Super GAN Implementation for Super-Resolution Image Generation and Binary Image Classification

1. Introduction

In this report, we will discuss the implementation of a Super GAN for super-resolution image generation and its integration with a binary image classification task using the Kaggle Cats vs. Dogs dataset. The report is organized as follows:

2. Dataset Description

The Kaggle Cats vs. Dogs dataset consists of a large collection of images of cats and dogs. It is commonly used for binary image classification tasks. For this report, we will use this dataset to train and test a binary classifier to distinguish between cats and dogs: https://www.kaggle.com/c/dogs-vs-cats/data.

Steps

- Split the dataset into train & validation for training
- Train the model_A and test the performance (run model_A_binary_classifier.ipynb)
- Train SRGAN, low res. Image = 32x32, high res. image = 128x128, epoch = 150
- Run create_SR_dataset.ipynb→SRGAN_final.ipynb, to generate and and save images
- Run model_B.ipynb to test the quality of the generated images
- SRGAN_attempt1.ipynb is a test program to generate some data with less epochs; result is not satisfactory.

3. Implementation Details

3.1 Binary Classifier for Cats vs. Dogs (model_A)

Architecture: The architecture for a binary classifier used for classifying cats vs. dogs typically involves a convolutional neural network (CNN). Here's a high-level overview of the architecture:

1. **Input Layer**: The input layer receives the images with three color channels (e.g., Red, Green, Blue) and the specified input size (e.g., 224x224 pixels).
2. **Convolutional Layers**: A series of convolutional layers are used to extract features from the input images. These layers consist of multiple filters that learn to recognize patterns, edges, and textures in the images.
3. **Activation Functions**: Non-linear activation functions (e.g., ReLU - Rectified Linear Unit) are applied after each convolutional layer to introduce non-linearity.
4. **Pooling Layers**: Max-pooling or average-pooling layers are used to reduce the spatial dimensions of the feature maps and decrease the number of parameters in the model.
5. **Flattening Layer**: The output from the convolutional and pooling layers is flattened into a 1D vector.
6. **Dense (Fully Connected) Layers**: One or more dense layers follow the flattening layer. These layers can capture high-level features and relationships in the data.
7. **Output Layer**: The output layer consists of a single neuron with a sigmoid activation function, which is suitable for binary classification. It produces an output value between 0 and 1, where values closer to 0 represent one class (e.g., cat) and values closer to 1 represent the other class (e.g., dog).
8. **Model Compilation**: The model is compiled with a loss function (commonly binary cross-entropy), an optimizer (e.g., Adam), and evaluation metrics (e.g., accuracy).

Evaluation Metrics:

1. **Accuracy**: The proportion of correctly classified images out of the total number of images in the testing dataset.
2. **Precision**: The fraction of true positive predictions (correctly identified dogs) out of all positive predictions (all instances predicted as dogs).
3. **Recall**: The fraction of true positive predictions (correctly identified dogs) out of all actual positive instances (all actual dogs).
4. **F1-Score**: The harmonic mean of precision and recall, which balances the trade-off between precision and recall.
5. **Confusion Matrix**: A table that shows the true positives, true negatives, false positives, and false negatives, providing insights into the model's performance.

3.2 Super GAN for Super-Resolution: (https://github.com/jbhuang0604/SelfExSR)

A brief architectural view of SRGAN: we train a generator network as a feed-forward CNN GθG parametrized by θG. Here θG = {W1:L; b1:L} denotes the weights and biases of a L-layer deep network and is obtained by optimizing a SR-specific loss function lSR.

$$\hat{\theta}_G = \arg\min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} l^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR}) \qquad (1)$$

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{\text{train}}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))] \qquad (2)$$

$$l^{SR} = \underbrace{\underbrace{l_{\text{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}} \qquad (3)$$

…

……

4. Experiments and Results

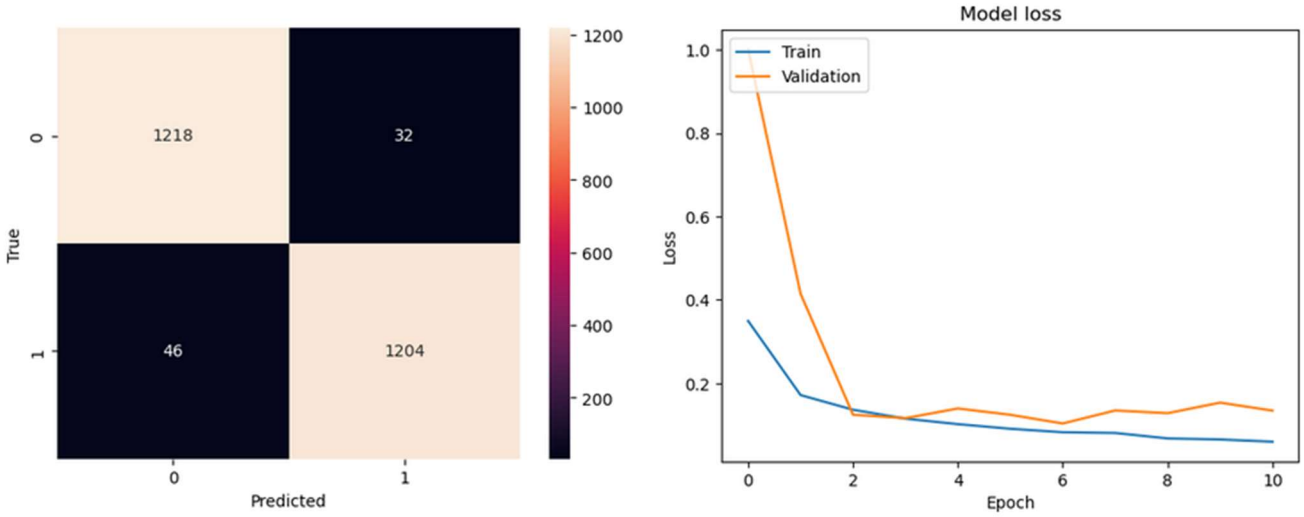
4.1 Training and Testing the Binary Classifier (model_A)

testing

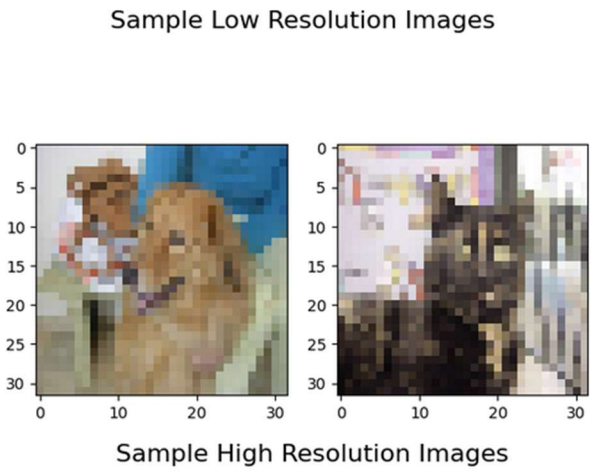
```
In [13]: test_loss, test_accuracy = model.evaluate(test_gen, steps=len(test_gen), verbose=1)

         print(f'Test Loss: {test_loss:.4f}')
         print(f'Test Accuracy: {test_accuracy:.4f}')
```

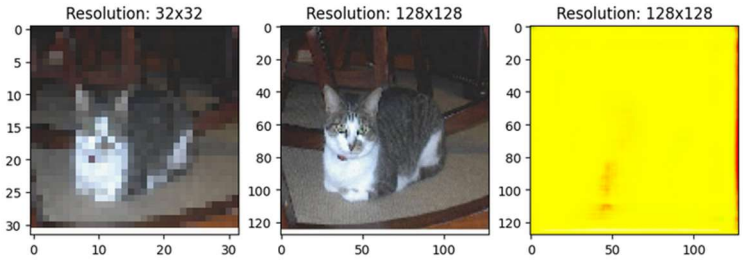
```
50/50 [==============================] - 27s 533ms/step - loss: 0.1404 - accuracy: 0.9688
Test Loss: 0.1404
Test Accuracy: 0.9688
```

F1_modelA: .968

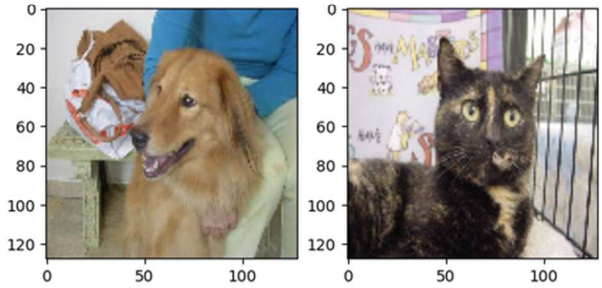


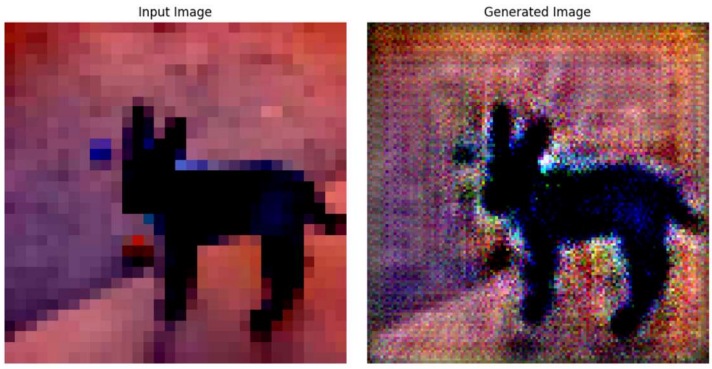
4.2 Training and Testing Super GAN





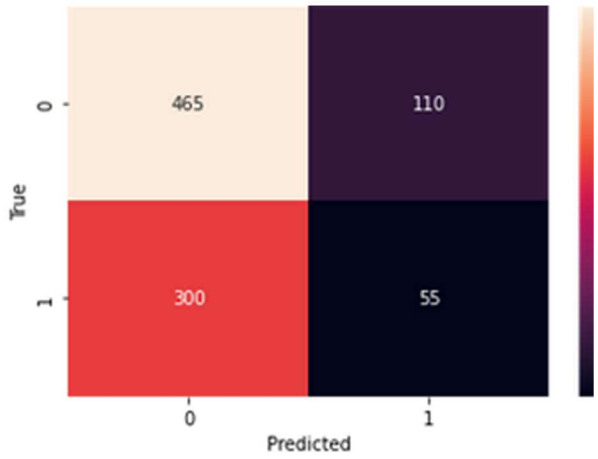
After epoch = 1

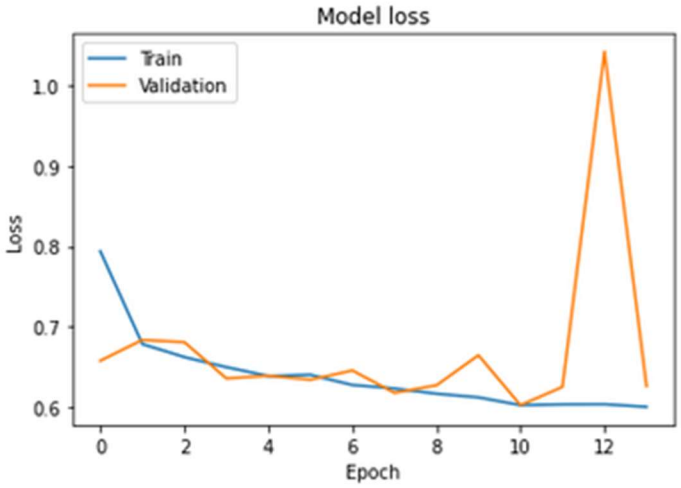




After final iteration

4.3 Evaluating the Image Classification Model (model_B)





Loss: .44

Accuracy: .5591

F1: .211

5. Discussion

Super GAN tries to tackle the challenge of generating high resolution image from low resolution images. Here, we tried to implement the code to generate high resolution cat, dog images. The computation was quite heavy and time consuming. Due to time limitation and internal structure, inspite of using gpu, it couldn't complete the whole 150 epochs hence less accuracy. After using the generated images from the already finished epochs, the performance was measured. If the issue of slow training could be resolved, the accuracy would be improved further.

7. References

https://github.com/jbhuang0604/SelfExSR