Final Project CV2.
Implementation and Improvement of model Pix2Pix

Brais Pérez Vázquez
Master in Artificial Intelligence
Universidade de Vigo
Ourense
Email: 14brais@gmail.com

Cristian Pérez Gómez
Master in Artificial Intelligence
Universidade de Vigo
Ourense
Email: cpgomez18@esei.uvigo.es

***Abstract*—In this project we will implement different types of GANS to generate a realistic image from a segmented image in labels. More concretely in our case we will use a CityScapes dataset, where the segmented images are part of the images of different European cities. For the implementation of the GANS we will apply a Pix2Pix model with a generator with a UNet architecture, then we will modify the generator to apply a ResNet architecture and finally we will make different tests mixing architectures.**

1. Introduction

In this project we will try to develop a tool that converts only labelled images into real images, that is to say, we will try to develop a generative Artificial Intelligence that creates these images with the best possible reliability. The images used in this project are the images known as *'CityScapes'* [1] corresponding to images of different locations of different European cities.

An example of the tool we will try to implement is the following, on the left the input of the model, on the middle the real image and on the left the output of the model (in this case is a early stopping model with very poor results):

Figure 1. Example of the model images

For this task we want to implement we will use 3 different tools and try to compare their results. We will use Python to implement the code and the models will be defined in the TensorFlow package [2].

Throughout this document we will explain step by step and in detail the process followed to carry out the project. To do so, we will divide the explanation into four sections, this first section where we are introducing the concepts, a second section where we will explain all the technical part of the problem and the models, a third part where we will explain and analyse the results obtained and finally some conclusions where we will finish and give the interpretations of everything obtained.

2. Technical Approach

As we have already advanced, in this section we will deal with the more technical explanations of the project. To do this, first we must understand that our intention is to create a GAN (Generative Adversial Network), therefore we must implement a generator that will create the images and a discriminator that will be the model that will say if these images are true or not, so we must go into detail what architecture we will use for both models. Furthermore, one of the major complexities involved in GANs is to evaluate the models, even more so in this case, where our output is an image, so we must compare two images and try to parameterise the similarity between them.

2.1. Metrics

As this is one of the most complex parts, the first point we will discuss is the metrics used. In our case, we will use 4 metrics which we explain below:

1) ***Histogram Correlation*** *(Corr.)* [3]:It is a metric used to assess the similarity between histograms, which represent the distribution of pixel intensities in an image. It measures how closely the histograms of two images match. In our project, we transform the images into grayscale before comparing their histograms. A correlation value close to 1 indicates a high degree of similarity, suggesting that the pixel intensity distributions of the images are similar. Conversely, a value close to -1 indicates a high level of dissimilarity, implying that the images have significantly different pixel intensity distributions.

2) ***Template Matching*** *(T.M.)* [4]: It is a technique employed to compare a small template image with different regions of a larger target image. While typically used to detect a specific part within an image, in our project, we adapt template matching to attempt to detect the entire image. The objective remains to find areas within the target image where the template closely matches a portion of the target. The template matching score indicates the degree of similarity between the template and the image patch at the location with the highest correlation. This score ranges from -1 to 1, with 1 indicating a perfect match between the template and the image patch, and -1 indicating a perfect mismatch.
3) ***Feature Extraction*** *(F.E.)* [5]: It involves identifying and extracting distinctive characteristics or keypoints from an image. After feature extraction, a matching ratio is calculated to evaluate the similarity between the extracted features of two images. The matching ratio ranges from 0 to 1, where 0 signifies that none of the keypoints are matching between the images, while 1 indicates a perfect match, with all keypoints aligning perfectly.
4) ***Similarity, SSIM (Structural Similarity Index)*** [6]: SSIM, or Structural Similarity Index, is a method for quantifying the similarity between two images. It compares three components—luminance, contrast, and structure—locally across the entire image. SSIM produces a similarity index for each pixel, reflecting how similar the local structures are between the two images. These local similarity indices are then averaged to compute the overall SSIM value for the entire image. SSIM values range from -1 to 1, where 1 indicates perfect similarity, and values closer to 1 denote higher similarity in terms of luminance, contrast, and structure.

Despite the use of these four metrics, the most robust of the four is the SSIM and will be taken as the most reliable. In addition to that, and without it being a conditioning factor, we will show a prediction of the model to see at a visual level the performance of the models.

2.2. Models

As we have already anticipated, for this model we will use a GAN and our first test will be about the Pix2Pix model, for this we will base our code on the already implemented TensorFlow model [7].

We will not go into further detail on the code used, but simply discuss more specifically the architectures implemented in the generator and the discriminator. In addition, as in all three approaches we will use the same discriminator, we will proceed to explain it below and in each of the sections we will simply explain the architecture used in the generator.

The discriminator architecture for our GAN project is designed to assess the authenticity of input images. It begins with two input layers for the input and target images, followed by concatenation of these images along the channel axis. This concatenated tensor undergoes a series of down-sampling layers, which progressively reduce the spatial dimensions of the feature maps while increasing the number of channels. Zero-padding layers ensure compatibility with subsequent convolutional operations. Two convolutional layers further process the feature maps, with the weights initialized using a normal distribution. Batch normalization stabilizes training, and a LeakyReLU activation function introduces non-linearity. Finally, the output of the last convolutional layer is a single-channel feature map representing the discriminator's prediction of image authenticity. This architecture aims to effectively discriminate between real and generated images, providing crucial feedback to the generator network during GAN training.

2.2.1. First Model: Pix2Pix. In this first model we will use the same architecture as the one given in the TensorFlow tutorial, i.e. the generator will follow a UNet architecture [8]:

- The ***generator*** comprises down-sampling layers, which reduce the spatial dimensions of the input image, and up-sampling layers, which increase the spatial dimensions back to the desired output size. Each down-sampling layer typically consists of convolutional operations followed by normalization and activation functions, while each up-sampling layer involves transpose convolutional operations, normalization, and activation functions. The architecture also incorporates skip connections, which concatenate feature maps from earlier layers with those from later layers during up-sampling to preserve fine details. Finally, the model includes a final convolutional layer to produce the output image.

2.2.2. Second Model: ResNet instead UNet. In this second model we use an architecture similar to a ResNet because, in general, despite having a higher computational cost, it obtains better results due to the fact that it avoids the problem of the vanishing gradient [9].

- The ***generator*** comprises several residual blocks, each consisting of two convolutional layers with batch normalization and ReLU activation. These residual blocks aid in learning the residual mapping, making training more effective. The generator first applies an initial convolutional layer to the input image, followed by several downsampling blocks to capture high-level features. Then, it employs upsampling blocks to upscale the features back to the original size, and finally, it applies a convolutional layer to generate the output image. The use of residual blocks helps mitigate the vanishing gradient problem, enabling the model to learn complex mappings more effectively. Additionally, the option to include dropout in the residual blocks allows for regularization during training, preventing overfitting.

2.2.3. Third Model: Ensemble Different Techniques. In this model we wanted to test an architecture that was not predefined [10] [11]. For this we mixed elements from different types of GANS:

- The ***generator*** blends concepts from various architectures like *DCGAN*, *ResNet*, and *StyleGAN*. It begins with an initial convolutional layer followed by downsampling blocks, akin to DCGAN, which progressively reduce spatial dimensions while increasing feature depth. These blocks utilize convolutional layers with batch normalization and LeakyReLU activation for feature extraction. Next, residual blocks, similar to ResNet, are employed to facilitate learning residual mappings and alleviate the vanishing gradient problem. These blocks enrich the model's ability to capture complex features and improve training stability. Following the residual blocks, the model incorporates upsampling blocks using transposed convolutional layers to upsample the features back to the original size. Finally, a convolutional layer with tanh activation, reminiscent of StyleGAN, generates the output image

It should be noted that in all of them we reuse the code already given by the TensorFlow model, modifying the metrics and the training of the model, and in the last two models we also modify, as it is obvious, the architecture of the generator by the one we explain.

2.3. Training and Evaluating Methodology

Once we know the metrics we will use to evaluate the models and the models we will use, it is necessary to know how we will train, test and evaluate the models. To do this we will divide the set of CityScapes images into the three sets, with 400 images for the training set, 50 images for the validation set and another 50 images for the test set. All images will have a dimension of 256x256. Regarding the models, we will use two Adam optimisers and the loss of the discriminator will be Binary Crossentropy.

Regarding the training, in all three models we will use 60k steps for the total training, where every thousand steps will save the four metrics of the model, both for training and validation. Finally, once the model has finished training, we will use the test set to calculate the metrics. These metrics are calculated with the mean and standard deviation of all the images. All these metrics will be represented in graphs to compare the models in a simpler and more intuitive way. Finally, we will also store the execution time of each model every thousand steps. To do this, we will try to train the models on the same computer and with similar conditions so that the comparison is as objective as possible. All these metrics will be taken into account when drawing conclusions and explaining the results obtained.

It should be noted that we could also perform fine tuning with the models for the hyperparameters. In addition we could perform some kind of data augmentation to add more images to the training and so on.

3. Results

Once we have discussed the models and how we have implemented them, we will continue by representing the results obtained by each model. To do so, we will represent the metrics obtained in the test set for each of the models in a table, taking the mean and the standard deviation resulting from evaluating the model in all the images. Also, one of the metrics we will use is the execution time (E.T.), to take as metric the execution time we will use the time it takes for the model to train a thousand steps. . In addition, at the end we will represent a graph where we will observe the evaluation of the metrics in the training set throughout the 60k steps.

	Model 1	***Model 2***	***Model 3***
Corr. (Std)	-0.4981 (0.0813)	**0.5942 (0.2045)**	0.56084 (0.1811)
T.M. (Std)	0.1370 (0.1945)	**0.5911 (0.1699)**	0.5633 (0.1884)
F.E. (Std)	0.0069 (0.0071)	**0.0331 (0.0280)**	0.0270 (0.0146)
SSIM (Std)	0.1273 (0.0251)	**0.4366 (0.0803)**	**0.4357 (0.0782)**
E.T.	**800.09 sec**	893.21 sec	958.30 sec

We mark in bold the best values comparing the three models, where we can observe that they are always obtained by model 2, except for the execution time, where the first model has a lower execution time, and the SSIM, where both models 2 and 3 obtain very similar results. Once we have seen the metrics in numerical form, let's look at a graph showing the four metrics and their evolution over the steps.

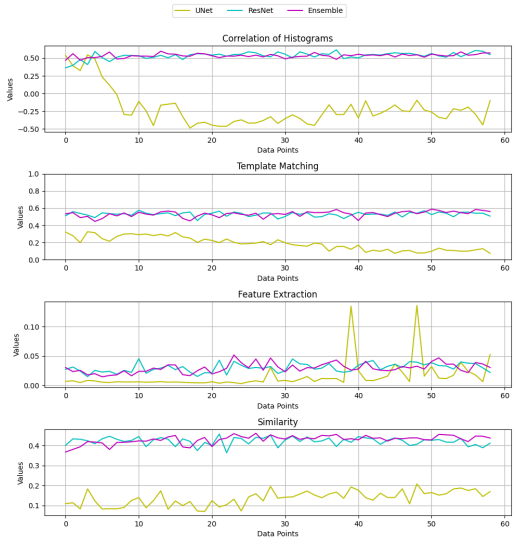

Figure 2. Evolution of the metrics through the training

We can observe the graphs where the green lines represent the first model, the purple lines the second one and the blue lines the third one. We can see how clearly models two and three are better than model one, while the results between them are quite similar. In all cases the results are quite poor. Finally, as a way to evaluate the models in a more direct and not so mathematical way, we will show the image generated by the best model (the second one).

Figure 3. Example of the second model

Looking graphically at the generated image we can see that it is not a great result but it is quite similar to the original image without having so much detail, i.e. the model takes the general shapes well (the shape of the bus or the bollards) but the details like the different parts of the bus it is not able to recognise them.

4. Conclusion

Now that we have seen the results obtained by the models, we can draw certain conclusions.

1) We can observe that ***the best metrics are those obtained by the second model*** in all cases, so we can undoubtedly affirm that the model implemented with a ResNet generator is the one that works best.
2) ***The metrics obtained are not, in any case, a good result***. Furthermore, it shows that the selection of metrics is excessively complicated, as the only one that is relatively consistent is the SSIM since in the first two we obtain a large standard deviation while in the third one, feature extraction, we obtain very low results (perhaps provoked because the images generated are not very bad but they do not have the figures marked and there are no key points).
3) The ***computational cost is very high***, making training very expensive.

In addition, we can comment on certain improvements or additional tests that could be carried out to obtain a better result or simply to find out why we do not obtain it.

1) Train the models with a ***higher number of steps***, observing whether this is a problem of the model or a lack of training. However, it is probably the former, as the visual results of the images over the course of the training do not improve significantly. If the problem is about models, we ***could try new architectures***.
2) One problem can come from using a very small dataset. Because of this we could ***try different data augmentation techniques*** to increase the number of training images and the variability of these images. Another solution could be to ***directly increase the training dataset*** in order to achieve better results, but most probably at an even higher computational cost.

References

[1] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The Cityscapes Dataset for Semantic Urban Scene Understanding*, 2016. [Online]. Available: https://www.cityscapes-dataset.com

[2] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. [Online]. Available: https://www.tensorflow.org/?hl=es-419

[3] OpenCV, *Histogram Comparison*, 2022. [Online]. Available: https://docs.opencv.org/4.x/d8/dc8/tutorial_histogram_comparison.html

[4] OpenCV, *Template Matching*, 2022. [Online]. Available: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

[5] J. Brownlee, *OpenCV SIFT, SURF, and ORB KeyPoints*, Machine Learning Mastery, 2022. [Online]. Available: https://machinelearningmastery.com/opencv_sift_surf_orb_keypoints/

[6] scikit-image, *Structural Similarity Index (SSIM)*, 2022. [Online]. Available: https://scikit-image.org/docs/stable/auto_examples/transform/plot_ssim.html

[7] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, *Image-to-Image Translation with Conditional Adversarial Networks*, 2017. [Online]. Available: https://www.tensorflow.org/tutorials/generative/pix2pix?hl=es-419

[8] P. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. [Online]. Available: https://www.geeksforgeeks.org/u-net-architecture-explained/

[9] K. Gaurav, *Understanding and Visualizing ResNets*, 2018. [Online]. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

[10] M. Rivera, *Deep Convolutional Generative Adversarial Networks (DCGAN)*, [Online]. Available: http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/dcgan/dcgan.html

[11] Y. Huang, Y. Li, C. Pleiss, Z. Liu, J. Hopcroft, and K. Q. Weinberger, *GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism*, 2018. [Online]. Available: https://arxiv.org/abs/1812.04948