Visual Recognition - Assignment 4

Biswesh Mohapatra | IMT2016008

April 14, 2019

Contents

1 AUTO ENCODER

Problem Statement:

Implement Convolution2D, Pooling2D, Upsampling Layer

1. Implement Conv2D, Pool2D, Upsampling from scratch without using Keras
2. Take a trained Conv. Autoencoder (using Keras, the one discussed in the class) and save the weights as Numpy
3. Make a forward pass with your implemented layers and reproduce the actual output
4. Compare the output of your implementation with Keras implementation

1.1 BUILDING AUTOENCODER USING KERAS

I trained an autoencoder with the MNSIT Dataset using Keras. The following results of the original images and their corresponding decoded images can be seen in Figure 1.1.

I used the architecture given below and trained it for 30 epochs. I used my own implementation of autoencoder where I used a sequential model for an encoder and another sequential model for decoder. Later I combined them together in another sequential model which became my main model. The benefit of this was that I could use the encoders and decoders separately as well as combined together.

Figure 1.1: Keras Auto-encoder output where decoded images are below original images

Listing 1: Encoder Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 8)           0
=================================================================
Total params: 1,904
Trainable params: 1,904
Non-trainable params: 0
_________________________________________________________________
```

Listing 2: Decoder Architecture

```
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
Total params: 2,481
Trainable params: 2,481
Non-trainable params: 0
```

Listing 3: Autoencoder Implementation using Keras

```
class Model():
    def __init__(self, params):
        self.params = params
        self.model = Sequential()

        self.encoder = Sequential()
        self.encoder.add(Conv2D(16, (3, 3), activation='relu', padding='same',
                        input_shape=self.params.img_shape))
        self.encoder.add(MaxPooling2D(pool_size=(2,2), padding='same'))
        self.encoder.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
        self.encoder.add(MaxPooling2D(pool_size=(2,2), padding='same'))
        self.encoder.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
        self.encoder.add(MaxPooling2D(pool_size=(2,2), padding='same'))

        self.decoder = Sequential()
        self.decoder.add(Conv2D(8, (3, 3), activation='relu', padding='same',
                        input_shape=self.encoder.layers[-1].output_shape[1:]))
        self.decoder.add(UpSampling2D((2,2)))
        self.decoder.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
        self.decoder.add(UpSampling2D((2,2)))
        self.decoder.add(Conv2D(16, (3, 3), activation='relu'))
        self.decoder.add(UpSampling2D((2,2)))
        self.decoder.add(Conv2D(1, (3, 3), activation='sigmoid', padding='same'))
        self.model.add(self.encoder)
        self.model.add(self.decoder)

    def compile(self, loss="binary_crossentropy", metrics=['accuracy']):
        self.model.compile(loss=loss, optimizer=self.params.optimiser)

    def fit(self, x_train, y_train, validation_data, shuffle=True,
        callbacks=[TensorBoard(log_dir='conv_autoencoder')], verbose=2):
        self.model.fit(x_train, y_train, epochs=self.params.epochs,
                        batch_size=self.params.batch_size, shuffle=shuffle,
                        validation_data=validation_data,
                       callbacks=callbacks, verbose=verbose)
```

1.2 My own functions

Following are my own implementation of convolution, upsampling and maxpool function.

1.2.1 Convolution

The function takes in the trained weights as the inputs and then uses basic convolution operation on the input matrix. It also takes care of padding. It assumes stride to be 1.

Listing 4: Convolution Implementation

```
def my_conv2d(input_matrix, kernel_weights, padding="not same"):
    s = kernel_weights[0].shape
    output_channels = s[3]
    input_channels = s[2]
    output=[]
    img_rows = input_matrix.shape[0]
    img_cols = input_matrix.shape[1]
    row_pad_cut = int((s[0]-1)/2)  #number of rows to be cut if padding is not same
    col_pad_cut = int((s[1]-1)/2)

    if padding == "same":
        output=np.zeros((img_rows, img_cols, output_channels))
    else:
        output=np.zeros((img_rows-2*row_pad_cut, img_cols-2*col_pad_cut, output_channels))

    for out_chan in range(output_channels):
        for in_chan in range(input_channels):
            mat = input_matrix[:,:,in_chan]
            k = kernel_weights[0][:,:,in_chan, out_chan]
            for x in range(img_rows):
                for y in range(img_cols):
                    if padding == "same":
                        output[x,y,out_chan] += convolve(x, y, mat, k)
                    elif (x>=row_pad_cut and y>=col_pad_cut
                            and x<(img_rows-row_pad_cut) and y<(img_cols-col_pad_cut)):
                        output[x-row_pad_cut, y-row_pad_cut,out_chan] +=
                                        convolve(x, y, mat, k)

    return output
```

I built a helping function to do the final convolution given the x,y coordinates along with the called convolve. It can be seen here.

Listing 5: Convolve function

```
def convolve(x, y, mat, k):
    val = 0
    center_x = int((k.shape[0]-1)/2); center_y = int((k.shape[1]-1)/2)
    for i in range(-1*center_x, center_x+1):
        for j in range(-1*center_y, center_y+1):
            idx_x = x+i
            idx_y = y+j
            if idx_x>=0 and idx_y>=0 and idx_x<mat.shape[0] and idx_y<mat.shape[1]:
                val+=mat[idx_x][idx_y]*k[i+center_x][j+center_y]
    return val
```

1.2.2 Max Pool

My implementation of max pool assumes stride to be same as shape given.

Listing 6: Maxpool Implementation

```
def my_maxpool(shape, matrix):
    rows=int(math.ceil(matrix.shape[0]/shape[0]))
    cols=int(math.ceil(matrix.shape[1]/shape[1]))
    output=np.zeros((rows, cols, matrix.shape[2]))

    for out_chan in range(matrix.shape[2]):
        for x in range(0, matrix.shape[0], shape[0]):
            for y in range(0, matrix.shape[1], shape[1]):
                mx = -1
                for i in range(0,shape[0]):
                    for j in range(0, shape[1]):
                        if (x+i)<matrix.shape[0] and (y+j)<matrix.shape[1]:
                            if matrix[x+i][y+j][out_chan]>mx:
                                mx=matrix[x+i][y+j][out_chan]
                output[int(x/shape[0])][int(y/shape[1])][out_chan]=mx

    return output
```

1.2.3 Up Sampling

Here the values are copied in the given number times that is given by the size as a parameter.

Listing 7: Upsampling Implementation

```
def my_upsample(shape, matrix):
    rows=matrix.shape[0]*shape[0]
    cols=matrix.shape[1]*shape[1]
    output=np.zeros((rows, cols, matrix.shape[2]))

    for out_chan in range(matrix.shape[2]):
        for x in range(matrix.shape[0]):
            for y in range(matrix.shape[1]):
                for i in range(shape[0]):
                    for j in range(shape[1]):
                     output[x*shape[0]+i][y*shape[0]+j][out_chan]=matrix[x][y][out_chan]

    return output
```

1.3 Decoding using my own functions

On using the functions in the way mentioned below, I was able to generate the image using my own implementations.

Listing 8: Final usage

```
params = Parameters(img_shape=x_train.shape[1:], epochs=30,
                    optimiser=keras.optimizers.Adadelta(lr=0.5, rho=0.95,
                    epsilon=None, decay=0.0))
m = Model(params)
m.compile()
m.fit(x_train, x_train, (x_test, x_test), verbose=1)
idx=4
input_matrix = x_test[idx]
z = my_conv2d(input_matrix, m.encoder.layers[0].get_weights(), "same")
z = np.maximum(0,z)
z = my_maxpool((2,2),z)
z = my_conv2d(z, m.encoder.layers[2].get_weights(), "same")
z = np.maximum(0,z)
z = my_maxpool((2,2),z)
z = my_conv2d(z, m.encoder.layers[4].get_weights(), "same")
z = np.maximum(0,z)
z = my_maxpool((2,2),z)

z = my_conv2d(z, m.decoder.layers[0].get_weights(), "same")
z = np.maximum(0,z)
z = my_upsample((2,2), z)
z = my_conv2d(z, m.decoder.layers[2].get_weights(), "same")
z = np.maximum(0,z)
z = my_upsample((2,2), z)
z = my_conv2d(z, m.decoder.layers[4].get_weights())
z = np.maximum(0,z)
z = my_upsample((2,2), z)
z = my_conv2d(z, m.decoder.layers[6].get_weights(), "same")
z = (np.vectorize(sigmoid))(z)
```

Figure 1.2: decoded images using my own implementation is below the original images

---

2 IMPLEMENT ONLY DECODER

Problem Statement: Write a decoder to convert random noise into image (MNIST images). Use Keras for this. For any random vector, I provide, the decoder should output a MNIST like image

2.1 PHASE 1

A decoder was trained on MNIST images on random noise. Initially I used random numbers for random images. There was no connection between input noise and the label image. This produced a very generic image after training with white in the middle and black elsewhere.

2.2 PHASE 2

Later I decided to use some relation between random numbers and the mnist images. Hence I set the min and max values of the random generator according to the image which I was training it on. Later on passing some random numbers, I was able to see images very much like MNIST images resembling to the connection that the noise had with its min and max number.

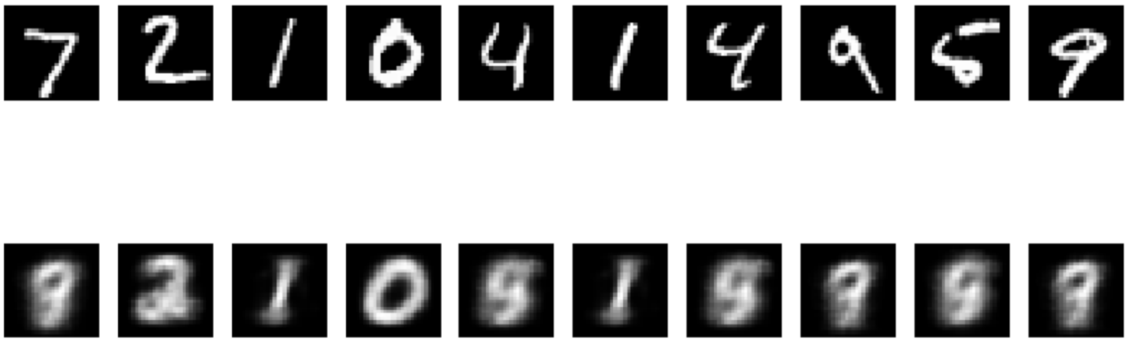

Figure 2.1: First row is original. Second row is decoded images from noise.

We can observe that the noisy image roughly appears to be the digit as shown in the original image. Here the original images are test images and we had generated some noise according to the values they represent. eg. if thhe test image is that of 0 then a noise picked from range of (0, 100) would more likely give an image close to 0 while a noise picked from range of (100, 200) would likely give an image close to 1 etc.

Listing 9: Generate noise encoding

```
train_noise=[]
for i in range(x_train.shape[0]):
        train_noise.append(np.random.uniform(y_train[i]*100, (y_train[i]+1)*100, (1, 1, 100)))

train_noise = np.array(train_noise)

test_noise=[]
for i in range(x_test.shape[0]):
        test_noise.append(np.random.uniform(y_test[i]*100, (y_test[i]+1)*100, (1, 1, 100)))

test_noise = np.array(test_noise)
```

Listing 10: Generating new images from trained decoder

```
n = 10
decoded_imgs = m.decoder.predict(test_noise)
plt.figure(figsize=(10, 4), dpi=100)
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.set_axis_off()

    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.set_axis_off()

plt.show()
```

REFERENCES

[1] https://github.com/snatch59/keras-autoencoders/blob/master/convolutional_autoencoder.py.

[2] Keras: Documentation. https://keras.io/.

[3] Numpy: Documentation. https://docs.scipy.org/doc/.