Image Compression with Deep Learning Colourization

***Abstract*—Presented in this paper will be a compression algorithm for images known as ICDLC (Image Compression with Deep Learning Colourization). This algorithm takes an original image and compresses it by taking the greyscale of it along with some side information. This is all that is needed to encode the image and the decoders goal is to take the greyscale image and side information and to attempt to recreate the original image. This is done on the decoding side by taking the greyscale image and side information to be fed into a convolutional neural network. The convolutional neural network is based on the Xception architecture and mimics it precisely in what is called the entry and middle flows. However, it is adjusted in the exit flow to instead be an inverse of the entry flow. Passing the data through the convolutional neural network returns the predicted colours of the original image (CbCr values). Combining the predicted colours with the greyscale image produces the final colour image. The performance of this algorithm is currently not far away from standard JPG compression but offers some extra benefits and possible places of future improvements.**

I. Introduction

There are multiple examples of image compression algorithms and image colourization networks, but they have yet to be combined. The idea of combining the two together will create a colourization network which can deduce a compressed image back into its original after adequate training.

A. Colourization Problem

Colourization takes a greyscale image and attempts to add colour back to it. One of the main problems that is faced by ICDLC compared to other colourization networks is that there is a measure of success that is more precise. With a regular colourization network it is not important as to what the original image was, but more so that the network produces a realistic output. However, ICDLC requires that the output produced is as close as to the original image as possible, otherwise it would not be a good compression algorithm. This is shown in Figure 1 where the original image on the right is in blue, but the network takes the greyscale image and produces one that has red in place of the blue. The colourization is realistic if looking at just the red image but comparing it to the original it would be completely inaccurate.

Figure 1: From left to right we have the original image, greyscale image, and deep learning colourized image [1]

In order to combat this issue, colourization alone will not suffice and therefore additional information will need to be given, known as side information. This side information will be key in helping the colourization network deduce the issue previously mentioned.

B. Side Information

The side information used is a resizing of the original image by taking 32x32 blocks and using a single pixel to represent the entire block. Therefore, an image of size 256x256 would be resized to an 8x8 image. This 8x8 images CbCr values would be used as the side information. The greyscale image and the side information would be the only requirement for encoding the image, leading to a roughly 2.999:1 compression ratio.

The decoding would require taking the greyscale image, finding the edges of the image, and reading the side information and feeding all that information into the ICDLC convolutional neural network. The neural network would finally produce an output that is the predicted CbCr values of the original image and combined with the greyscale image to create a new image. This final image would be the decoded image.

II. Background

A. YCbCr

First and foremost, it is required to understand what is meant by YCbCr. YCbCr stands for the luminance (Y) and the Cb and Cr are the two chroma components which defines the colour. When greyscale is mentioned, it is the Y or luminance that is taken from the image and the CbCr values are excluded. That is why the ICDLC network must predict the CbCr values and combine it together with the Y value, recreating a standard YCbCr image.

B. Colourization with Fusion Layer

The current state of the art colourization is based on [2] where the innovative strategy used is a fusion network. Their network is basically made up of three components: encoder, fusion, and decoder as can be seen in Figure 2. The encoder and decoder as just simple 2D convolutions with as well as pooling in the encoder and up sampling in the decoder. The encoder and decoder can be used separately, but the fusion helps add extra side information without having to add extra input information. This is because the fusion network is a classification network.

While the fusion layer can be added to ICDLC in the future, the fusion network is currently ignored. This is because currently ICDLC focusses on taking in inputs with people's faces/portraits. This is done to simplify the network since the removal of the fusion network greatly decreases training time.

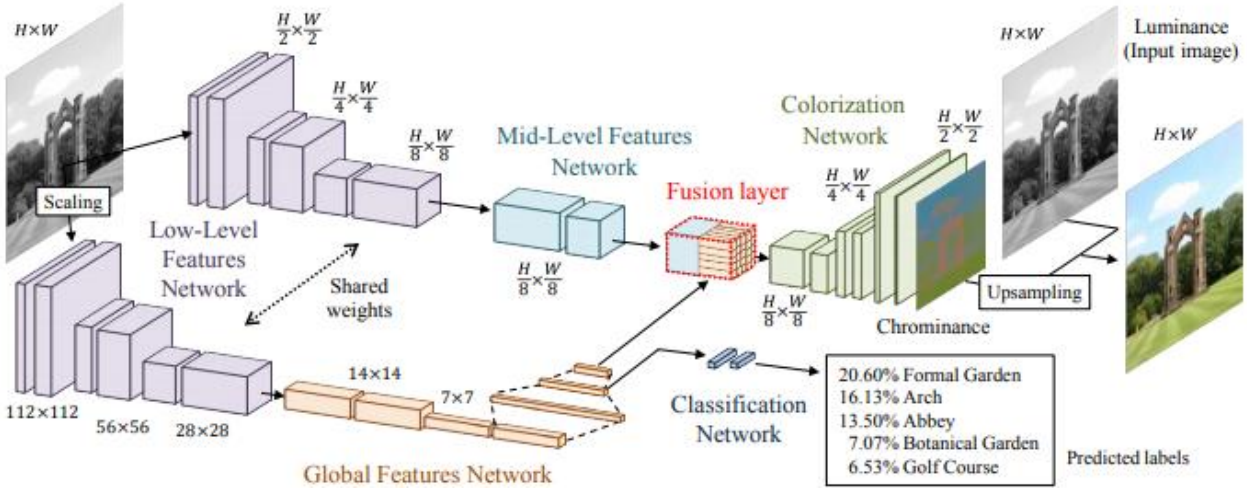

Figure 2: Overview of image colourization network with fusion layer

C. *Xception*

The previous fusion convolution neural network had a simple convolutional neural network but the one used in ICDLC is much deeper as it is based off Xception [3]. Xception is a 36-convolution layer network which consists of an entry flow, middle flow, and exit flow. The network is used primarily for classification but also has been known for its use in image super-resolution, the act of trying to improve the resolution of an image. This flow can be seen in detail in Figure 3.

D. *Current Colourization*

While we have covered the basics needed for this paper, it would also be important to include how current colourization performs. As explain in the introduction, most colourization networks aren't concerned with being the same as the original but to be realistic. This is because for many problems that colourization faces it does not have a frame of reference to compare to. Many times, colourization is used for black and white images back from the days before colour images was invented. As such it is very hard to impossible to know exactly what colours some parts of an image may be which is why being realistic may be adequate. Unfortunately for ICDLC this is not the case and there is a standard that the final image must be compared to.

III. Details

A. *Dataset*

In order to simplify the work required and to produce a meaningful output it was decided to simplify the network to use just images of peoples faces (with varying backgrounds). This was done, as previously mentioned, to simplify the network and as such allowed ICDLC to remove the fusion layer of the neural network since classifying what is shown in the image is not required anymore (since they are all peoples faces). In order to do this the LFW dataset was used [4] which has a total of over 13,000 images with faces.

The images are all 250x250 and as such the ICDLC was trained on this size network. Because of this all inputs are required to be of this size. However, this all handled by ICDLC currently as it resizes the image to 250x250. Using any image size is further explained in the future improvements section of the paper.

Unfortunately, using 250x250 does not work well with convolution networks as pooling becomes a problem since the dimensions are no longer integers. In order to fix this problem, the images are pasted onto a black 256x256 canvas.

B. *Greyscale image*

As the goal of ICDLC is to use colourization with some side information it is first needed to supply the greyscale image. This is simple to extract from either a PNG or JPG format image, but for most other images it can also be extracted. Since getting the greyscale image is trivial, the next important aspect is to get the best side information. Initially the fusion network was used as a preliminary test for side information as it was very fast to train.

C. *Side Information*

The first proposal regarding side information was to have every greyscale pixel appended with a 2-bit value which determined if the pixel was red dominant, green dominant, or blue dominant. This would in theory fix the issue seen before (in Figure 1), but with 2 bits and 3 values an extra possible value was being wasted.

The next proposal for side information was a quick fix that would instead use the 2-bit value to store the CbCr values. The first bit would correspond to the Cb value and the second bit the Cr value. The values would normally range from 0 to 255, but in order to compress that into 1 bit each, anything less than 128 was considered 0 and otherwise it was considered a 1. Ultimately this took an image from 24 bits per pixel and reduced it to 10 bits per pixel (8 bits for greyscale plus appended 2 bits) which equaled to a compression ratio of 2.4:1 from what is normally 3:1 without any side information. After some

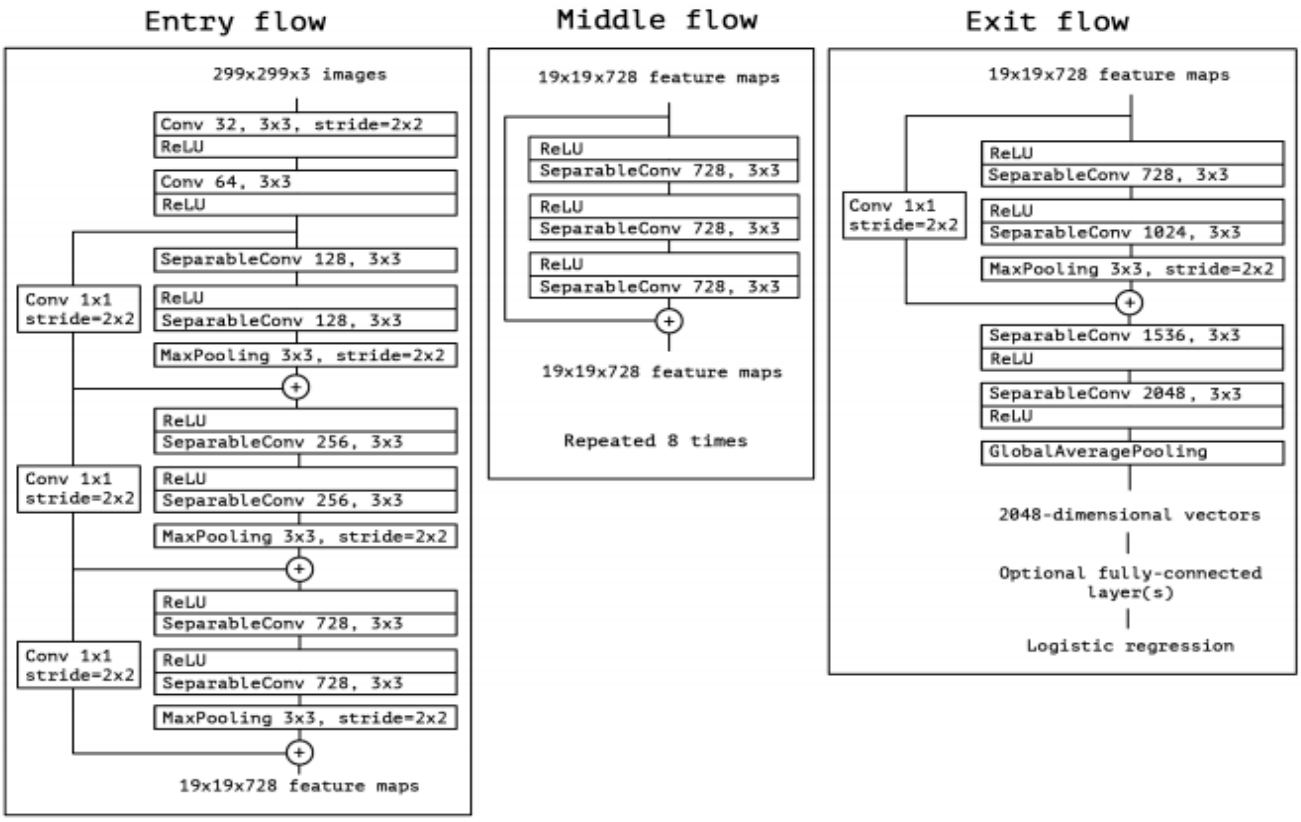

Figure 3: Xception Network Overview

testing, the side information using the 2 additional bits per pixel, it was found not to be as accurate as original thought.

The final attempt at side information was instead compressing each 32x32 block of an image into a single pixel and extracting the CbCr values from just that single pixel. For example, a 256x256 image would be resized to 8x8 in which only the Cb and Cr values were extracted from that image, since the Y (greyscale) value was already available in the greyscale image. This led to a compression ratio of approximately 2.999:1 as shown in Equation 1 below. This compression ratio is much better than the previous 2.4:1 and ended up producing better results. The side information is then stored in a file without any further compression as the computation time cost of doing so would greatly outweigh any minimal savings in bytes.

It is important to note that the compression ratios specified are assuming that no additional encoding, quantization, or transforms are used. Therefore, the final compression will be slightly different as using PNG uses some lossless compression to minimize the size of the file.

$$3 - \left(\frac{1}{1024}\right) : 1$$
$$2.9990234375 : 1$$

Equation 1: Compression ratio for ICDLC

D. Encoder

The encoding of the image then ends up being very simple to implement as it requires just extracting the greyscale values of the image and for the side information just resizing each 32x32 block of the image into a single pixel (1x1 block) and saving the CbCr values into a text file. This is one of the advantages of using ICDLC over other standards as the encoding algorithm is very simple in terms of understanding and time complexity.

E. Decoder

The decoding side of the ICDLC algorithm is much more involved. First it needs to read the greyscale image and perform edge detection on that image. The edge detection algorithm used is important as a better algorithm will produce better results. The exact method used will be described in detail in the results and future improvements section, but simply a better edge detection algorithm will generally give better results. Next the decoder must take the side information and decode it from a text file into an image. Then it will have to resize the image by multiplying its height and width by 32. This will give an image with the same size as the greyscale image but each 32x32 block all be the same pixel colour. It was also beneficial to add a gaussian blur of radius size 1 to this expanded side information image. Now the expanded side information image, the original greyscale image, and edge detection image are all fed into the ICDLC network.

Since the encoding is very simple, as expected the decoding is anything but. The decoding of the image requires some pre-processing in order to then use the ICDLC neural network. First the greyscale image and the side information must be gathered obviously. The side information is converted back into an image and then resized into the original image size. The greyscale image has an edge detection algorithm ran on it and creates another image that detects edges. Now with all three of these in place: the greyscale image, the edge detection image, and the CbCr of the side information is passed into the ICDLC

network. The network then simply outputs the Cb and Cr values that it has predicted and those are compared with they greyscale image (Y) to create a YCbCr image. This is the final predicted image.

F. ICDLC Convolutional Neural Network

The network itself is based on the Xception network, as previously mentioned. The ICDLC network mimics a similar entry and middle flow from the Xception network, but the exit flow is changed. The only difference in the entry flow is the image being fed in is not a 299x299x3 image but instead 256x256x4. First 256x256x1 is the greyscale image, secondly is the resized CbCr side information of size 256x256x2, and finally is the edge detection information of size 256x256x4, thus adding up to 256x256x4.

The reason for using a custom exit flow is that the Xception network creates a fully connected layer which is just a single vector whereas the ICDLC requires a 256x256x2 (Cb and Cr values) matrix to be output. Thus, the exit flow was changed to be an inverse of the entry flow which now makes the ICDLC network have a total 41 convolution layers compared to the regular Xception network of 36. By inverting the entry flow, all of the pooling now becomes upsampling, but all of the strides and convolution sizes remain the same. The only addition is that at the very end an additional upsampling is used and a convolution of 2 layers with no stride length and activation function of tanh. This allows the network to output a 256x256x2 image.

Once passed through the network, the output is simply combined with the greyscale image and that is the final predicted output. The network is saved as a model and takes approximately 191 MB of space. This space is not considered later as part of the size of the algorithm as it is a constant that never changes. It can be thought of as simply being a part of the code size of the algorithm.

G. Quick Walkthrough

The following will be a quick walkthrough of how the entire ICDLC algorithm works from beginning to end. We can see in Figure 4 the image on the left is the original image of size 250x250. The convolutional neural network requires the input to be of size 256x256, so in the image to the right of the original is the extended image of size 256x256. The extra space is now instead taken up by blackness.

Figure 4: Original image and extended image (left to right)

Next the encoder gets the greyscale image of the extended image as well. It also resizes the image to be from 256x256 to be 8x8 (each 32x32 block is 1 pixel). This 8x8 block is saved in a text file as only the Cb and Cr values. These can be seen visually in Figure 4Figure 5.

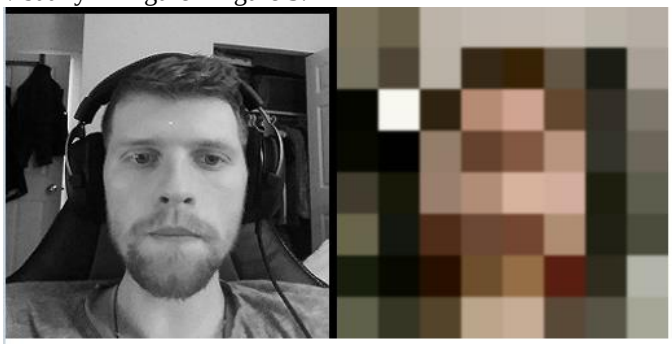

Figure 5: Greyscale and side information images (left to right)

After this the encoding is done and the two files are sent over. The decoder takes both images in Figure 5 and uses edge detection on the greyscale image and resizes the side information back into 256x256, but each 32x32 block is still a single colour. The outcome of the edge detection can be seen in Figure 6.

Figure 6: Edge detection image

Finally, the greyscale image, edge detected image, and side information is sent to the convolutional neural network and it produces the CbCr values that when combined with the greyscale image produce the image seen in Figure 7. As a comparison against a real colourization implementation we can see in Figure 8, the outcome of the Colorful Image Colorization [1]. The image was done by specific inserting the original image into the site https://demos.algorithmia.com/colorize-photos/.

Figure 7: ICDLC Predicted Image

Figure 8: Colourization done by Algorithmia

IV. Results

A. Visual Results

Some of the results of ICDLC can be seen in Figure 9 as well as the result from the previous section in Figure 7. The images in Figure 2 come from the LFW dataset but were not used during training or validation when training the ICDLC network. This is important because this means that these images have never been seen by the network and have been predicted without any prior knowledge. Each image is paired together, where the image on top is the original image and the image on the bottom is the predicted image. The LFW dataset has 250x250 images but the ICDLC network takes in and outputs 256x256 images which is why the images all have a little bit of a black border as they have been expanded to be 256x256.

At first glance many of the images look very similar, if not the same, but under closer inspection some differences can be seen. One of the negative trends of the ICDLC network is that it seems to smooth out redness in images. This can be seen easiest in Figure 9 with the third image to the left of the bottom row in which the woman is speaking into the microphone. It can be seen in her hair how originally it is brown with a small amount of red, whereas the prediction of her hair is a typical brown colour. In general, among some images ICDLC seems to struggle a little with keeping a small amount of redness.

Another example of issues is bleed-over where a colour is shown where it should not be. This is most evident on the chair in Figure 7 where it is correct on the left side with a separation between the black and red, but on the right the red bleeds over onto where it should be black.

Image Name	**PSNR (dB)**
Michael	36.6264
Ted_Christopher_0001	33.5300
Ted_Costa_0001	34.9679
Ted_Maher_0001	31.2061
Ted_Maher_0002	33.0673
Ted_Nolan_0001	33.0749
Ted_Turner_0001	34.6175
Ted_Washington_0001	38.6514
Ted_Williams_0001	33.5204
Terence_Newman_0001	37.0385
Teresa_Graves_0001	42.9970
Teresa_Heinz_Kerry_0001	33.2055
Teresa_Williams_0001	34.8500
Tersa_Worbis_0001	34.8492
Teri_Files_0001	33.2591
Teri_Garr_0001	36.7415
Teri_ORourke_0001	29.1745
Average PSNR	**34.7869**

Table 1: PSNR of images

B. PSNR Results

Table 1 above shows the PSNR values of the images in Figure 9 as well as the PSNR of my own image in Figure 7 (michael). The images can be found in the LFW dataset and are named appropriately so they can be easily found if traversing the data set.

We can see that the average PSNR of the 17 sampled images is almost 35 dB. During ICDLC training and validation, the average PSNR was calculated to be about 35 dB (see Equation 2) which matches with our small data sample set above. Therefore, we can assume that the average PSNR calculated from images comes out to be around 35 dB.

$$Average\ Normalized\ MSE_{average_{normalized}} = 0.0012$$
$$MSE_{average} = 0.0012 * 128^2 = 19.6608$$
$$PSNR_{average} = 10 * log10\left(\frac{255^2}{MSE_{average}}\right) = 35.1948\ dB$$

Equation 2: Average PSNR from ICDLC network training

C. JPG Comparison

As a point of reference, the file michael.png was converted to

Figure 9: 16 Random images from LFW dataset, top images are original while bottom are predicted

JPG of varying quality and its PSNR and file size is displayed in Table 2. We can see that if we take the PNG image as input and output the greyscale image also in PNG format, we ultimately end up with a PSNR of 36.6264 dB with a file size of 38,740 bytes. The side information is a minuscule 128 bytes which added together is 38,868 bytes. In terms of quality we can see that the PSNR is roughly equal to 80 quality in terms of JPG. However, the file size is quite big and lands between 90-100 JPG quality, but more so to the lower end of that scale.

JPG Quality	PSNR (dB)	File Size (bytes)
PNG (original)	N/A	91,785
10	27.0578	3,580
20	29.9449	5,030
30	31.4237	6,191
40	32.4332	7,137
50	33.1734	8,052
60	33.8646	9,109
70	34.9469	10,677
80	36.3672	13,335
90	39.3215	20,102
100	50.0630	51,776
ICDLC (PNG greyscale)	36.6264	38,740 + 128 = 38,868
ICDLC (JPG greyscale)	33.3332	10,647 + 128 = 10,775

Table 2: PSNR and file size comparison of JPG and ICDLC

One way to minimize the size of our file is by using JPG for the greyscale image. With that we receive a PSNR of 33.3332 dB and a file size of 10,647 bytes. Once again, we have the side information of 128 bytes, so our total size is actually 10,775 bytes. In terms of file size, we have reduced it greatly and is equivalent in size to approximately a JPG image of quality 70. However, the PSNR has also lowered and is halfway between a JPG quality of 50 and 60.

While the compression in terms of PSNR is not currently better than standard JPG, there are areas of improvement that are mentioned in the future improvements section. As a personal estimate, those improvements would amount to a 2-3 PSNR increase at least, making ICDLC competitive to JPG in terms of compression.

D. PSNR not the only measurement

One thing to note is that PSNR is not complete story of how good an image is. At the end of the day it is how good a person seems to perceive the image that is of utmost important. One thing that is very important the ICDLC does that JPG and little other lossy compression types offer is that each pixel's luminance is not lost. It also offers a very easy encoding scheme.

As such the results displayed in this paper show great incentive as a first attempt at ICDLC. If the future improvements that are mentioned in the next section will be considered for the next iteration of ICDLC, there is a high probability (in my own opinion) that it will be able to surpass JPG in terms of PSNR vs file size eventually.

V. Future Improvements

A. Variable Input Sizes

There is much in terms of improvement that can be done to ICDLC. One of the improvements that can be done is instead of using a fixed 256x256 sized inputs, to have variable sized image inputs. This is possible since the convolutional neural network

will output the same size as the input but requires more work to allow this to happen. Given more time to work on this this would be another useful feature as it would also allow the training set images to be of different sizes, therefore allowing a larger and more diverse dataset to be used.

B. Fusion Layer

Another improvement that can be implemented is adding in a fusion layer. Since the network currently focuses on peoples faces, adding the fusion layer would allow ICDLC to take any type of input image, be it a person's portrait, a car, etc. While this can currently be attempted with the current iteration of ICDLC, the outcomes are not as good as when unless using a person's face.

C. Edge Detection

One of the biggest potential problems but easiest to fix would be in optimizing the edge detection algorithm. Currently the standard edge detection function from Python's Pillow library is used. At first use it worked very well but on some images, it can make mistakes and even provides false positives. Some of the bleed of colours seen on the predicted images is mainly caused by the weak edge detection algorithm, as already covered in the details section of this paper.

D. Side Information

Another issue is that some images have small details which do not get properly assigned the correct colour. This can clearly be seen in Figure 9 with the bottom right image. The microphone being held in the corner has yellow text in the original, but the predicted image shows those colours as much whiter than they should be. This is because the side information given is not good enough to help decide what colour the text on the microphones should be. The solution is to look at each 32x32 block of the original image and predicted image and calculate the PSNR of them individually and set a threshold. If the PSNR is above the threshold, keep the block as 32x32, however if it is below than reduce the size of the block to four 16x16 blocks. Keep doing the same procedure until the PSNR is above the threshold. This solution works but provides less and less compression for an image with many small details.

E. GPU Runtime

The last improvement would be to do on the training of the ICDLC neural network for longer. Currently the network ran for approximately 8.5 hours on a Tesla K80 which has 12 GB of Memory along with 61 GB RAM via FloydHub (https://www.floydhub.com). Some other networks (such as Xception) of the same complexity have been known to run for a few days to get their weights. As such running the network longer would probably be able to very easily gain another dB of PSNR, even 2 dB if being optimistic.

VI. User Manual

A. Files Given

The files enclosed with this paper contain the code needed to train, predict, encode, and decode the ICDLC network as well as calculate the PSNR of images. In the code directory there will be: the lfw directory, the image files michael.png and model.png, and the files decode.py, encode.py, main.py, my_model.h5, prediction.py, and psnr_calculator.m. All the includes, imports, and settings needed to run the files are included at the top of each file and should be done before proceeding.

B. Training and Testing Results

To train the network and obtain prediction results the main.py and prediction.py files will be required. Running the main.py file will simply train the network on 6000 images in the lfw directory. This will use up a lot of memory on the GPU and RAM. If training from scratch, the my_model.h5 file should be removed, as that is the ICDLC neural network weights model. Otherwise it will be imported, and training will continue from the current model. Additionally, the model.png image shows the image of the entire model in detail, being able to see every detail of the convolutional neural network.

Once satisfied with the training results the user can manually stop the code since the model is saved after every epoch or wait until all 100 epochs have finished. Then the user can run the prediction.py file which will store the original and predicted images as seen in Figure 9 in a results directory.

C. Custom Image Encoding and Decoding

If the user wishes to view how the network predicts their own image, they can do so. It is strongly recommended to use a PNG image of size 250x250, but not required. As an example, michael.png is available to use. Simple run encode.py and pass the first argument as the name of the file. Using michael.png as the file it would be:

```
% python encode.py michael.png
```

This will create the encoded and encoded_demo directories. The encoded directory holds the greyscale and side information but the encoded_demo is just an output that can be used to see the intermediate steps like the walkthrough done previously in this paper.

Next is simply running the decode.py, but it is important to note that the my_model.h5 file is required for this since the model must be loaded for the network to predict an image. This will create the decoded and decoded_demo directories in which the decoded holds only the final predicted image and the decoded_demo is for visualization purposes.

If the user wishes to try again with another image, they should delete all four of the newly created files: encoded, encoded_demo, decoded, and decoded_demo.

Also, if the PSNR is to be calculated, the file psnr_calculator.m is a MATLAB function that can be ran. It takes as its first input argument the location of the original extended image (256x256 in size) and as second input argument the location of the predicted image. It returns the PSNR between the predicted and the original.

D. FloydHub

For the training of the convolutional neural network of ICDLC FloydHub was used with its Tesla K80 GPU. To see the outputs, log, and code used one can simply go the three latest jobs:
https://www.floydhub.com/thechyz/projects/compression_colourization/70/

https://www.floydhub.com/thechyz/projects/compression_colourization/71/
https://www.floydhub.com/thechyz/projects/compression_colourization/72/

VII. References

[1] R. Zhang, P. Isola and A. A. Efros, *Colorfur Image Colorization.*

[2] S. Iizuka, E. Simo-Serra and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors".

[3] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions".

[4] University of Massachusetts, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments," [Online]. Available: http://vis-www.cs.umass.edu/lfw/#download. [Accessed 26 October 2018].

[5] taehoonlee, "GitHub," [Online]. Available: https://github.com/keras-team/keras-applications/blob/master/keras_applications/xception.py. [Accessed 18 November 2018].

[6] E. Wallner, "Colorizing B&W Photos with Neural Networks," FloydHub, [Online]. Available: https://blog.floydhub.com/colorizing-b-w-photos-with-neural-networks/. [Accessed 27 September 2018].