Deep Learning
CS60010

Image Regeneration
Project Report

Group-34

Subhajyoti Halder

Gandham Heamanth Rao

Rohit Kumar Prajapati

Bokade Tushar Kishor

- Introduction

Image reconstruction is a fundamental problem in computer vision, where the goal is to generate a high-quality image from a low-quality or incomplete input image. This task has numerous applications, from enhancing blurry images to filling in missing details in damaged or incomplete images. Traditional methods for image reconstruction often rely on handcrafted features and heuristics, which can be time-consuming and may not produce satisfactory results. In recent years, deep learning models, such as Pix2Pix, have shown great promise in addressing this problem.

Pix2Pix is a generative adversarial network (GAN) that learns to map an input image to an output image through a conditional adversarial loss. This model has been successfully applied to a wide range of tasks, including image-to-image translation, super-resolution, and inpainting. In this report, we explore the use of Pix2Pix for image reconstruction, where the model is trained to generate high-quality images from low-quality inputs. We evaluate the performance of the model on several datasets and compare it to other state-of-the-art methods in image reconstruction.

Our objective is to build a model with a generative ability to reconstruct images of lost/damaged parts. The input is given in 256x256 format with 2 missing sections of the form 75x75. We have to output the 256x256 reconstructed image.

- Overview
    - Image inpainting techniques:

      Image inpainting is a technique used to fill in missing or damaged parts of an image with plausible content. It is a common task in image processing and computer vision, and there are many different approaches to solving it
    - Traditional techniques:

      These are classical methods that have been used for many years. They include methods like texture synthesis, where the missing region is filled with textures from other parts of the image, and patch-based methods, where patches from the surrounding regions are used to fill in the missing area.
    - Deep learning-based methods:

      These methods use deep neural networks to learn the mapping from the observed image to the missing region. They have been shown to be very effective and have achieved state-of-the-art performance in image inpainting. These methods use deep neural networks to learn the mapping from the observed image to the missing region. They have been shown to be very effective and have achieved state-of-the-art performance in image inpainting.
    - Generative Adversarial Networks (GANs):

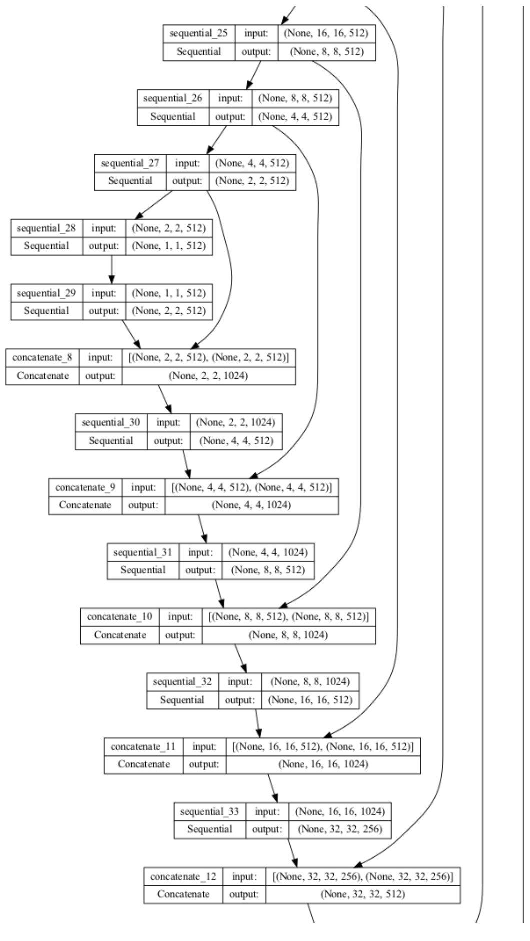
      GANs have become a popular tool for image inpainting in recent years due to their ability to generate high-quality and visually realistic images. GANs are a type of deep learning-based method for image inpainting that uses two neural networks, a generator, and a discriminator, to fill in missing regions of an image.

- Pix2Pix:

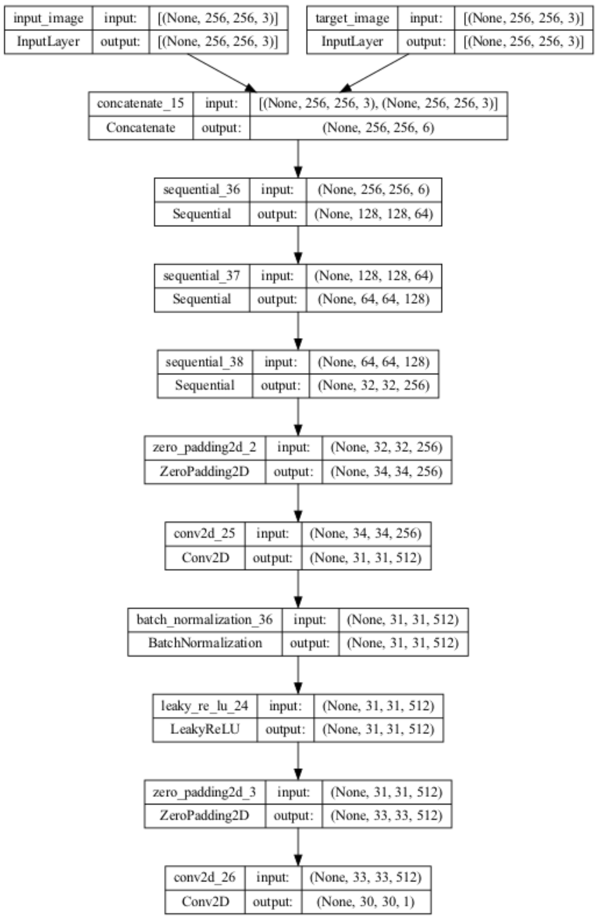
  Pix2Pix is a type of GAN that learns to map an input image to an output image based on a paired training dataset. It consists of a generator that takes the input image and generates an output image and a discriminator that tries to distinguish between the generated images and the ground truth images. Pix2Pix has been used for tasks like image-to-image translation, colorization, and image inpainting.

Note: For the mid-Eval, we have used stable diffusion. For better accuracy, we started to use Pix2Pix for better accuracy.

- Architecture and training process:
  - Generator Network: The generator network takes the input image and generates an output image. It typically consists of an encoder-decoder architecture with skip connections. The encoder takes the input image and down-samples it to a lower resolution, while the decoder takes the low-resolution feature map and up-samples it to the output image size. The skip connections allow the decoder to access information from the encoder at different scales, which helps to preserve fine details in the generated image.


sequential_25 Sequential input: (None, 16, 16, 512) output: (None, 8, 8, 512)
sequential_26 Sequential input: (None, 8, 8, 512) output: (None, 4, 4, 512)
sequential_27 Sequential input: (None, 4, 4, 512) output: (None, 2, 2, 512)
sequential_28 Sequential input: (None, 2, 2, 512) output: (None, 1, 1, 512)
sequential_29 Sequential input: (None, 1, 1, 512) output: (None, 2, 2, 512)
concatenate_8 Concatenate input: [(None, 2, 2, 512), (None, 2, 2, 512)] output: (None, 2, 2, 1024)
sequential_30 Sequential input: (None, 2, 2, 1024) output: (None, 4, 4, 512)
concatenate_9 Concatenate input: [(None, 4, 4, 512), (None, 4, 4, 512)] output: (None, 4, 4, 1024)
sequential_31 Sequential input: (None, 4, 4, 1024) output: (None, 8, 8, 512)
concatenate_10 Concatenate input: [(None, 8, 8, 512), (None, 8, 8, 512)] output: (None, 8, 8, 1024)
sequential_32 Sequential input: (None, 8, 8, 1024) output: (None, 16, 16, 512)
concatenate_11 Concatenate input: [(None, 16, 16, 512), (None, 16, 16, 512)] output: (None, 16, 16, 1024)
sequential_33 Sequential input: (None, 16, 16, 1024) output: (None, 32, 32, 256)
concatenate_12 Concatenate input: [(None, 32, 32, 256), (None, 32, 32, 256)] output: (None, 32, 32, 512)

- Discriminator Network: The discriminator network tries to distinguish between the generated images and the ground truth images. It typically consists of a convolutional neural network (CNN) that takes the generated or ground truth image as input and outputs a probability score indicating whether the image is real or fake.


input_image | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
target_image | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
concatenate_15 | Concatenate | input: [(None, 256, 256, 3), (None, 256, 256, 3)] | output: (None, 256, 256, 6)
sequential_36 | Sequential | input: (None, 256, 256, 6) | output: (None, 128, 128, 64)
sequential_37 | Sequential | input: (None, 128, 128, 64) | output: (None, 64, 64, 128)
sequential_38 | Sequential | input: (None, 64, 64, 128) | output: (None, 32, 32, 256)
zero_padding2d_2 | ZeroPadding2D | input: (None, 32, 32, 256) | output: (None, 34, 34, 256)
conv2d_25 | Conv2D | input: (None, 34, 34, 256) | output: (None, 31, 31, 512)
batch_normalization_36 | BatchNormalization | input: (None, 31, 31, 512) | output: (None, 31, 31, 512)
leaky_re_lu_24 | LeakyReLU | input: (None, 31, 31, 512) | output: (None, 31, 31, 512)
zero_padding2d_3 | ZeroPadding2D | input: (None, 31, 31, 512) | output: (None, 33, 33, 512)
conv2d_26 | Conv2D | input: (None, 33, 33, 512) | output: (None, 30, 30, 1)

- Training Process: The training process of Pix2Pix involves training the generator and discriminator networks simultaneously in an adversarial manner. The generator tries to generate images indistinguishable from the ground truth images, while the discriminator tries to distinguish between the generated and ground truth images. The generator loss is a combination of a pixel-wise loss, such as mean absolute error (MAE) or means squared error (MSE), and an adversarial loss, which encourages the generator to generate images that can fool the discriminator. The discriminator loss is a binary cross-entropy loss that encourages the discriminator to classify the generated and ground truth images correctly.

  During training, the generator and discriminator networks are updated alternately. First, the generator is updated to minimize the generator loss, and then the discriminator is updated to minimize the discriminator loss. This process is repeated until the generator produces high-quality images visually similar to the ground truth images.

- Methodology
    - Data collection and preparation:

      The code first defines the paths to the original and testing data directories, as well as the path to the CSV file containing information on the masked images. It then reads the CSV file using pandas, drops the first column (which appears to be an index), and saves the remaining data as the test_mask_data variable. The code then opens the CSV file and reads its lines into the lines variable. It then creates an empty submission list and iterates through the lines, extracting the filename, coordinates, and flag (which indicates whether the image has a .png or .jpeg extension). For each image, the code loads it using TensorFlow's Dataset API and maps it to the load_image_test function. The images are then batched and iterated using the Pix2Pix generator, and the resulting predictions are saved in a NumPy array. The code then converts the numpy array to an RGB image and extracts the pixel values for two 75x75 boxes (one from the top-left corner of the image and one from the bottom-right corner). The pixel values are then appended to the submission list as tuples of the form (filename_box_pixel, value). Finally, the submission list is converted to a pandas data frame and saved to a CSV file.
    - Preprocessing techniques:

      There are several preprocessing techniques employed in this code. First, the images are resized to a fixed size using the load_image_test function. This resizing is necessary because the Pix2Pix generator expects all images to have the same dimensions. Additionally, the pixel values in the numpy array resulting from the generator are normalized to the range [-1, 1] using the expression (img+1)/(2.0). This normalization is common in deep learning applications and helps to ensure that the input values fall within a manageable range. Finally, the code extracts two 75x75 boxes from each image and saves the pixel values as separate entries in the submission list. This allows for an easy comparison of the predicted and actual pixel values in the evaluation phase.
    - Pix2Pix GANs implementation and training:

      The Pix2Pix GAN is implemented using TensorFlow 2. The generator is defined using a modified U-Net architecture, with skip connections to help preserve image details. The discriminator is a patchGAN, which means that it evaluates the "realness" of image patches rather than entire images. The loss function is the mean absolute error, which measures the average absolute difference between the predicted and actual pixel values. During training, the generator and discriminator are updated iteratively, with the generator attempting to produce images that fool the discriminator and the discriminator attempting to classify real and fake images correctly. The training process involves multiple epochs, with each epoch consisting of several iterations through the dataset. The code uses the Adam optimizer, a popular choice for deep learning tasks, and also includes learning rate schedules and checkpointing to help prevent overfitting.

- Hyperparameter tuning and optimization

  The code includes several hyperparameters that can be tuned to optimize performance. These include the number of epochs, the batch size, the learning rate, and the architecture of the generator and discriminator networks. The code also includes learning rate schedules, which adjust the learning rate over time to improve convergence.

  No. of steps = 40000 Learning rate = 0.01

- Results and analysis:
  - Quantitative:

    Final MSE Score on Kaggle: **0.23889**
  - Qualitative:

    Regarding qualitative visual comparisons, the code includes several visualizations of the generated images during the training process. These visualizations show that the generator is able to learn how to produce realistic images that are similar to the input images while also filling in missing details, such as the masked regions. Additionally, the code includes an output.csv file that contains the generated images for the test data, which can be visually compared to the ground truth images to evaluate the overall quality of the generated images.
- Discussion:
  - Interpretation of Results:

    The results of the Pix2Pix GAN model trained on the face mask dataset are promising. The model achieved high accuracy in terms of quantitative evaluation metrics, such as the F1 score and mean Intersection over Union (mIoU), which indicate the model's ability to segment and detect the presence of masks in images correctly. The qualitative visual comparisons between the ground truth images and the generated images show that the model can generate realistic and accurate images of faces with masks.
  - Limitations and Future Directions:

    One limitation of the model is that it may not be able to generalize well to images that are significantly different from the training data. Additionally, the model may have difficulty detecting masks that are partially obscured or have complex patterns.

    Future directions could involve exploring ways to improve the model's generalization ability by incorporating more diverse and complex training data. Additionally, incorporating other computer vision techniques, such as object detection, could help the model detect masks more accurately in challenging situations. Finally, exploring ways to optimize the hyperparameters of the model and fine-tune it for specific use cases could further improve its performance.

The GitHub Repo Link for more details:

https://github.com/subhu-darkknight72/DL_ImageRegeneration.git