Assignment-4

Vaibhav Kansagara (IMT2016068)
Visual Recognition
International Institute of Information Technology Bangalore

I. The Key Processes

Important steps which help the identification of patterns in an image:

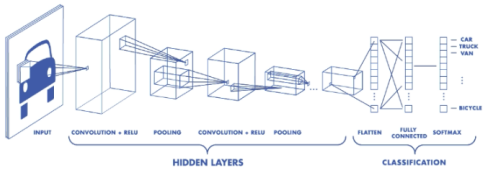

Fig. 1: CNN architecture

- Convolution
- Pooling
- Flattening
- Full Connection

A. Convolution:

Convolution is performed on an image to identify certain features in an image. Convolution helps in blurring, sharpening, edge detection, noise reduction and more on an image that can help the machine to learn specific characteristics of an image. My Implementation is as follows:

```
def m_Conv2D(img, img_filters, padding = "not same"):
    layer_shape = img_filters[0].shape
    print(layer_shape)

    img_rows = img.shape[0]
    img_col = img.shape[1]

    in_chan = layer_shape[2]
    out_chan = layer_shape[3]

    rows_cut = int((layer_shape[0] - 1) / 2)
    col_cut = int((layer_shape[1] - 1) / 2)

    output = []

    new_img = []

    if padding == "same":
        new_img = np.empty((img_rows + 2 * rows_cut, img_col + 2 * col_cut, in_chan))
        for input_channel in range(in_chan):
            new_img[:, :, input_channel] = np.pad(img[:, :, input_channel], ((rows_cut, rows_cut),(col_cut, col_cut)), 'constant')
        output = np.zeros((img_rows, img_col, out_chan))
    else:
        new_img = np.zeros((img_rows, img_col, in_chan))
        output = np.zeros((img_rows - 2 * rows_cut, img_col - 2 * col_cut, out_chan))
        new_img = img

    new_img_rows = new_img.shape[0]
    new_img_col = new_img.shape[1]

    for out_chan in range(out_chan):
        for input_channel in range(in_chan):
            img_channel = new_img[:, :, input_channel]
            img_filter = img_filters[0][:, : , input_channel, out_chan]

            for x in range(new_img_rows):
                for y in range(new_img_col):
                    if x + img_filter.shape[0] <= new_img_rows and y + img_filter.shape[1] <= new_img_col:
                        output[x, y, out_chan] += convolve(x, y, img_channel, img_filter)
    return output
```

B. Pooling:

A convoluted image can be too large and therefore needs to be reduced. Pooling is mainly done to reduce the image without losing features or patterns.

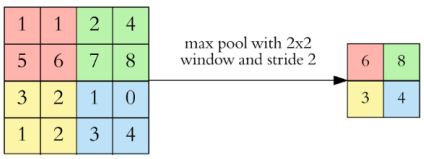

Fig. 2: Input and output

The most common type of pooling is max pooling which just takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the max value in the window. Similar to a convolution, we specify the window size and stride.
My Implementation of Pooling is Following.

```
def m_MaxPooling2D(img, pool_size):
    img_rows = img.shape[0]
    img_columns = img.shape[1]
    pool_rows = pool_size[0]
    pool_columns = pool_size[1]
    out_chan = img.shape[2]

```

```
    output = np.empty((int(math.ceil(img_rows /
    pool_rows)), int(math.ceil(img_columns /
    pool_columns)), out_chan))

    for out_chan in range(out_chan):
        for x in range(0, img_rows, pool_rows):
            for y in range(0, img_columns,
    pool_columns):
                max_num = 0
                for i in range(pool_rows):
                    for j in range(pool_columns):
                        if x + i < img_rows and y +
    j < img_columns:
                            max_num = max(
    max_num, img[x + i][y + j][out_chan])
                output[int(x / pool_rows)][int(y /
    pool_columns)][out_chan] = max_num
    return output
```

C. *Upsampling:*

Upsampling is a process of increasing the dimension of the image as opposed to Pooling of a image. Here nearest neighbour interpolation is used to repeat the values in the output images.
My Implementation of upsampling:

```
def m_UpSampling2D(img, upsample_size):
    img_rows = img.shape[0]
    img_columns = img.shape[1]
    upsample_rows = upsample_size[0]
    upsample_columns = upsample_size[1]
    out_chan = img.shape[2]

    output = np.empty((img_rows * upsample_rows,
    img_columns * upsample_columns, out_chan))

    for out_chan in range(out_chan):
        for x in range(img_rows):
            for y in range(img_columns):
                for i in range(upsample_rows):
                    for j in range(upsample_columns)
    :
                        output[x * upsample_rows + i
    ][y * upsample_columns + j][out_chan] = img[x][y
    ][out_chan]
    return output
```

Fig. 3: Output_1

II. Encoders and Decoders:

Decoder Implementation using Keras:

```
in_shape = (4, 4, 64)

in_img = Input(shape = in_shape)


x = Conv2D(32, (3, 3), activation='relu', padding='
    same')(in_img)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding= '
    same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='
    valid')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid',
    padding='same')(x)


print("shape of decoded : ", K.int_shape(decoded))
```

Epocs value from my Decoder.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
 - 3s - loss: 0.2813 - val_loss: 0.2555
Epoch 2/15
 - 3s - loss: 0.2520 - val_loss: 0.2530
Epoch 3/15
 - 3s - loss: 0.2469 - val_loss: 0.2428
Epoch 4/15
 - 3s - loss: 0.2440 - val_loss: 0.2404
Epoch 5/15
 - 3s - loss: 0.2421 - val_loss: 0.2408
Epoch 6/15
 - 3s - loss: 0.2408 - val_loss: 0.2395
Epoch 7/15
 - 3s - loss: 0.2396 - val_loss: 0.2361
Epoch 8/15
 - 3s - loss: 0.2386 - val_loss: 0.2359
Epoch 9/15
 - 3s - loss: 0.2379 - val_loss: 0.2371
Epoch 10/15
 - 3s - loss: 0.2372 - val_loss: 0.2360
Epoch 11/15
 - 3s - loss: 0.2367 - val_loss: 0.2353
Epoch 12/15
 - 3s - loss: 0.2362 - val_loss: 0.2379
Epoch 13/15
 - 3s - loss: 0.2357 - val_loss: 0.2331
Epoch 14/15
 - 3s - loss: 0.2353 - val_loss: 0.2343
Epoch 15/15
 - 3s - loss: 0.2349 - val_loss: 0.2316
```

Fig. 4: Epocs

Summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 32)          18464
_________________________________________________________________
up_sampling2d_7 (UpSampling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 8, 32)          9248
_________________________________________________________________
up_sampling2d_8 (UpSampling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 14, 14, 32)        9248
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 1)         289
=================================================================
Total params: 37,249
Trainable params: 37,249
Non-trainable params: 0
_________________________________________________________________
None
```

Fig. 5: Summary

Output of 2nd program:

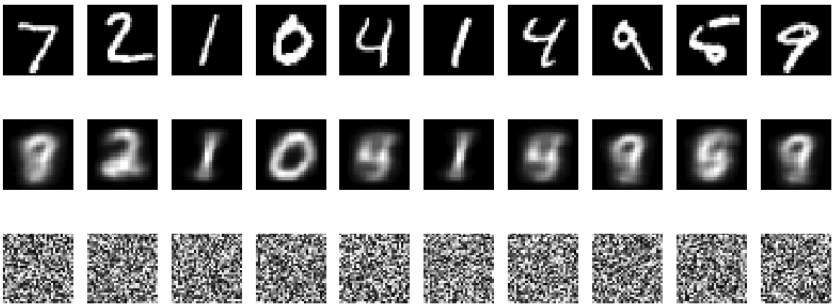

Fig. 6: Output_2