Enhancing Low Resolution Images with Convolutional Neural Network

MATH 596

Sandy Urazayev*

345; 12020 H.E.

But since I knew now that I could hope for nothing of greater value than frivolous pleasures, what point was there in denying myself of them?

– M. Proust

Abstract

We live in a rather boring world, full of inconsistencies and patterns that we fit our lives into. We still love it and try to capture its best shades for safe storing. Unfortunately, time goes and the sharpness of our pictures and data speedily fades. My final project would revolve around digital media enhancement and resolution improvement. We will build a convolutional neural network that will be able to learn the relationship between lower and higher resolution images.

Introduction

As we know, all digital imagse have different dimensions, encodings, sharpness, colors, etc. No matter what the combination is, the human perception is always the tower property of any digital media. The technology of the past was severely limited. For example, about 30 years ago, an average CRT monitor would have a resolution of 320x200, which is mere 64000 pixels. cite:crt Our current monitors have a resolution of at least 1920x1080 (2073600 pixels) and the

*University of Kansas (ctu@ku.edu)

new standard now is 3840x2160 (8294400 pixels). However, as technology progresses, the old recorded data doesn't.

Our old pictures, images, films, and movies in lower resolution don't get automatically "converted" into higher and better resolution media. As time progresses, the older media starts to scale badly and pales in comparison with current media. We also do understand that legacy has to be preserved. We have Criterion Collection, a whole industry company based on old films restoration. And many other companios are primarily focused on media restoration, such as images. However, the whole process is very manual in its method, which is proving to be very expensive and slow. We can do better than this.

This paper will focus on a way we can automate the procedure with a rather simple neural network and a couple of interesting techniques, such as convolution layers. Some of the work introduced here is inspired by a previous research done on super resolution images. cite:shi2016realtime

Data Overview

Machine Learning researchers cannot stress enough the importance of the training and validation data we use to fit our models. I have been researching the best ways to collect data.

Firstly, I had an idea to scrape the google image search results and then take those images, compress them, and use them as the basis for the data. This approach proved to introduce a couple of unfortunate complications, such as ethical issues with our datasets, image filtering, very varying dimensions of images, little pattern correlation, etc.

To address the issues above, I will use ImageNet image databases,cite:imagenet which is a carefully curated set of training images that have a set size and ethical issues resolved. I will use one of their Berkeley image nodes that has in total 500 images. Some examples from the set are shown below

(a) First Sample (b) Second Sample (c) Third Sample (d) Fourth Sample

Figure 1: ImageNet random samples

Design

Let's talk about how we would design the neural network to achieve our set goal. A traditional neural network design would look the following way

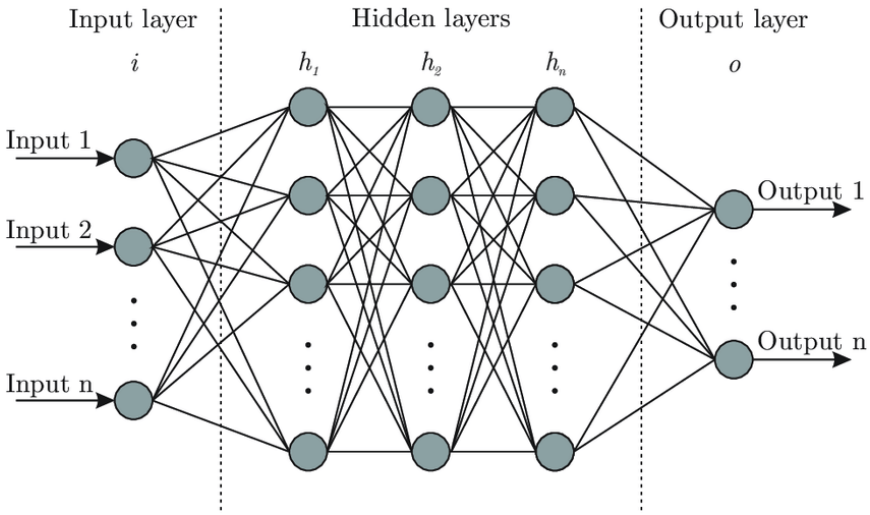


Figure 2: Traditional Neural Network

This has the same logic as our topics we learned in MATH 596. Neural Network is a big distributed version of a multi-layered matrix. The learning process is the process of filling up the values in our matrix and adjusting the weight of their connections, so then patterns would emerge from input-output relationship. The difference from convolutional neural network (CNN) is the application of milti-layered matrices in the neural network schema itself, therefore it is more deeply involved when fitting the training data. This of course increases the number of parameters we have to take care of. However, this is still the best available machine learning technique we have for computer vision applications. An example design of a CNN is below

Multiple pooling layers are used to combine the pooled data from those convolution layers.

Implementation

Our CNN implementation will be written in python using Tensorflow 2 and MacBook Pro M1 2020 hardware to train the model. First of all, we will define our input and output dimensions, which are 100x100 and 300x300, respectively.

```
# Basic input-output relationships
image_dimension = 300
```

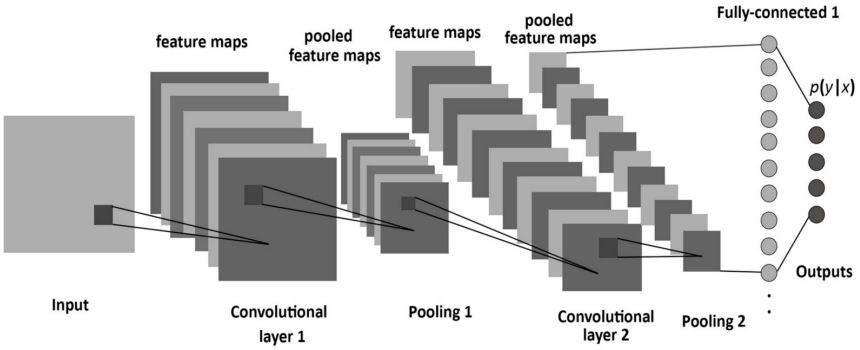


Figure 3: Convolutional Neural Network

```
scaling_factor = 3
input_dimension = image_dimension // scaling_factorp
```

Next, let us define the way we will process the input and the output images. Regular images have three layers: Red, Green, and Blue (RGB). However, running a training triage on all three layers would be very heavy on our computational resources. We will convert our images to YUV encoding, which would allow us to easily isolate the brightness layer (luma component) and train our model on the grayscale version of the image. Later on, just add on the other layers to complete the color gamma.

```
def scale_image(image, _):
    return image / 255.0

def prepare_input(image, image_size, scaling):
    yuv = tf.image.rgb_to_yuv(image)
    y, u, v = tf.split(yuv, 3, axis=(len(yuv.shape) - 1))
    return tf.image.resize(y, [image_size, image_size],
                           method="area")

def prepare_output(image):
    yuv = tf.image.rgb_to_yuv(image)
    y, u, v = tf.split(yuv, 3, axis=(len(yuv.shape) - 1))
    return y
```

- `scale_image` simply scales each image color encoded value from $[0, 255]$ to $[0, 1]$. This normalized version would ease off the burden of training.
- `prepare_input` simply resizes the image into the input requirements and extracts the Y channel from an YUV encoding of our image.

- `prepare_output` prepares the output images by extracting the same Y channel as the input layer.

Keras, our neural network training framework has a great function call for us to embed a whole training dataset with a single function call. Our training dataset and our validation dataset are validated below

```
train_directory = image_dataset_from_directory(
    images,
    batch_size=batch_size,
    image_size=(image_dimension, image_dimension),
    subset="training",
    validation_split=validation_split,
    seed=seed,
).map(scale_image)

validation_directory = image_dataset_from_directory(
    images,
    batch_size=batch_size,
    image_size=(image_dimension, image_dimension),
    subset="validation",
    validation_split=validation_split,
    seed=seed,
).map(scale_image)
```

For this paper, we will select the 10% of our whole dataset from ImageNet to be used for validation and the remaining 90% for training purposes.

Finally, let us get into the model structure. The beauty of convolution layers, is that we have to carefully arrange them together and properly match their dimensions. We will have a single input layer to accept 100x100 images, and 4 convolution layers, with 64 neurons. 64 is chosen by fine-tuning. It's a natural heap size and works out good!

```
def build_model():
    layer_configs = {
        "activation": "relu",
        "kernel_initializer": "Orthogonal",
        "padding": "same",
    }

    input_layer = keras.Input(shape=(None, None, 1))
    x = layers.Conv2D(64, 5, **layer_configs)(input_layer)
    x = layers.Conv2D(64, 5, **layer_configs)(x)
    x = layers.Conv2D(64, 5, **layer_configs)(x)
    x = layers.Conv2D(scaling_factor ** 2, 3, **layer_configs)(x)
```

```
    output_layer = tf.nn.depth_to_space(x, scaling_factor)

    return keras.Model(input_layer, output_layer)

model = build_model()
model.summary()
```

The resulting model is summarized as follows

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 1)]   0
_________________________________________________________________
conv2d (Conv2D)              (None, None, None, 64)    1664
_________________________________________________________________
conv2d_1 (Conv2D)            (None, None, None, 64)    102464
_________________________________________________________________
conv2d_2 (Conv2D)            (None, None, None, 64)    102464
_________________________________________________________________
conv2d_3 (Conv2D)            (None, None, None, 9)     5193
_________________________________________________________________
tf.nn.depth_to_space (TFOpLa (None, None, None, 1)     0
=================================================================
Total params: 211,785
Trainable params: 211,785
Non-trainable params: 0
```

Mathematical Basis

Relu is a balancing function of the form relu$(x) = \max\{0, x\}$ that will activate only the neurons that really matter to the patterns we're studying. If a connection on the weight bias is weak, relu will floor it to 0.

For our main optimization method, we will use a variance of Stochastic Optimization, called Adam Algorithm. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

Convolutional Neural Networks incorporate all the topics we studied during this semester. First of all, the optimization method, Adam optimization is a stochastic gradient descent method that is trying to find the best approximations for global minima, which are our errors. The greatest goal of any neural network is to minimize the error of our model. We do that by trying to find the global minima of our error space.

For error measurement, we will use a Peak signal-to-noise ratio (PSNR) error approximation. It's similar to RMSE, with a couple of modifications that allow us to apply it to differently-sized source and target matrices. Let I be our original "ground-truth" image and K our approximation of the image. Both have type $\mathbb{Z}^{m\times m}$ Then let MSE be

$$MSE = \frac{1}{m^2}\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}[I(i,j)-K(i,j)]^2 \tag{1}$$

then PSNR is defined as

$$PSNR = 20\times\log_{10}(MAX_I)-10\times\log_{10}(MSE) \tag{2}$$

In this case, higher PSNR value means that the approximation is closer to the source.

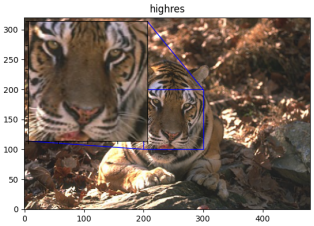
Results

The training for this model took about ~2 hours just in 50 epochs and 450 images. The better results would be achieved by training at least for 200 epochs with more than 100 thousand images. Let us review our results. I will use a matplotlib module for enhancing a small part of the image's center to show the improvements the model can bring.

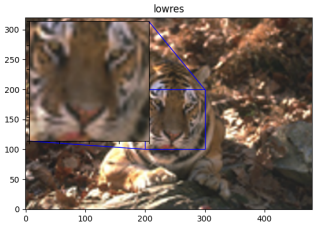
Recall that a higher PSNR symbolizes a better approximation and the following test images have never been seen by our model before. They are clean images pulled up from untouched image datasets. The value in the captions is the PSNR against their respective originals.

Notice that on average, the PSNR of our predictions are higher than the downscaled ones. This is a firm confirmation that the sharpening works! Even a model with as little training as this one, we were able to enhance an low resolution image with a convolutional neural network.

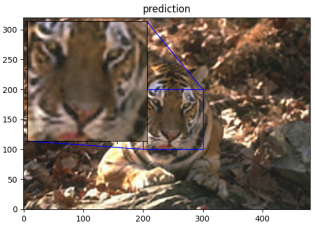
bibliography:paper.bib bibliographystyle:ieeetr



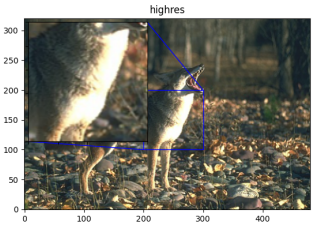
(a) Original



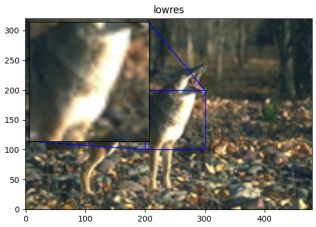
(b) Downscaled, 24.4263



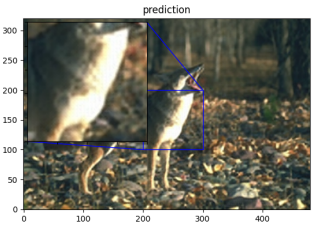
(c) Prediction, 25.3418



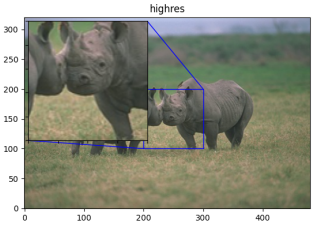
(d) Original



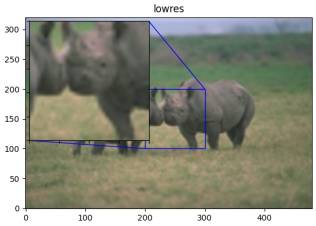
(e) Downscaled, 25.0398



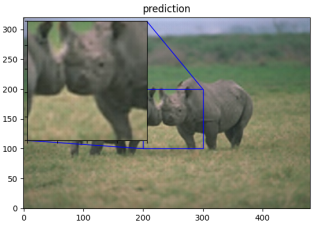
(f) Prediction, 25.6215



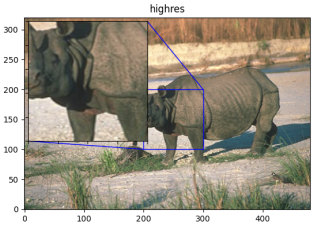
(g) Original



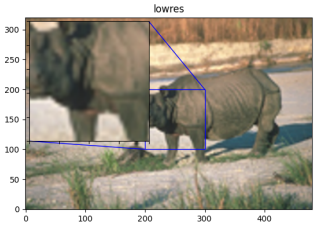
(h) Downscaled, 34.9506



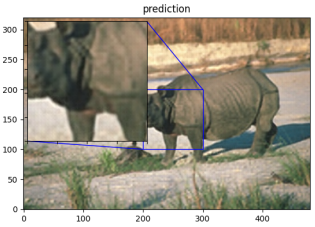
(i) Prediction, 34.9005



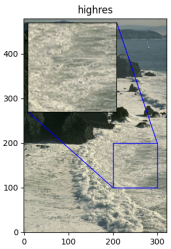
(j) Original



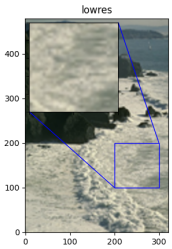
(k) Downscaled, 26.6250



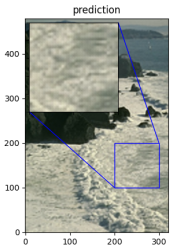
(l) Prediction, 27.0954



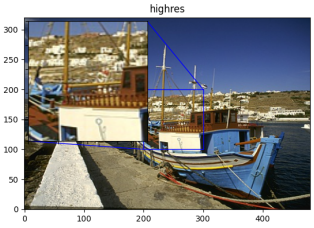
(m) Original



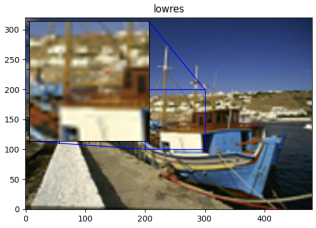
(n) Downscaled, 25.1637



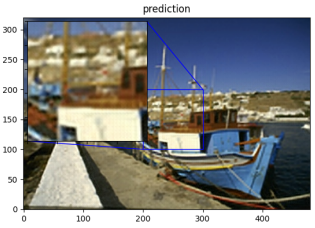
(o) Prediction, 25.5566



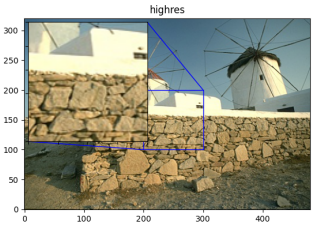
(a) Original



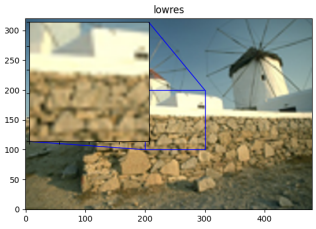
(b) Downscaled, 22.6681



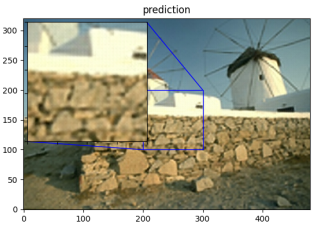
(c) Prediction, 23.2446



(d) Original

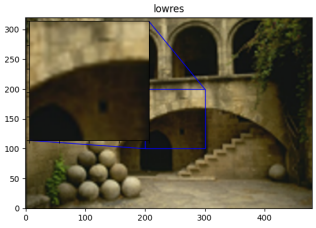


(e) Downscaled, 23.9902

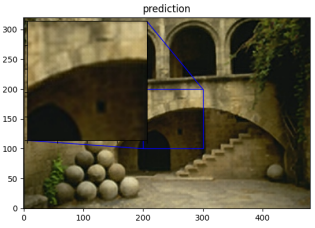


(f) Prediction, 24.4366

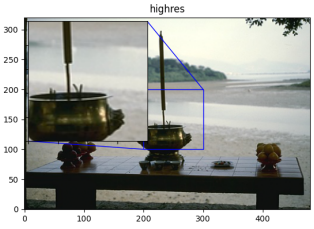
(g) Original



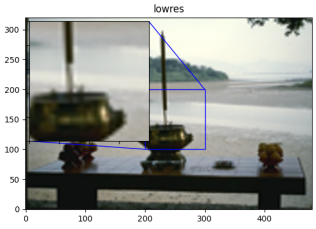
(h) Downscaled, 26.3472



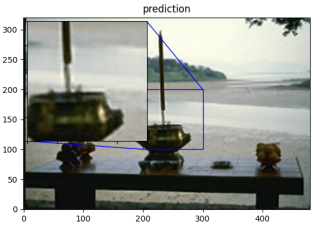
(i) Prediction, 26.8450



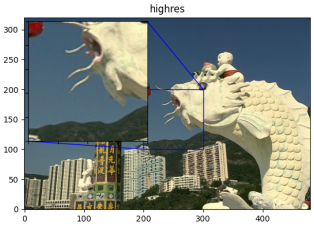
(j) Original



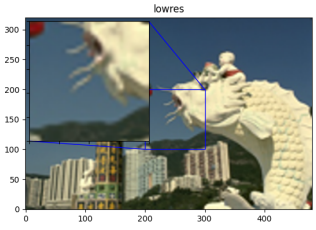
(k) Downscaled, 27.6660



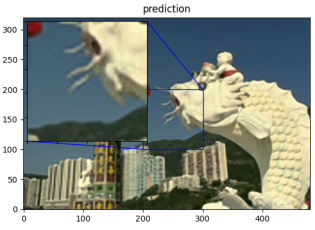
(l) Prediction, 29.0058



(m) Original



(n) Downscaled, 21.8129



(o) Prediction, 22.2765