Rohith Yogi (IMT2016072)

Visual Recognition

Assignment 4 (REPORT)

Overview

The report will contain the details involved in the Implementation of Convolution2D, Pooling2D and Upsampling in general. And also Implementation of a decoder so that it converts random noise into an image. These will be covered in two parts.

Question

Part 1: Implement Convolution2D, Pooling2D, Up-Sampling

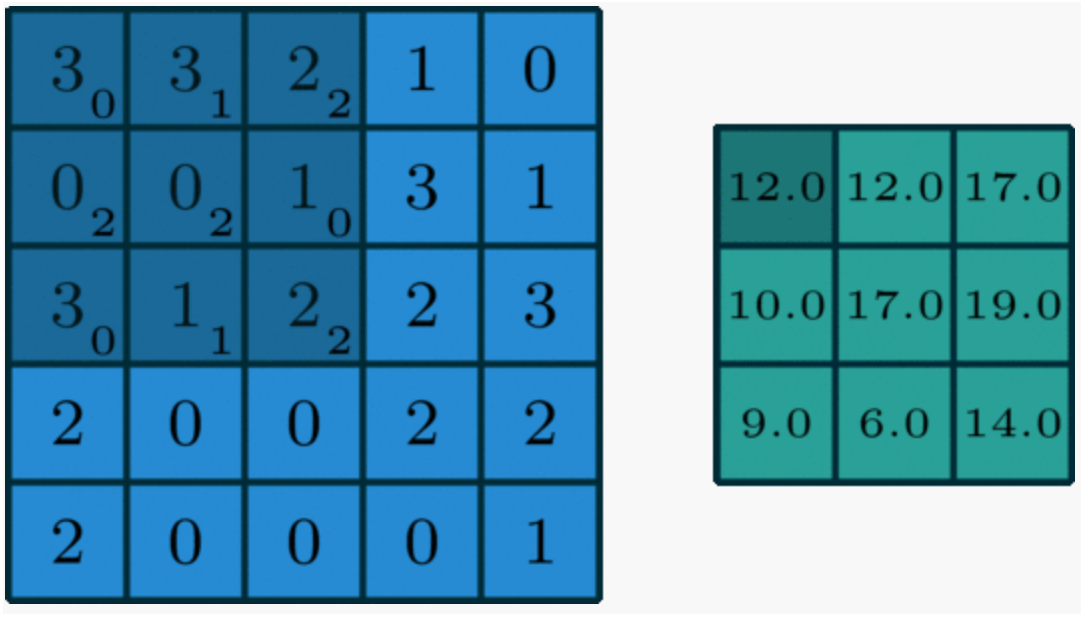
- Implement Conv2D, Pool2D, Up-Sampling from scratch without using Keras
- Take a trained Conv. Autoencoder (using Keras, the one discussed in the class) and Save the weights as Numpy
- Make a forward pass with your implemented layers and reproduce the actual output
- Compare the output of your implementation with Keras implementation

Part 2: Implement only decoder

- 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.

PART I

To Implement a Convolution2D operation a prior knowledge to what the operation does is very important.



For Example, the above matrix is convolved with a 3x3 matrix whose row values are [0 1 2] ; [2 2 0] and [0 1 2] respectively and the convolution operations happen in the following manner :

- We align the convolution matrix parallel with the original matrix and perform dot product followed by the summation of the values obtained. In the above example this gives us [0*3 + 1*3 + 2*2 + 0*2 + 0*2 + 1*0 + 3*0 + 1*1 + 2*2] = 12.
- The remaining values are also calculated in a similar fashion. Just by moving the alignment.
- This is just a regular convolution. But there also few techniques such as padding and strides.

- Padding: To pad the edges with extra pixels (we use 0, hence the term “zero padding”).
- Strides: To skip some of the slide locations of the kernel in order to obtain an output with a lower size than the input.
- This can also be done using Pooling. Where we look at the internal matrix and consider only the max value (max pool). Similarly, other poolings can also be implemented.

Conv2D and MaxPool are available in the keras library. But these can be implemented without using the library as well. Also, the activation functions play a important role in these implementations.

Keras Conv2D has the following parameters.

- Filters,
- kernel_size,
- strides=(1, 1),
- padding='valid',
- data_format=None,
- dilation_rate=(1, 1),
- activation=None,
- use_bias=True,
- kernel_initializer='glorot_uniform',
- bias_initializer='zeros',
- kernel_regularizer=None,
- bias_regularizer=None,
- activity_regularizer=None,
- kernel_constraint=None,
- bias_constraint=None

In the keras implementation shown we only use 4 parameters.

Ex: Conv2D(16, (3, 3), activation='relu', padding='same')

They are :

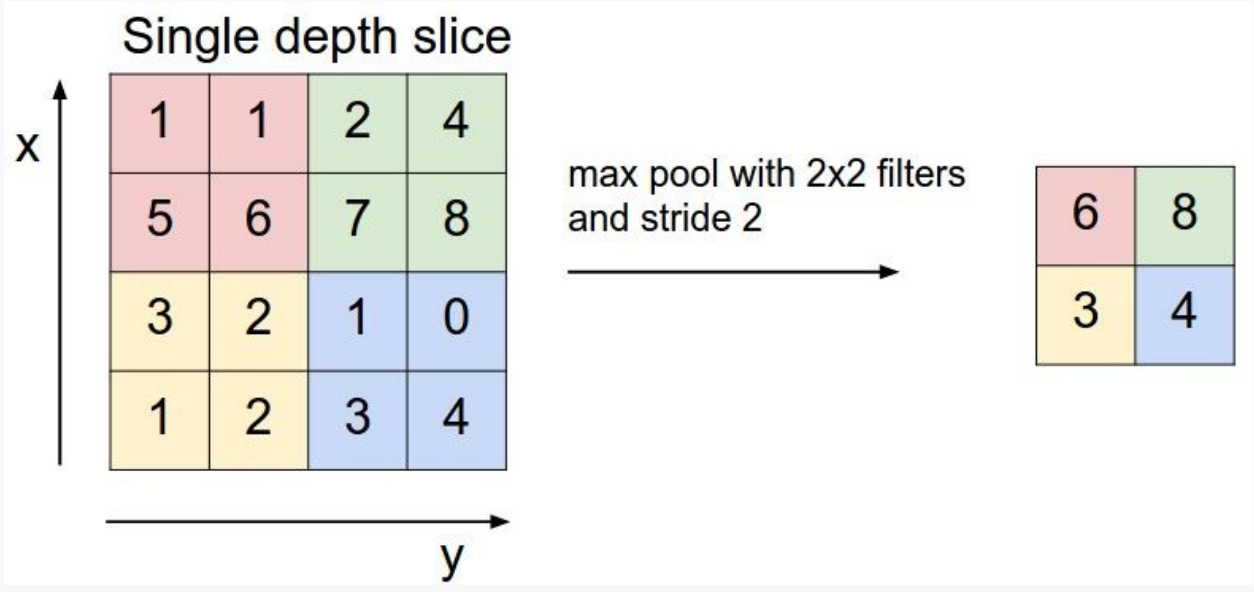
- Filters =16,
- kernel size is (3,3),
- Activation used is relu,
- Padding is the same.

So the implementation consists of these parameters use itself.

Conv2D:

def myConv2D(X, W, filters, kernel_size, strides=1, padding='valid', activation=None, bias=0).

Pool2D:



Similarly, keras MaxPooling2D consists of parameters such as

- Pool_size,
- Strides,
- padding='valid',
- data_format=None

In the keras implementation shown we use only 2 parameters.

Ex: MaxPooling2D((2, 2), padding='same')

They are :

- Pool size is (2,2),
- Padding is the same.

def myMaxPool2D(X, kernel_size, strides=2, padding='valid'):

Keras Upsampling2D consists of

- size,
- data_format=None,
- interpolation='nearest'

In the keras implementation shown we use only 1 parameter.

Ex: UpSampling2D((2, 2))

That is :

- Size = (2,2)

Up-Sampling: We use this to transform images of low resolution to high resolution. This can be implemented in various ways. But I used this approach.

Up-Sampling:

def myUpSampling2D(X, kernel_size, data_format='channels_last', interpolation='nearest'):

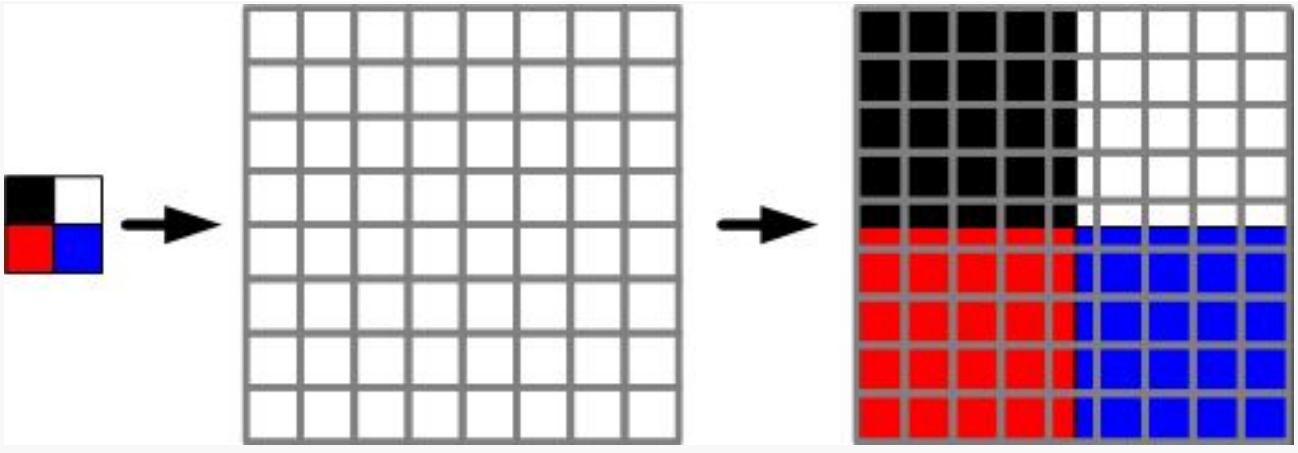
First, we are given an input matrix of size nxn and a kernel of size mxm then we need a output of size (nxm)x(nxm). For example.

We have a 2x2 matrix an input image then we have a 8x8 as our kernel size then we get an output of 16x16.

If we have a,b,c,d elements in the 2x2 matrix they will finally be sampled in the format shown below. Where we can assume black denotes a , white b and red and blue for c and d respectively.

This can be implemented without using keras. By using only the parameters like

- Input image,
- Kernel size.



On Making a forward pass with my implemented layers to reproduce the actual output. I have obtained the results below.

After that, I compared the output of my implementation with Keras implementation as you can see below.

The results of these can be clearly seen below.

Keras CNN area under the curve = 0.99969230622

My implementation area under the curve = 0.997107608775

Train on 60000 samples, validate on 10000 samples

Epoch 1/10

60000/60000 [==============================] - 32s 539us/step - loss: 0.9790 - val_loss: 0.3186

Epoch 2/10

60000/60000 [==============================] - 31s 510us/step - loss: 0.1405 - val_loss: 0.0451

Epoch 3/10

60000/60000 [==============================] - 36s 599us/step - loss: 0.0354 - val_loss: 0.0292

Epoch 4/10

60000/60000 [==============================] - 36s 593us/step - loss: 0.0242 - val_loss: 0.0244

Epoch 5/10

60000/60000 [==============================] - 22s 375us/step - loss: 0.0195 - val_loss: 0.0208

Epoch 6/10

60000/60000 [==============================] - 31s 509us/step - loss: 0.0163 - val_loss: 0.0194

Epoch 7/10

60000/60000 [==============================] - 43s 719us/step - loss: 0.0142 - val_loss: 0.0180

Epoch 8/10

60000/60000 [==============================] - 32s 529us/step - loss: 0.0130 - val_loss: 0.0176

Epoch 9/10

60000/60000 [==============================] - 32s 534us/step - loss: 0.0124 - val_loss: 0.0173

Epoch 10/10

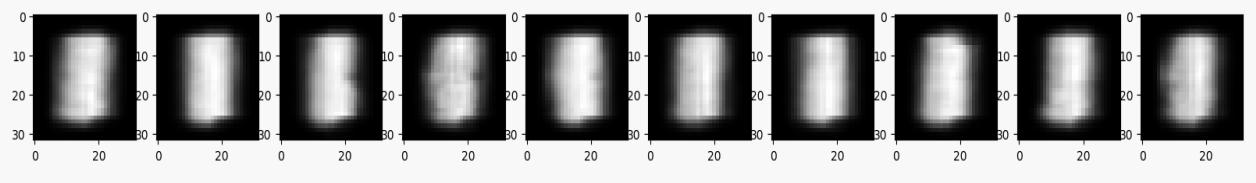
60000/60000 [==============================] - 32s 534us/step - loss: 0.0114 - val_loss: 0.0153

And on comparing the output of your implementation with Keras implementation.

PART II

The decoder implementation is quite simple. The main objective of the decoder is to convert random noise into image (MNIST images).

The output image:



After using the implementation of the decoder we need to use the random noise image generated by us to decode its nature to be a MNIST image.

This task is fairly simple but requires training through the data.

Results:

```
Epoch 1/10
60000/60000 [==============================] - 96s 2ms/step - loss: 0.3363
Epoch 2/10
60000/60000 [==============================] - 73s 1ms/step - loss: 0.2898
Epoch 3/10
60000/60000 [==============================] - 47s 778us/step - loss: 0.2864
Epoch 4/10
60000/60000 [==============================] - 47s 781us/step - loss: 0.2799
Epoch 5/10
60000/60000 [==============================] - 46s 771us/step - loss: 0.2756
Epoch 6/10
60000/60000 [==============================] - 47s 778us/step - loss: 0.2715
Epoch 7/10
60000/60000 [==============================] - 54s 901us/step - loss: 0.2696
Epoch 8/10
60000/60000 [==============================] - 69s 1ms/step - loss: 0.2690
Epoch 9/10
60000/60000 [==============================] - 48s 793us/step - loss: 0.2685
Epoch 10/10
60000/60000 [==============================] - 47s 779us/step - loss: 0.2681
```

Resources:

http://stackoverflow.com/

https://docs.opencv.org/

https://towardsdatascience.com/