Colour This

Pérez Mariana, ISC A01206747, Tecnológico de Monterrey Campus Querétaro

***Abstract*— This document presents the implementation, explanation and analysis in the use of an Encoder-Decoder Neural Network and its performance in colouring black and white images and the use of parallelization techniques to improve the waiting times during training.**

I. INTRODUCTION

Colorizing images that were taken hundreds of years ago allows us to see distant history in a new light. Colorization of old photographs has been done since the 1800's when images were coloured by hand or through a process called photochrom, which added anywhere form 6 to 15 layers of colour to a photo negative, the problem with these techniques was that the resulting images didn't look as realistic as one could expect.[1] Nowadays, colorization of photographs is done by hand using different software such as photoshop or illustrator, along with a vast number of online resources have led artists to reconstruct images with more accuracy. A picture can take up to one month to colorize and in order to do it requires an extensive research of the time period to find the exact colours that would recreate a moment in time. The process of colouring photographs can take a lot of time because for instance human skin can take up to 20 layers of pinks, yellows, greens, reds and blues.

II. STATE OF THE ART

The Encoder-Decoder Network is a neural network design pattern, that is partitioned in two parts, the encoder and the decoder. This architecture is the standard neural machine translation method and is the core technology inside Google's translate service.[2] The encoder encodes the inputs into state, which often contains several tensors, this means that the encoder takes the raw input and then outputs a feature tensor that holds the information of the given input. Then this state is passed into the decoder whose job is to generate the wanted outputs. In machine translation, the encoder transforms a source sentence, e.g. "Hello world" into state, e.g., a vector, that captures its semantic information then the decoder uses this state to generate a translated target sentence, e.g., "Bonjour le monde".[3]. A classic Encoder-Decoder mechanism is the one where an image is fed into an Encoder network which encodes the image forming a representation and passes it to the decoder which tries to reconstruct the original input image. This paper uses a slight modification of this for image reconstruction, the Encoder and Decoder is implemented using Convolutional layers to map images into a function in order to do feature extraction.

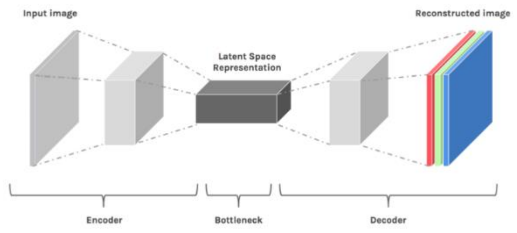


Figure 1. Graphic representation of the Encoder-Decoder

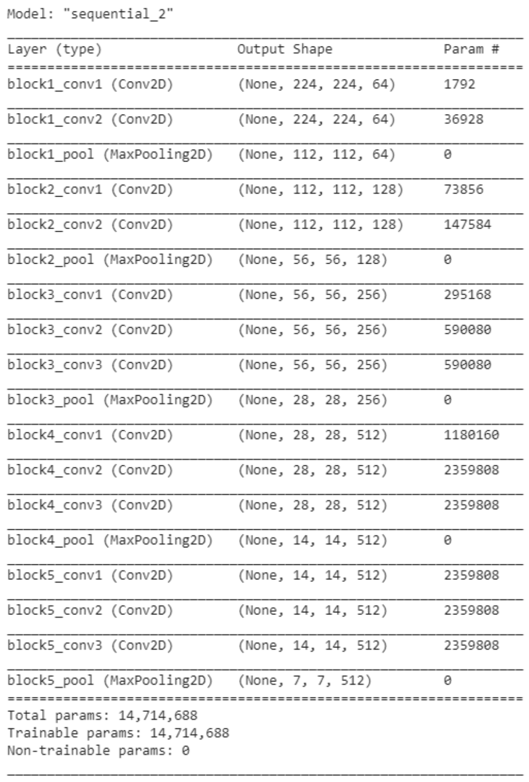
III. DATA SET

The dataset comes from the Unsplash dataset created by Emil Wallner[4], that contains a database of about 9000 RGB images of 256x256 used for training and about 500 black and white images of 256x256 that are not in the training sample.

Figure 2. Example of the type of images that are present in the dataset.

IV. MODEL PROPOSAL

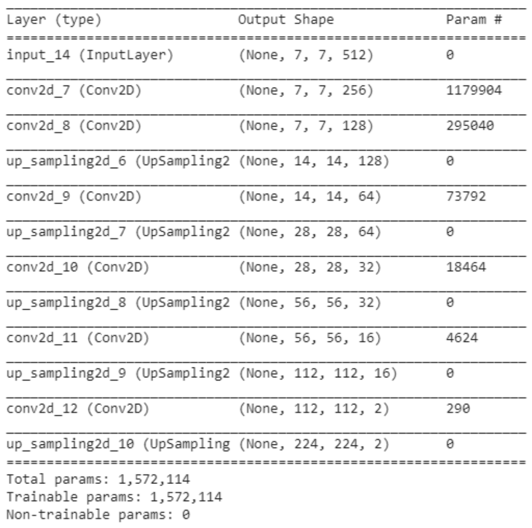
The Encoder-Decoder model proposed to colorize images uses VGG16 as an encoder to do feature extraction with its predetermined weights



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Figure 3. Summary of VGG16 (Encoder) without pooling layers.

The Decoder as the Encoder is composed by Conv2D and Up Sampling layers.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        (None, 7, 7, 512)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 7, 256)         1179904
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 7, 7, 128)         295040
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 64)        73792
_________________________________________________________________
up_sampling2d_7 (UpSampling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 32)        18464
_________________________________________________________________
up_sampling2d_8 (UpSampling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 56, 56, 16)        4624
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 112, 112, 2)       290
_________________________________________________________________
up_sampling2d_10 (UpSampling (None, 224, 224, 2)       0
=================================================================
Total params: 1,572,114
Trainable params: 1,572,114
Non-trainable params: 0
```

Figure 4. Summary of Decoder.

In order to predict the colour of the images the input images need to be pre-processed before entering the model. The pre-processing consists first on applying some transformations to the images in order to do data augmentation.

After applying the transformations, the input images are changed intro LAB images, which are an image format that separates the lightning from the RGB images.

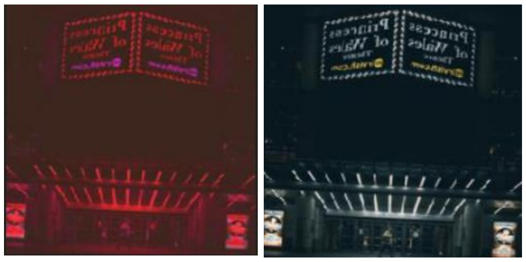


Figure 6. Example of a LAB image (left) with its RGB counterpart (right)

After transforming the image the layers of the LAB photograph are separated into two arrays, an X array containing the L layer (luminosity) and a Y array containing the A and B layer (red/green and yellow/blue)

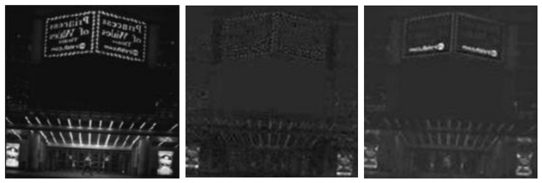
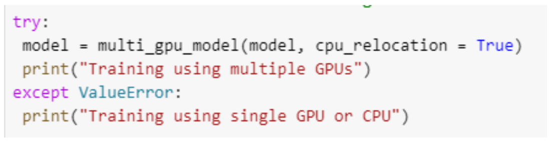


Figure 7. Example of L layer (left) and A layer (middle) and B layer (right)

Then the processed image is passed though the encoder, which in this case is the VGG16 network and it outputs an image of size 7x7 with 512 filters. This image is then used as the input for the Decoder network. The model is compiled using the mean square error as a loss function and an Adam optimizer.

Another objective of this project was to test the viability of parallelizing the training progress of the network. In order to do this, there are approaches than can be implemented. The first one is to replicate the model into all the available GPUs obtaining a single machine multi-GPU parallelism. This approach divides the model inputs into multiple sub-batches and then a copy of the model is applied on each sub-patch, therefore every copy of the model is executed on a dedicated GPU and at the end the results are concatenated into a big batch in the CPU.



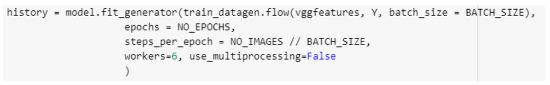
```
try:
 model = multi_gpu_model(model, cpu_relocation = True)
 print("Training using multiple GPUs")
except ValueError:
 print("Training using single GPU or CPU")
```

Figure 8. Code of the duplicate of the model per GPU

This method permits that if we have a batch size of n images and want to use for instance two GPU's to train the network, then the batch size will be divided into the number of GPU's and create x number of sub-batches where x is the number of GPU's. Then each GPU process each sub-batch on its corresponding GPU and at the end it will return a full batch of 64 processes samples.[5]

The other method is to make use of the use_multiprocessing, and workers that are implemented in the fit_generator function. The workers parameter which takes as input an integer will spawn threads or processes when ingesting data batches. This if not done properly, meaning that the way of fetching the data is not thread safe could lead to some problems such as data duplication, this is only done when the use_multiprocessing is equal to false.

When the parameter use_multiprocessing is set to true, the parallelism changes a bit and now the training uses two or more central processing units also called CPUs and allocate the corresponding number of workers (processes) to each CPU.



```
history = model.fit_generator(train_datagen.flow(vggfeatures, Y, batch_size = BATCH_SIZE),
                    epochs = NO_EPOCHS,
                    steps_per_epoch = NO_IMAGES // BATCH_SIZE,
                    workers=6, use_multiprocessing=False
                    )
```

Figure 9. Code of multiprocessing and multi-threading

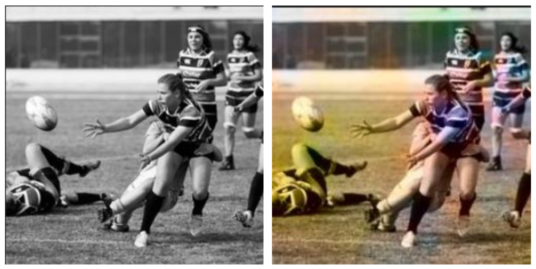
V. TEST AND VALIDATION

After running the training for 1000 epochs, using 7000 images to train to prevent the RAM depleted, with a batch size of 64, and 109 steps per epoch, had the following performance.

ADD GRAPH

ANALYSIS OF GRAPH

With the obtained weights after the training of the model, 500 black and white images where predicted. Some of the resulting images did not end up with colour, but there were some photographs that ended up with a nice result. The images shown below are some samples of the generated images alongside the original black and white photograph. The complete list of generated images can be found in the corresponding link. [6]



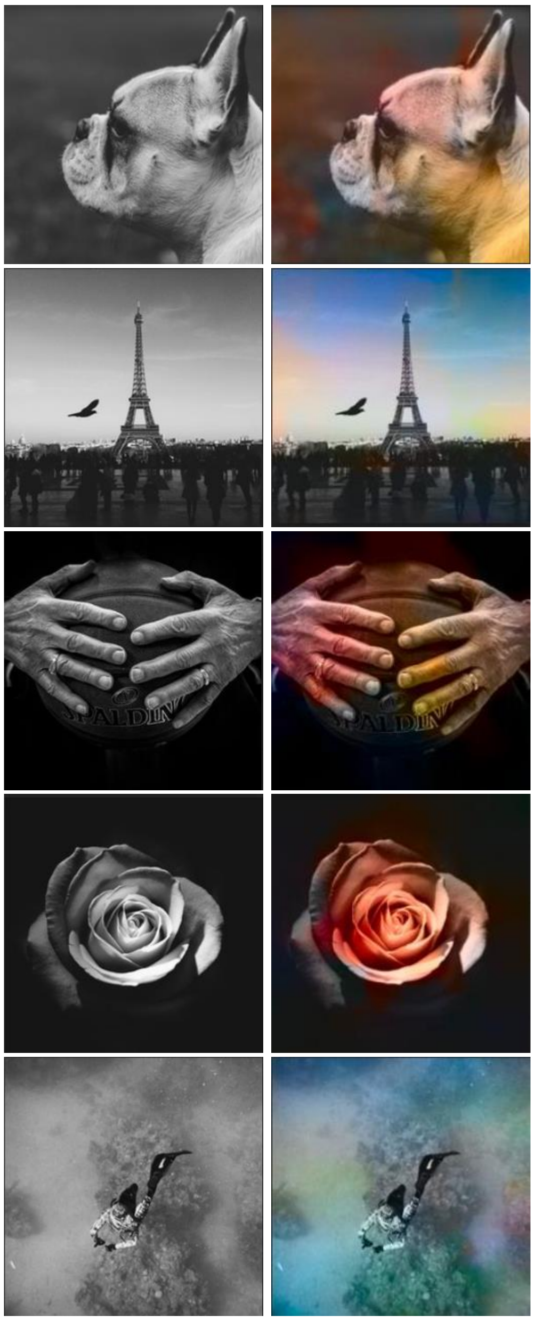


Figure 10. Comparison between black and white images and their corresponding colorization.

Regarding the parallelization of the training the project was tested with multi-threading and multi GPU parallelization.

ADD TABLE OF COMPARISON

ANALYSIS COMPARISON

Not always best results, show graph, parallelism has a runtime cost, faster to access sequentially than the parallel ingestion.

VI. CONCLUSIONS

Image colorization can lead to see history in a new perspective, it helps to stop looking at it as a linear timeline but rather as a tapestry of all the moments that were lived by all the people

VII. REFERENCES

[1]. (2017)Lowndes C. How obsessive artists colorize old photos. Retrieved from https://www.youtube.com/watch?v=vubuBrcAwtY&t=308s

[2] Brownlee J. (2018). Encoder Decoder Recurrent Neural Network Models for Neural Machine Translation. Retrieved from https://machinelearningmastery.com/encoder-decoder-recurrent-neural-network-models-neural-machine-translation/

[3] Hui J. (2018). GAN – DCGAN (Deep Convolutional Generative Adversarial Networks). Retrieved from https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

[4] Wallner E. (2017.). Colornet. Retrieved from https://www.floydhub.com/emilwallner/datasets/colornet

[5] Keras Documentation. (n.d.). AKeras Documentation. Retrieved from https://keras.io/utils/#multi_gpu_model

[6] Pérez M. (2019). Complete directory of generated images. Retrieved from https://drive.google.com/open?id=1--E-oeE9fOOYOe6410xxu6-ORIyaOR1J

The implemented code used as a base the following codes:

DataSnaek. (2018). Simple Deep Convolutional Generative Adversarial Network built in Keras. Retrieved from https://github.com/DataSnaek/DCGAN-Keras/blob/master/DCGAN.py

Shibuya N. (2017). Face Generation with DCGAN. Retrieved from https://github.com/naokishibuya/deep-learning/blob/master/python/dcgan_celeba.ipynb