AI Generated Zoology

An Application of Image to Image Translation

Sandra Buchen (26317987)
Nigel Yong Sao Young (40089856)
Dan Raileanu (40019882)
Inés Gonzalez Pepe (40095696)
Marc Vicuna (40079109)

A Report Submitted in partial fulfillment of the requirements of COMP 473

Gina Cody School of Engineering and Computer Science
Concordia University
Canada

December 11 2020

AI Generated Zoology

Sandra Buchen Nigel Yong Sao Young Dan Raileanu Inés Gonzalez Pepe
Marc Vicuna

Abstract

The increased interest in Image-to-Image translation, especially the resolution of this open problem through the use of Condition Generative Adversarial Networks (cGANs) has been globally publicized and promoted. Moreover, the generalized loss function use in [4] has led the following question: how generalizable is this algorithm, how far can its limits be pushed? With this project, we aimed to generate realistic images of dogs using the pix2pix (edge2pix on diversified subjects) algorithm and thus, demonstrate its ability to create realistic pictures from very limited (edges) and diversified (dogs) images. We trained on both dog and cat images. Our results are comparable to the original paper which was trained on cat images, and they show that while pix2pix may not be applicable to high-diversity data, it is a solid baseline for many simple tasks, and could potentially be used as a subcomponent of more complex models.

1. Introduction

Image to Image translation is a problem in Computer Vision where a machine is required to translate a given input image domain to a given target image domain. Many problems in Computer Vision, Computer Graphics and other domains can be generalized to Image to Image translation. While those domains have implemented many algorithms to solve many reduced forms of the Image to Image translation problem, up to recently, all the algorithms lacked one thing: a generalized automatic loss function. Loss functions in the context of Image to Image Translation are usually very application-specific, making the implementation of these algorithms less accessible to the general public.

The paper Image-To-Image Translation With Conditional Adversarial Networks from Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros[4] changes this habit by introducing an automatic loss function generation through the combination of both GAN-loss and L1-loss. The Conditional Generative Adversarial Network used has the distinctive characteristic of using as a generator a U-Net-based architecture and as a discriminator a convolutional PatchGAN classifier, both models being recent and state-of-the-art algorithms in their own way. Through the use of these models, the loss function can adapt to seemingly any Image-to-Image Translation task, thus creating a "general-purpose solution to Image to Image Translation". This last statement is the motivation of this project: demonstrating further than ever the ability of this algorithm to adapt to difficult tasks. While other implementations of this project concentrate on applying the algorithms to new Image-to-Image Translation subcategories, we concentrate on applying a difficult but known task to diverse data.

Our goal is to generate realistic images of dogs using the pix2pix (edge2pix on diversified subjects) and thus, demonstrate its ability to create realistic pictures from very limited (edges) and diversified (dogs) images. For these purposes, we applied the pix2pix algorithm on both dogs and cats. We chose to train this algorithm on images on dogs for two major reasons:

- Dogs are more difficult to model. Cats are very similar in shape and appearance (with the popularization of the American Shorthair), and the general tiger-like like stripes many light-color cats have, as striped patterns tend to be very recognizable by such deep neural networks[8]. While similar, dogs do not have a close-to-majority breed, causing photos of dogs to be much less uniform in appearance to both humans and deep neural networks. It tests robustness for this general algorithm through a similar and more difficult example.

- Dogs have many equally represented breeds. For a computer vision perspective, the jump from breeds to species is a small one. Thus, if we were satisfyingly implement a generator for multiple breeds, further research could potentially implement a generator for multiple species, or, in a more limited but still useful way, implement a classifier of edge picture to determine the species represented and use a corresponding trained generator to generate any animal.

We chose to train on cats as a benchmark to our success on dogs, as an intuitive measure of success. Indeed, training on a known problem using our current CPU limitations gives a very accurate representation of the accuracy of our results.

To our knowledge, while some have tested the limits of pix2pix, none have analysed their results in the perspective of generalizability, as we do in this research. We consider this paper as a small addition to the analysis of cGAN and their possible future implementations towards greater generalizability.

2. Experimental Details

In this section, we will discuss the multiple steps we took to reach our goal, including what worked and what did not work and what we learned while doing so.

Before getting into actually training the model, we made sure to understand the paper and to be able to reproduce the existing examples available. The simplest and most light dummy testing was done using the facades dataset. After getting a good grasp of the relationship of the generator and discriminator, we were confident to start training on our data.

2.1. Obtaining the data

Next, we had to obtain the data. At first, we aimed to create a sketch to image translation of multiple animals. So, the idea was, given a sketch, it would turn it into an image having multiple animal features. Our first dataset included 10 animals: dogs, cats, elephants and butterflies, among others. [1]

However, after training, the results did not look good at all. The unbalance between the availability of pictures across multiple species made this task particularly difficult. Moreover, the great difference between the physical attributes of the animals made the generator very random.

We learned that an 'animal generator' might be possible, but it would take a lot of time and high-quality preprocessed data, thus the initial project proposal goal was too ambitious. We settled on a single species: dogs. We also trained our model on cats, as a reasonable benchmark to our experiment. The new datasets used were the 20,000 Stanford Dogs dataset [6] and the 9000 Cats Dataset. [2]

2.2. Images to sketches

The cGAN can translate A to B and B to A. It learns from given examples. Hence, from images of dogs and cats, we needed their sketches. We explored different ways of turning images to sketches and the one which worked the best was a pretrained model, using cGAN itself. After turning the images to sketches, however, we noticed that many of the sketches were bad because of their background, we then decided to remove the background of the images using a Photoshop script before turning them into sketches.

Garbage in, garbage out. To give the model the best data, we subsequently filtered each image one by one, removing images with bad edge detection, images including other things then the dog itself, images that were not dogs or images that an unintuitive angle. This process was repeated for Dogs data and Cats data.

This overall approach of obtaining data, that is background removal, sketch transformation and then filtering has proven its results in the training.

2.3. Training

After obtaining the data, we then move onto training. We first load the data and apply random jittering and mirroring to the training dataset where the image is resized to 286 x 286 and then randomly cropped to 256 x 256 and the image is randomly flipped horizontally i.e left to right. As specified in the paper, we similarly define the generator, discriminator and their corresponding loss functions.

In the Jupyter Notebook (our chosen IDE), we start by iterating over the dataset and so the generator gets the input image and we get a generated output. The discriminator then receives the input image and the generated image as the first input. The second input is the input image and the target image. Next, we calculate the generator and the discriminator loss. Then, we calculate the gradients of loss with respect to both the generator and the discriminator variables(inputs) and apply those to the optimizer.

We then began training, experimenting with different values of batch size and lambda (regularization of L1 loss). Finally, we found that for our training, a batch size of 1 and a lambda value of 100 was best. The training was performed on Google Colab's GPUs and was run for hour-long periods, usually overnight.

The main resources which helped us code the algorithm are GitHub repositories from affinelayer pix2pix-tensorflow implementation [3] and YunYang1994 tensorflow2.0 examples on cGANs [7]. All in all, our algorithm was capable of successfully training a cGAN to turn sketches to realistic images of dogs and cats.

2.4. Methodology

Our model uses conditional generative adversarial network (cGAN) to learn a mapping from an input image to an output image. First, the generator takes a sketch as input and generates some output based on our loss objective. Figure 1 expresses the structure in an intuitive sense the forward loop of the network: using the edge image, the Generator downsamples and upsamples the image to recreate a new image, and, using an edge image and a real or fake image, the Discriminator downsamples the inputs to guess the category the colored image belongs to.

We then feed the generated image together with the original sketch, as well as the ground truth with the sketch in order to train it to distinguish between generated and ground truth images.

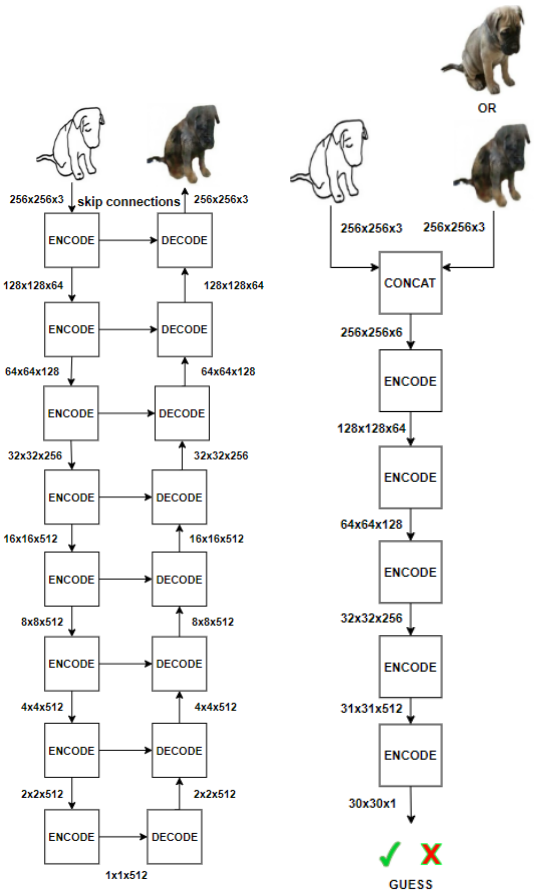

Figure 1. Generator on left and Discriminator on right.

3. Results

3.1. Evaluation (metrics)

Evaluating the quality of the results proves to be a challenging problem. As [4] mentions, traditional metrics such as per-pixel mean-squared error do not assess whether the results are conclusive. As such, a human observer proves to be the best evaluation option. The human observer should be looking for how realistic the results appear and what similarities appear between the input and generated output images.

3.2. Examples

After training our model for 500 train steps on the dog dataset of 2350 images, we obtained the results in Figure 2.

The results showed that the model has difficulty generating dog faces, possibly due to the large variance in appearance between various dog breeds. However, it had some success in properly drawing the rest of the body. It should be noted that only 2 350 images were used, as opposed to the paper's 20 000, which also had the benefit of being on a cats dataset. Cats vary much less in appearance between various breeds, thus we hypothesized that running the model on a larger dataset of cats should provide better results. Figure 3 shows our results after 500 train steps on the cat dataset of 5700 images.

We can definitely see an improvement in the model's ability to draw faces. The decrease in variance and increase in training size did provide better results as predicted.

3.3. Reproducibility

To be able to make good comparisons across different model versions, it's good practice to be able to reproduce the same results every time. Hence, we set random seeds before training and saved TensorFlow training checkpoints after training.

4. Discussion

We compare the results of both models to the paper, as well as both models between themselves. Consider the results in the paper were obtained with 5 to 10 times more

Figure 2. Dog predictions on held-out samples after 500 train steps.

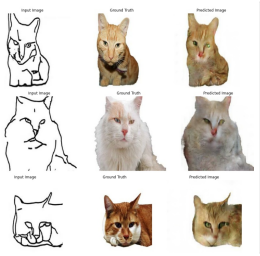

Figure 3. Cat predictions on held-out samples after 500 train steps.

training data then we had, so we expected to have weaker image generation. However, we now consider our results on dogs to be on par with the paper.

4.1. Evaluating Dogs

Looking closely at the dog results, one can notice the blurriness in most images. However, the breed is identifiable even if the dog faces have undesirable features. The predicted images are rich in the dog fur colour palettes and follow a resemblance of fur patterns albeit lacking sharpness in detail of colour separation.

The output is not as satisfying as the results obtained in other implementations of the edge to image translation algorithm likely due to the limited training data at hand. However, we do want to note here that the edge2pix examples, even in the paper, were among the least realistic results. While colorization of black and white images is an 'easy' task for deep conditional GANs, since the edge image is so limited as an input vector, outputs to the edge2pix experiments are not to be held to the same standard as colorization tasks, or most tasks with much more complex inputs.

4.2. Optimizing Training Data

The dataset must be large enough and diverse enough to capture all the desired angles a dog would have, as well as every dog breed. In our case, the data had diverse images of many breeds but lacked enough images of every angle for each breed. If the focus would remain on one breed of dog viewed from an up-close angle instead of a full-body angle, for example, the results would improve. Similarly, if full-body training images were used exclusively, then results for full-body sketches of dogs would improve. The dataset used in this experiment was too diverse to obtain a clear edge to image translation.

Another problem arises during the training image pre-processing. The creation of edges is a separate task that does not always produce desirable sketches of dogs. No dataset exists with these edge images of dogs. The edges are extracted from existing real pictures of dogs. This process adds another error overhead. The generated edge images are not always clear and must be assessed individually by a human observer for optimal pre-processing.

Every picture was analyzed by a human who then decided whether it was an edge image that represented a dog and would be appropriate to train the model. This measure is subjective to the person making the decision and can bias results further. Many of the full-body dog edge images were low-quality and not representative of a clear dog shape and were thus discarded during proper image selection.

The bias is evident when looking at certain questionable edge images that still appear as being a dog, but are predicted with missing facial features such as no eyes or nose visible (See Figure 2, the fourth example). Many full-body image predictions will output the correct fur colour and present a visible dog breed but will have deformed facial features, further showing that not enough full-body images were used in training.

More optimal results could have been obtained with a clear rule on which edge images to focus on keeping for the training model as well as isolating the dog breeds. The less the input variation, the better the predicted images.

4.3. Evaluating Cats

A similar pattern is observed with the results on cats. The dataset contains less variation of cat breeds thus it is expected that the predicted output would have less input diversity to account for. However, the same problems with edges arise. The dataset is of real cat images, so the generated edge images are not always optimal and must be evaluated by a human before being sent into training. The key idea is that most available images of cats happen to be from the domestic cat breed, the tabby cat. These cats look quite

similar and have less variation in facial and body features. In this dataset, it is no longer possible to categorize a cat breed since we are dealing with less breed diversity.

The dataset of cats concentrates much more on up close images instead of full-body images. This is visible in the model in two ways: the cat faces tend to be more realistic than the dog faces, but since for humans faces are so much more important to recognize than the full-body, intuitively, our results for dogs look better, as we tend to not examine the body, but we do examine the faces. This issue is addressed in [5] Facial features in cats are more defined than in dogs, but the eyes tend to be much harder to generate for the psychological reason mentioned earlier.

In comparison to dogs, cat fur patterns are more easy grasped by the model. Cats have clearer patterns than dog predictions. We consider this result to be a result of the lesser diversity in the cat dataset. Overall, for both models, the level of fur detail achieved in the paper is not reached. We expect this kind of optimization to happen purely using more data and higher quality data.

Overall, our model on cats performs generally worse than our model on dogs. The predictions are noticeably worse, to our surprise. While we thought the greater diversity in breed would make dogs harder to generate, their general lack of fur patterns makes it easier for simple images of dogs to be more photorealistic than cats. The lack of correct patterns in cat predictions disadvantages our second.

In short, the results have the appearance of what they are trying to predict. The output resembles that of a dog or a cat in most cases. The task of generating realistic dog images is achieved for full-body images. Up close images are not as realistic, but we consider these results, even for up close, reasonably good. However, the details within that shape need to be sharper in order to conclude that an image is not computer-generated. In the case of our cats model, it is clear that more training needed to be done on larger data in order to at least model cat fur patterns more efficiently.

5. Conclusions

To summarize, the goal of this project was to reimplement the generalized loss function proposed in this paper in order to generate images from edge image on an intuitively more difficult dataset. To this end, generative adversarial networks were used to create a tool that would test the image to image translation properties of the proposed algorithm. More specifically, the ideal neural network built should take any sketch of an animal (referred to as an edge image) and turn it into a realistic image of a given subject.

Once the project was underway, we realized that in order to achieve our objective, a narrower field must be established. Instead of training our model on any type of animal, dogs became the primary focus as previous training had demonstrated that too many different species and a lack of sufficiently large training datasets on our end led to a very confused model and bad results. When focusing solely on dog datasets, the results improved, but it was observed that while the animal was able to be distinguished by a human observer, the facial features were still unfocused and/or missing and there was an overall lack of detail and image sharpness.

To further simplify our objective for the model, we decided to train it on cats, which have fewer breeds and therefore less variability and more physical similarities among them, which should make turning an edge image into a realistic image an easier task. However, the same problems that occurred with the dogs occurred with the cats as facial features were extremely difficult for the model to construct. After analysis, it can be concluded that this is due to the fact that the datasets our model has been trained on simply is not sufficient for it to properly learn how to construct a fully realistic image of an animal.

Looking back, the objective of generating realistic images of dogs that are identifiable as such to the human observer was successfully achieved. Moreover, the model was expanded upon and can also generate realistic images of cats at the same level of quality as the dog images. Therefore, it can be concluded that the model based on the Image to Image translation paper was properly implemented and its generalization capabilities have been tested. We do not think in further research, this model as is could potentially implement a generator for multiple species still the train in diversified data remains slow and seems to be even slow for a constant level of realism and greater diversity. However, we do think further research to create bigger models on top of this existing one, such as implementing a classifier of edge picture to determine the species represented and then, use a corresponding cGAN trained generator to generate the predicted animal, resulting in a generator for many species.

In terms of precision, the animal images generated are not perfect as details are lacking and facial features tend to be blurry or missing regardless of the animal tested. As mentioned, this is due to the limited quantity of data available, much of which had to be generated manually due to lack of resources. If this project were to be reattempted, a different approach could be to either improve upon the edge image data generation methods or to find/crowd-source a better dataset online. Future work could be conducted to improve the model's facial reconstruction capabilities and with sufficiently large amounts of data of better quality, the model could potentially be trained for greater diversity in images. While there is potential to grow, there is a solid base to build upon.

References

[1] Corrado Alessio. Animals-10, Dec 2019. https://www.kaggle.com/alessiocorrado99/animals10. 2

[2] Chris Crawford. Cat dataset, Feb 2018. https://www.kaggle.com/crawford/cat-dataset. 2

[3] Christopher Hesse. affinelayer/pix2pix-tensorflow, 2017. https://github.com/affinelayer/pix2pix-tensorflow. 2

[4] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017. 1, 3, 6

[5] Santosh Kumar and Sanjay Kumar Singh. Biometric recognition for pet animal. *Journal of Software Engineering and Applications*, 2014. Publisher: Scientific Research Publishing. 5

[6] Jessica Li. Stanford dogs dataset, Nov 2019. https://www.kaggle.com/jessicali9530/stanford-dogs-dataset. 2

[7] YunYang1994. Yunyang1994/tensorflow2.0-examples, 2019. https://github.com/YunYang1994/TensorFlow2.0-Examples. 2

[8] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1

Appendix

To access the Github of this project, go to AI-Generated-Zoology.

Code implementation

This section contains useful code that may help in the understanding of our implementation.

```python
# Downsampling, implementation of the encoder.
def downsample(filters, size, apply_batchnorm=True):

    initializer = tf.random_normal_initializer(0., 0.02)
    result = tf.keras.Sequential()
    # 1st layer, Conv
    result.add( tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
                                       kernel_initializer=initializer, use_bias=False))
    # 2nd layer, Batchnorm
    if apply_batchnorm:
        result.add(tf.keras.layers.BatchNormalization())
    # 3rd layer, Leaky ReLU
    result.add(tf.keras.layers.LeakyReLU())

    return result
```

```python
# Upsampling, implementation of the decoder.
def upsample(filters, size, apply_dropout=False):

    initializer = tf.random_normal_initializer(0., 0.02)
    result = tf.keras.Sequential()
    # 1st layer, Conv
    result.add(tf.keras.layers.Conv2DTranspose(filters, size, strides=2, padding='same',
                                               kernel_initializer=initializer, use_bias=False))
    # 2nd layer, Batchnorm
    result.add(tf.keras.layers.BatchNormalization())
    # 3rd layer, Dropout (Randomization)
    if apply_dropout:
        result.add(tf.keras.layers.Dropout(0.5))
    # 4th layer, regular ReLU
    result.add(tf.keras.layers.ReLU())
    return result
```

Figure 4. Multilayered step of down and up-sampling.

Figure 4 shows the implementation of the downsampling, reducing image dimensionality. It is used extensively in both Generator and Discriminator. The upsampling step, or data augmentation step, is only necessary in the Generator to recreate an image using the latent features, as a decoder would do. This is an essential part of the model implemented as it allow the data to be simplified to simpler latent features, taking less time to train.

```python
# Defining the discriminator loss
def discriminator_loss(disc_real_output, disc_generated_output):

    real_loss = loss_object(tf.ones_like(disc_real_output), disc_real_output)
    generated_loss = loss_object(tf.zeros_like(disc_generated_output), disc_generated_output)
    total_disc_loss = real_loss + generated_loss

    return total_disc_loss
```

```python
# Defining the generator loss
def generator_loss(disc_generated_output, gen_output, target):
    LAMBDA = 100
    gan_loss = loss_object(tf.ones_like(disc_generated_output), disc_generated_output)

    # mean absolute error
    l1_loss = tf.reduce_mean(tf.abs(target - gen_output))
    total_gen_loss = gan_loss + (LAMBDA * l1_loss)

    return total_gen_loss
```

Figure 5. Implementation of the generalized loss in this implementation.

For backpropagation, Figure 5 addresses the central implementation of the paper [4], that is, a new original and efficient loss function for the generator. Notice the addition of the L1-distance and the GAN loss.

```python
# Defining the generator
def Generator():
    # Downsampling stack
    down_stack = [
        downsample(64, 4, apply_batchnorm=False), # (bs, 128, 128, 64)
        downsample(128, 4), # (bs, 64, 64, 128)
        downsample(256, 4), # (bs, 32, 32, 256)
        downsample(512, 4), # (bs, 16, 16, 512)
        downsample(512, 4), # (bs, 8, 8, 512)
        downsample(512, 4), # (bs, 4, 4, 512)
        downsample(512, 4), # (bs, 2, 2, 512)
        downsample(512, 4), # (bs, 1, 1, 512)
    ]
    # Upsampling stack
    up_stack = [
        upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
        upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
        upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
        upsample(512, 4), # (bs, 16, 16, 1024)
        upsample(256, 4), # (bs, 32, 32, 512)
        upsample(128, 4), # (bs, 64, 64, 256)
        upsample(64, 4), # (bs, 128, 128, 128)
    ]
    # Initialization
    initializer = tf.random_normal_initializer(0., 0.02)
    last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS, 4,
                                           strides=2,
                                           padding='same',
                                           kernel_initializer=initializer,
                                           activation='tanh') # (bs, 256, 256, 3)
    concat = tf.keras.layers.Concatenate()

    inputs = tf.keras.layers.Input(shape=[None,None,3])
    x = inputs
    # Downsampling through the model
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)

    skips = reversed(skips[:-1])
    # Upsampling and establishing the skip connections
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = concat([x, skip])

    x = last(x)

    return tf.keras.Model(inputs=inputs, outputs=x)
```

Figure 6. Implementation of the generator.

```python
# Defining the discriminator
def Discriminator():

    # Initialization
    initializer = tf.random_normal_initializer(0., 0.02)
    inp = tf.keras.layers.Input(shape=[None, None, 3], name='input_image')
    tar = tf.keras.layers.Input(shape=[None, None, 3], name='target_image')
    x = tf.keras.layers.concatenate([inp, tar]) # (bs, 256, 256, channels*2)

    # Downsampling blocks instantiation
    down1 = downsample(64, 4, False)(x) # (bs, 128, 128, 64)
    down2 = downsample(128, 4)(down1) # (bs, 64, 64, 128)
    down3 = downsample(256, 4)(down2) # (bs, 32, 32, 256)

    zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3) # (bs, 34, 34, 256)
    # 1st layer, Conv
    conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                  kernel_initializer=initializer,
                                  use_bias=False)(zero_pad1) # (bs, 31, 31, 512)
    # 2nd layer, Batchnorm
    batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
    # 3rd layer, Leaky ReLU
    leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
    zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu) # (bs, 33, 33, 512)
    # 4th layer, Conv
    last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                  kernel_initializer=initializer)(zero_pad2) # (bs, 30, 30, 1)

    return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

Figure 7. Implementation of the discriminator.

Figure 6 shows the implementation of the Generator, which is based on both downsampling and upsampling. Moreover, the second part implements the U-Net structure of the GAN, which is another characteristic of this generator.

Figure 7 shows the implementation of the Discriminator, which is based on upsampling and multilayered decision network called in the literature a 'PatchGAN'.

Using those essential functions and preprocessed data, we are able to train and predict dog images. The figures in the Results section are some outputs of our train models. Running our test set on our pretrained model should output similar results to the ones shown. This code methodology is common in the image to image translation tasks. We used it to make sure we would get similar results to previous research. While other implementations are possible, we wanted to give an intuitive approach to this task, so that future research might be done in the same fashion for more complex models.