using Its Images To Train a Classifier

Acknowledgements

I used this github as the base for my SRGAN:
https://github.com/vishal1905/Super-Resolution

Additionally, I used a notebook from my class as the basis of my classifier.

The dataset included in from Kaggle
https://www.kaggle.com/c/dogs-vs-cats/data

Description of Project

For this project, I took high definition images of cats and dogs and used them to train a SRGAN. The images were downsized to 32x32 before going through the SRGAN for 150 epochs and being brought up to 128x128.

I then used the generated images to train a classifier and compared the results to a classifier trained on the original images.

The Dataset

The dataset I used contains 25000 images of dogs and cats. They have been included along with this project and are all contained in the same folder. The csv files for training, validation and testing have also been included. The distribution of images is as follows:

Data Set	Dog Pictures	Cat Pictures	Percentage
Training	8,750	8,750	70%
Test	3,750	3,750	30%

Training the SRGAN

First, replace the path listed here with your dataset location. I used google drive to store mine since training the GAN took place over multiple sessions and it took less time to get started. You can however, use any source that you would like.

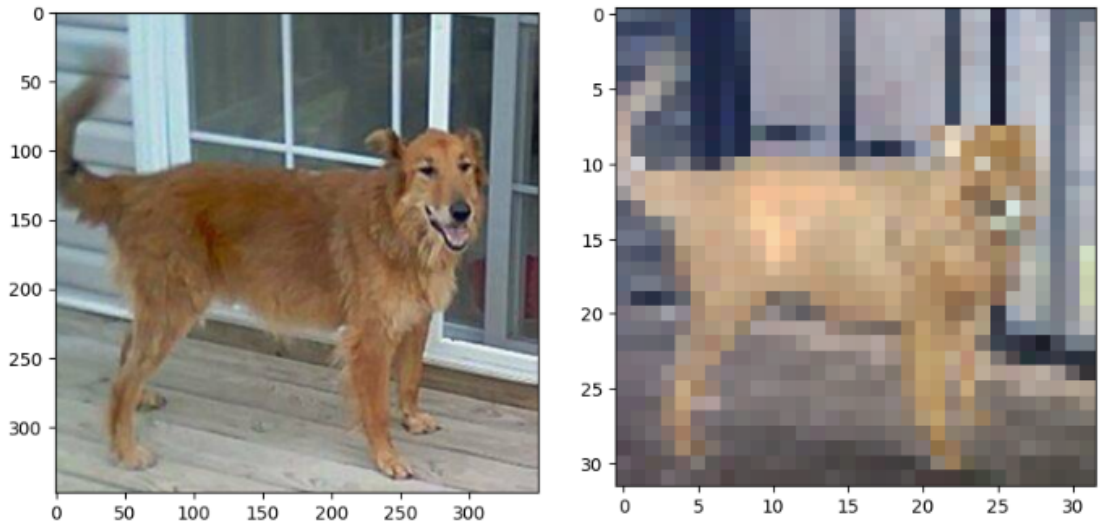
```
#download images from google drive and save locally
!unzip '/content/drive/MyDrive/University/Applied AI/Midterm/Data/train.zip' -d '/content'
```

Next, update the path listed here to wherever your images were output after unzipping.

```
[ ]  catDogData = '/content/train'
     images = os.listdir(catDogData)
     imageList = images[:25000]
```

This next block is used to demonstrate downsizing the high definition images. You only need to change the (32,32) at the end if you are changing it to any other ratio. My output can be seen below. Please note that while the images are displayed at the same size, the size of the x and y axis are different.

```
originalImage = plt.imread(os.path.join(catDogData, imageList[1]))
modifiedImage = cv2.resize(cv2.GaussianBlur(cv2.imread(os.path.join(catDogData, imageList[1])),(5,5),cv2.BORDER_DEFAULT),(32,32))

image1 = Image.fromarray((originalImage).astype('uint8'))
plt.figure(1)
plt.imshow(image1)
plt.figure(2)
image2 = Image.fromarray((modifiedImage * 255).astype('uint8'))
plt.imshow(image2)
```



Run the next few cells in order until you get to the actual training. There are two lines that give summaries of the generator and discriminator. I would recommend running these with you dimensions inputted so that you can verify the output is correct.

```
summary(gen,(3,32,32))
```

The following is my output and shows that the last layer (at the bottom) is 128 x 128 x 3 which is what I want. If your desired dimensions are not shown, you may need to change some of the layers of the SRGAN.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	15,552
PReLU-2	[-1, 64, 32, 32]	1
Conv2d-3	[-1, 64, 32, 32]	36,864
BatchNorm2d-4	[-1, 64, 32, 32]	128
PReLU-5	[-1, 64, 32, 32]	1
Conv2d-6	[-1, 64, 32, 32]	36,864
BatchNorm2d-7	[-1, 64, 32, 32]	128
Conv2d-8	[-1, 64, 32, 32]	36,864
BatchNorm2d-9	[-1, 64, 32, 32]	128
PReLU-10	[-1, 64, 32, 32]	1
Conv2d-11	[-1, 64, 32, 32]	36,864
BatchNorm2d-12	[-1, 64, 32, 32]	128
Conv2d-13	[-1, 64, 32, 32]	36,864
BatchNorm2d-14	[-1, 64, 32, 32]	128
PReLU-15	[-1, 64, 32, 32]	1
Conv2d-16	[-1, 64, 32, 32]	36,864
BatchNorm2d-17	[-1, 64, 32, 32]	128
Conv2d-18	[-1, 64, 32, 32]	36,864
BatchNorm2d-19	[-1, 64, 32, 32]	128
PReLU-20	[-1, 64, 32, 32]	1
Conv2d-21	[-1, 64, 32, 32]	36,864
BatchNorm2d-22	[-1, 64, 32, 32]	128
Conv2d-23	[-1, 64, 32, 32]	36,864
BatchNorm2d-24	[-1, 64, 32, 32]	128
PReLU-25	[-1, 64, 32, 32]	1
Conv2d-26	[-1, 64, 32, 32]	36,864
BatchNorm2d-27	[-1, 64, 32, 32]	128
Conv2d-28	[-1, 64, 32, 32]	36,864
BatchNorm2d-29	[-1, 64, 32, 32]	128
Conv2d-30	[-1, 256, 32, 32]	147,456
PixelShuffle-31	[-1, 64, 64, 64]	0
PReLU-32	[-1, 64, 64, 64]	1
Conv2d-33	[-1, 256, 64, 64]	147,456
PixelShuffle-34	[-1, 64, 128, 128]	0
PReLU-35	[-1, 64, 128, 128]	1
Conv2d-36	[-1, 3, 128, 128]	15,552

Before running the training code, you can change the number of epochs and batch size based on your machine. My settings are as follows. I used the V100 and didn't run into a memory issue.

```
[ ]  #change the batch-size based on your system memory

     #lower to not crash it
     #12.7 GB
     epochs=150
     batch_size=64
```

Next, set up the base path of where to save your data. It is important that you select somewhere that is not local to google colab if you use it since training can take a long time and if the session crashes, you don't want to lose everything.

```
[ ]  import os
     #base_path = os.getcwd()
     #set base path to your google drive folder where you are saving everything
     base_path = Path('/content/drive/MyDrive/University/Applied AI/Midterm')

     #lr_path = os.path.join(base_path,"trainImages")
     #hr_path =celebData
     hr_path =catDogData
     #valid_path = os.path.join(base_path,"SR_valid")
     weight_file = os.path.join(base_path,"SRPT_weights")
     out_path = os.path.join(base_path,"out")

     if not os.path.exists(weight_file):
         os.makedirs(weight_file)

     if not os.path.exists(out_path):
         os.makedirs(out_path)


     #LR_images_list = os.listdir(lr_path)
     HR_images_list = imageList
     batch_count = len(HR_images_list)//batch_size
     batch_count
```

If you are just starting your training, you can leave the code as is.

```
[ ]  #model = load_checkpoint(Path('/content/drive/MyDrive/University/Applied AI/Midterm/SRPT_weights/SR148.pth'))
```

```
[ ]  #batch_count=60
     for epoch in range(epochs):
         d1loss_list=[]
         d2loss_list=[]
         gloss_list=[]
         vloss_list=[]
         mloss_list=[]
```

However, if you are training from a checkpoint, uncomment the first line and select the checkpoint you want to load. Then change the range in the loop to start on the epoch after the one that was loaded.

```
[ ]  model = load_checkpoint(Path('/content/drive/MyDrive/University/Applied AI/Midterm/SRPT_weights/SR148.pth'))
```

```
[ ]  #batch_count=60
     for epoch in range(149, epochs+1):
         d1loss_list=[]
         d2loss_list=[]
         gloss_list=[]
         vloss_list=[]
         mloss_list=[]
```

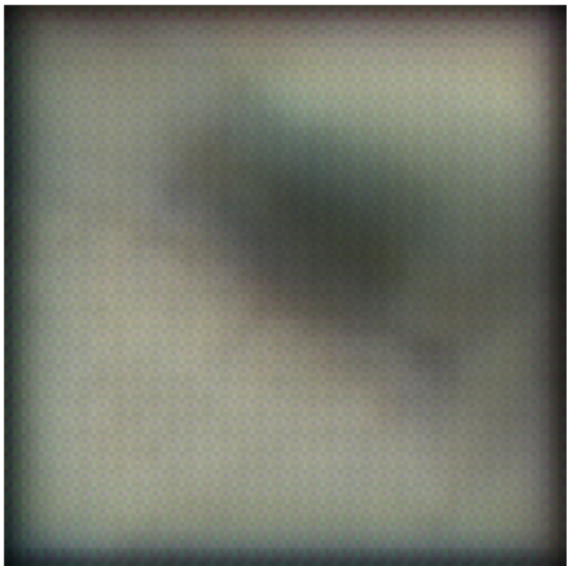
The training will make a save every 3 epochs and output a sample image. Both the model and the sample image will be saved to your specified save location. If you want to save more or less often, you can change the number here.

```
if(epoch%3==0):

    checkpoint = {'model': Generator(),
          'input_size': 32,
          'output_size': 128,
          'state_dict': gen.state_dict()}
```

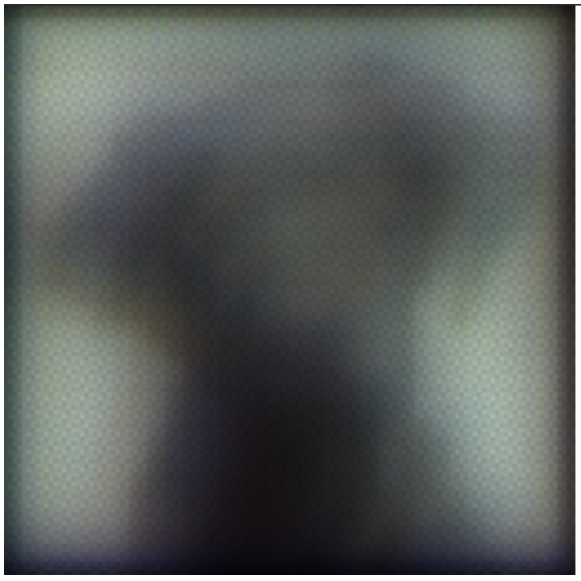
Results From Training

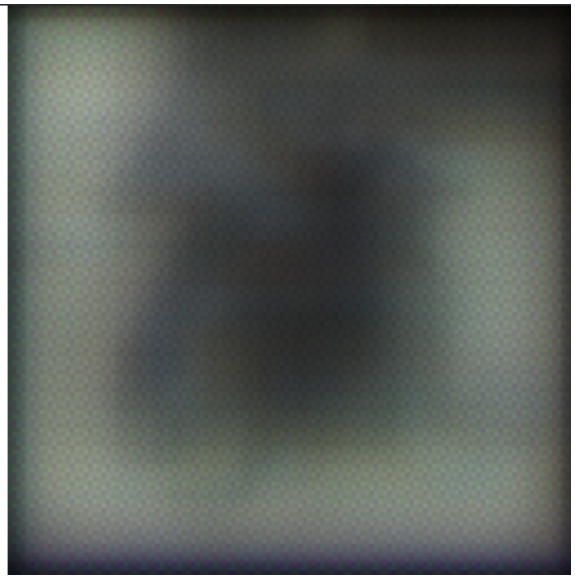
The following are some samples of the SRGAN's progress over the 150 epochs. This whole process with the V100 took about 13 hours.

Epoch 1



Epoch 51





Epoch 102

Epoch 150

The output starts out almost unrecognizable and slowly becomes more and more clear. However, it seems that the 150 epochs was not quite enough and would require significantly more training in order to create truly high resolution images.

Generating Images with the GAN

Next, we want to create images to train our classifier. Update the following path to your best weights.

```
[ ]  load_checkpoint(Path('/content/drive/MyDrive/University/Applied AI/Midterm/SRPT_weights/SR151.pth'))
```

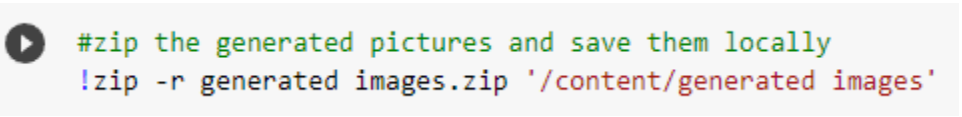
Update catDogData to your high resolution images, pathToLastModel to your best model, and change the (32,32) to whatever size you are shrinking the images to. For the batch size, select a number based on your memory. This section will take a little while to convert all of the images but can easily run out of memory if the batch size is set too high.

```
#load images into list
catDogData = '/content/train'
imageDirectory = Path(catDogData)
images = os.listdir(catDogData)
imageList = images[:25000]
path = catDogData
#path to the best trained model
pathToLastModel = Path('/content/drive/MyDrive/University/Applied AI/Midterm/SRPT_weights/SR151.pth')
#assign the generator
generator = Generator()
#make folder to save to locally
tempSaveLocation = Path('/content/generated images')
#create temporary save directory
if not os.path.exists(tempSaveLocation):
    os.makedirs(tempSaveLocation)
torch.cuda.empty_cache()
imagelist=[]
namelist = []
model = load_checkpoint(pathToLastModel)
batchSize = 60
for img in images:
    namelist.append(img) #save name of image
    resizedImage = cv2.resize(cv2.GaussianBlur(cv2.imread(os.path.join(hr_path,img)),(5,5),cv2.BORDER_DEFAULT),(32,32))
    imagelist.append(resizedImage)
    if(len(imagelist) >= batchSize or (len(os.listdir(tempSaveLocation)) + len(imagelist)) >= len(images)):
```

A sample of the output as it converts the images is shown below.

```
progress: 60 images saved. 24988 images total.
progress: 120 images saved. 24988 images total.
progress: 180 images saved. 24988 images total.
progress: 240 images saved. 24988 images total.
progress: 300 images saved. 24988 images total.
progress: 360 images saved. 24988 images total.
progress: 420 images saved. 24988 images total.
progress: 480 images saved. 24988 images total.
progress: 540 images saved. 24988 images total.
progress: 600 images saved. 24988 images total.
progress: 660 images saved. 24988 images total.
progress: 720 images saved. 24988 images total.
progress: 780 images saved. 24988 images total.
progress: 840 images saved. 24988 images total.
progress: 900 images saved. 24988 images total.
progress: 960 images saved. 24988 images total.
progress: 1020 images saved. 24988 images total.
progress: 1080 images saved. 24988 images total.
progress: 1140 images saved. 24988 images total.
progress: 1200 images saved. 24988 images total.
progress: 1260 images saved. 24988 images total.
progress: 1320 images saved. 24988 images total.
progress: 1380 images saved. 24988 images total.
progress: 1440 images saved. 24988 images total.
progress: 1500 images saved. 24988 images total.
progress: 1560 images saved. 24988 images total.
progress: 1620 images saved. 24988 images total.
progress: 1680 images saved. 24988 images total.
progress: 1740 images saved. 24988 images total.
progress: 1800 images saved. 24988 images total.
```

Once this block is done, go ahead and zip the images up for later. This is important so that you don't have to run this section again if it crashes. Download the now zipped images somewhere externally.



Training The Classifiers

You can run the training section without running any other part of the notebook. First, get your generated images and your high definition images. Change the paths to where your data is located.

```
[ ]  # Images that came OUT OF the gan
     !unzip '/content/drive/MyDrive/University/Applied AI/Midterm/Generated Images/generated_train.zip' -d '/content'
     # Images that went INTO the gan
     !unzip '/content/drive/MyDrive/University/Applied AI/Midterm/Data/train.zip' -d '/content'
```

Change the following:

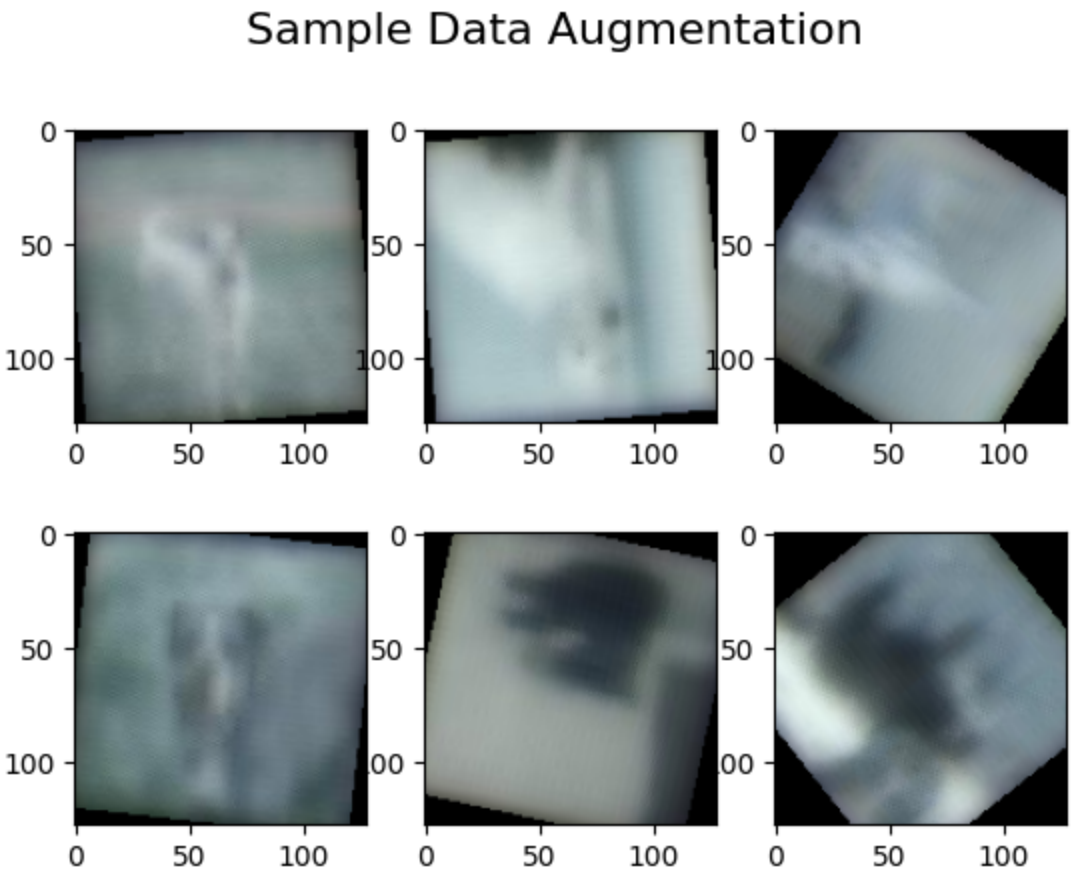
- image sizes to the output of the SRGAN (in my case 128x128x3).
- Data directory path. You will be training the generated data or the high definition data one at a time. Replace the path with the path to the data set you are currently training.
- Csv directory path. Place the path to the csv files with the classifications.
- For the last 2, put the paths and file names to your training and verification csv files respectively.

```
fold_num = 1

# Image size
img_width, img_height, img_depth = 128, 128, 3
data_dir = Path('/content/content/generated images')
csv_dir = Path('/content/drive/MyDrive/University/Applied AI/Midterm/Data')

# Data organization
csv_name = 'dog_cat_train.csv'
csv_path = csv_dir / csv_name
df_train = pd.read_csv(csv_path)
df_train = df_train.sample(frac = 1, random_state = fold_num)

csv_name = 'dog_cat_valid.csv'
csv_path = csv_dir / csv_name
df_valid = pd.read_csv(csv_path)
df_valid = df_train.sample(frac = 1, random_state = fold_num)
```

Next, you can change any of the training parameters including how many different classes there are.

```
# Training parameters
epochs = 10
freq = 20
batch_size = 34
num_classes = 2
train_valid_split = 0.2
nb_train_samples = int(len(df_train) * (1-train_valid_split))
nb_valid_samples = len(df_train) - nb_train_samples
```

Some sample data augmentations will be shown with your images. The following is mine.



Under Create/Import Network, you can change the model. I went with MobileNetV2 but you can change it to whatever you want.

```
# Build Model
image_input = Input(shape = (img_width, img_height, img_depth))
base_model = applications.MobileNetV2(input_tensor = image_input,
                                      include_top = False,
                                      weights = 'imagenet')
```

You can then change the hyperparameters and metrics for training here.

```
[ ]  # Compile the model
     opt = Adam(learning_rate = 0.00001)
     model.compile(loss = loss_fun, optimizer = opt, metrics = ['accuracy'])
```

Under model checkpointer, change the directories to somewhere external. This will be where the training will save the model and logs.

```
# Folder setup
init_time = datetime.now()
current_time = init_time.strftime('%Y%m%d_%H%M%S')
name_dir = '/content/drive/MyDrive/University/Applied AI/Midterm/Generated checkpoints/Trained Models' + current_time
os.mkdir(name_dir)

# Callbacks1: ModelCheckpointer
model_file_format = name_dir + '/model_' + str(fold_num) + '_model.{epoch:04d}.hdf5'
check = ModelCheckpoint(model_file_format, period = freq, verbose = 1)

# Callbacks2: TensorBoard
tensor_check = '/content/drive/MyDrive/University/Applied AI/Midterm/Generated checkpoints/Generated Logs' + current_time + '_train_testsplit' + str(fold_num)
tensor_board = TensorBoard(tensor_check)
```

Training Log

Finally, train your model. Here are my outputs for each dataset.Training was run on a T4 GPU

Original Data Set

```
Epoch 1/10
470/470 [==============================] - 131s 268ms/step - loss: 0.9647 - accuracy: 0.7439
Epoch 2/10
470/470 [==============================] - 139s 296ms/step - loss: 0.7130 - accuracy: 0.7968
Epoch 3/10
470/470 [==============================] - 139s 295ms/step - loss: 0.6344 - accuracy: 0.8068
Epoch 4/10
470/470 [==============================] - 119s 254ms/step - loss: 0.5822 - accuracy: 0.8149
Epoch 5/10
470/470 [==============================] - 139s 296ms/step - loss: 0.5232 - accuracy: 0.8289
Epoch 6/10
470/470 [==============================] - 119s 252ms/step - loss: 0.5099 - accuracy: 0.8349
Epoch 7/10
470/470 [==============================] - 118s 252ms/step - loss: 0.4955 - accuracy: 0.8357
Epoch 8/10
470/470 [==============================] - 138s 294ms/step - loss: 0.4764 - accuracy: 0.8411
Epoch 9/10
470/470 [==============================] - 119s 253ms/step - loss: 0.4548 - accuracy: 0.8499
Epoch 10/10
470/470 [==============================] - 139s 295ms/step - loss: 0.4479 - accuracy: 0.8498
```

SRGAN Data Set

```
Epoch 1/10
470/470 [==============================] - 120s 224ms/step - loss: 1.5992 - accuracy: 0.4965
Epoch 2/10
470/470 [==============================] - 105s 224ms/step - loss: 1.2916 - accuracy: 0.4962
Epoch 3/10
470/470 [==============================] - 128s 272ms/step - loss: 1.1716 - accuracy: 0.4939
Epoch 4/10
470/470 [==============================] - 103s 220ms/step - loss: 1.1007 - accuracy: 0.4991
Epoch 5/10
470/470 [==============================] - 107s 228ms/step - loss: 1.0706 - accuracy: 0.4984
Epoch 6/10
470/470 [==============================] - 105s 223ms/step - loss: 1.0482 - accuracy: 0.4992
Epoch 7/10
470/470 [==============================] - 104s 221ms/step - loss: 1.0297 - accuracy: 0.4998
Epoch 8/10
470/470 [==============================] - 110s 235ms/step - loss: 1.0136 - accuracy: 0.5016
Epoch 9/10
470/470 [==============================] - 108s 230ms/step - loss: 1.0052 - accuracy: 0.5039
Epoch 10/10
470/470 [==============================] - 116s 247ms/step - loss: 0.9999 - accuracy: 0.4992
```

It can be noted that the dataset trained with the generated images did significantly worse than the high definition dataset. This is to be expected based on the quality of the generated images.

Testing The Classifiers

Change the paths to your generated image dataset model and your high definition image dataset model. Load one of them by commenting out the other. Additionally, select any metrics you would like to track during testing.

```
[ ] GanModelLocation = Path('/content/drive/MyDrive/University/Applied AI/Midterm/Generated checkpoints/Trained Models20231106_193627/fold_num_1early_stop_model.hdf5')
    NormalModelLocation = Path('/content/drive/MyDrive/University/Applied AI/Midterm/Normal checkpoints/Trained Models20231106_150807/fold_num_1early_stop_model.hdf5')

    model.load_weights(GanModelLocation)
    #model.load_weights(NormalModelLocation)

    model.compile(loss = loss_fun, optimizer = opt, metrics = ['accuracy', 'AUC', 'FalseNegatives', 'FalsePositives'])
```

Change the testing data data path to your own. Then specify the csv file for your testing images and the path to reach it. Finally, change the class names to match your own dataset.

```
[ ] # Data organization
    test_data_dir = Path('/content/train')
    csv_name = 'dog_cat_test.csv'
    csv_path = Path('/content/drive/MyDrive/University/Applied AI/HW1') / csv_name
    df_test = pd.read_csv(csv_path)
    df_test = df_test.sample(frac = 1, random_state = fold_num)

    class_names = ['A. cat', 'B. dog']
```

Then just run the testing cell for each of your models and compare. The following shows my log.

Testing Log

Original Data Set

```
25/25 [==============================] - 16s 556ms/step - loss: 0.3573 - accuracy: 0.8291 - auc: 0.9192 - false_negatives: 427.0000 - false_positives: 427.0000
model.metrics_names ['loss', 'accuracy', 'auc', 'false_negatives', 'false_positives']
scores [0.3572516143321991, 0.8291316628456116, 0.9192368984222412, 427.0, 427.0]
```

SRGAN Data Set

```
25/25 [==============================] - 14s 509ms/step - loss: 0.7437 - accuracy: 0.4994 - auc: 0.4759 - false_negatives: 1251.0000 - false_positives: 1251.0000
model.metrics_names ['loss', 'accuracy', 'auc', 'false_negatives', 'false_positives']
scores [0.7436501979827881, 0.49939975142478943, 0.47589412331581116, 1251.0, 1251.0]
```

Metric	Original Dataset Model	Generated Dataset Model
Loss	0.3573	0.7437
Accuracy	0.8291	0.4994
auc	0.9192	0.4759
False negatives	427	1251
False positives	427	1251

The original dataset model performed much better than the generated one which seemed to just be guessing with its accuracy of 0.4994.