In [1]:
```
from tensorflow import keras
from tensorflow.keras import layers
from keras import models
from keras.models import Sequential
# from sklearn.metrics import confusion_matrix, cohen_kappa_score
import numpy as np
```

In [2]:
```
# MATLAB Code was:
# for i = 1:10
#     cl(i) = convolution2dLayer([2,2],1,"Bias",zeros(1,1,1),"Weights",[ones(1,2,1,
#         "Padding",[1 0 1 0],"Name","pasc"+int2str(i));
# end
# before we begin then, let's show how to stack matrices in python similarly
```

In [3]:
```
a = np.vstack((np.ones((1,2,1,1)),np.zeros((1,2,1,1))))
print(a)
np.shape(a)
```

```
[[[[1.]]

  [[1.]]]


 [[[0.]]

  [[0.]]]]
(2, 2, 1, 1)
```

Out[3]:

In [4]:
```
w = list(range(2))
w[0] = np.vstack((np.ones((1,2,1,1)),np.zeros((1,2,1,1)))) # perhaps this matches c
w[1] = np.zeros(1,) # bias. not (1,1,1) it's just as though squeezed, 1 each channe

print(w)
print(np.shape(w[1]))
```

```
[array([[[[1.]],

         [[1.]]],


        [[[0.]],

         [[0.]]]]), array([0.])]
(1,)
```

In [5]:
```
model = keras.Sequential(
    [
        keras.Input(shape=(10,10,1),name="inputLayer"),
        layers.ZeroPadding2D(padding=((1, 0),(1, 0)),input_shape=(10, 10, 1), data_
        layers.Conv2D(1, kernel_size=(2, 2), weights=w, padding="valid",name="c1Lay
        layers.ZeroPadding2D(padding=((1, 0),(1, 0)), data_format="channels_last",n
        layers.Conv2D(1, kernel_size=(2, 2), weights=w, padding="valid",name="c2Lay
        layers.ZeroPadding2D(padding=((1, 0),(1, 0)), data_format="channels_last",n
        layers.Conv2D(1, kernel_size=(2, 2), weights=w, padding="valid",name="c3Lay
        layers.ZeroPadding2D(padding=((1, 0),(1, 0)), data_format="channels_last",n
        layers.Conv2D(1, kernel_size=(2, 2), weights=w, padding="valid",name="c4Lay
        layers.ZeroPadding2D(padding=((1, 0),(1, 0)), data_format="channels_last",n
        layers.Conv2D(1, kernel_size=(2, 2), weights=w, padding="valid",name="c5Lay


    ]
```

```
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 p1Layer (ZeroPadding2D)     (None, 11, 11, 1)         0

 c1Layer (Conv2D)            (None, 10, 10, 1)         5

 p2Layer (ZeroPadding2D)     (None, 11, 11, 1)         0

 c2Layer (Conv2D)            (None, 10, 10, 1)         5

 p3Layer (ZeroPadding2D)     (None, 11, 11, 1)         0

 c3Layer (Conv2D)            (None, 10, 10, 1)         5

 p4Layer (ZeroPadding2D)     (None, 11, 11, 1)         0

 c4Layer (Conv2D)            (None, 10, 10, 1)         5

 p5Layer (ZeroPadding2D)     (None, 11, 11, 1)         0

 c5Layer (Conv2D)            (None, 10, 10, 1)         5

=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

In [6]:
```
# idea only. This is what MATLAB padding said (np
from tensorflow import pad
paddings = [[1, 0,], [1, 0]]
tensor = [[1,2],[1,2]]
pad(tensor, paddings, mode='CONSTANT', constant_values=0, name=None)
# this could have been used with funcitonalAPI and calling layer by layer.(x) type
# I think zeropad2dLayer is fine now I've realised it exists :)
# https://keras.io/api/layers/reshaping_layers/zero_padding2d/?adlt=strict
# recall it wanted a tuple for differing top,bottom,left and right
```

Out[6]:
```
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[0, 0, 0],
       [0, 1, 2],
       [0, 1, 2]])>
```

In [7]:
```
inputData = np.zeros((10,10,1))
inputData[0,0,0] = 1
inputData = inputData[np.newaxis,:,:,:]
p = model.predict(inputData)
```

```
1/1 [==============================] - 0s 77ms/step
```

In [8]:
```
print(np.squeeze(p))
np.shape(p)
```

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  5. 10. 10.  5.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Out[8]:
```
(1, 10, 10, 1)
```

In [9]:
```
layer_outputs = [layer.output for layer in model.layers[:10:2]]

# Extracts the outputs of the top 5 layers.
activation_model = models.Model(inputs=model.input, outputs=layer_outputs) # Create
```

In [10]:
```
activations = activation_model.predict(inputData)
# Returns a list of five Numpy arrays: one array per layer activation
```

```
1/1 [==============================] - 0s 44ms/step
```

In [11]:
```
len(activations)
for a in activations:
    print(np.squeeze(a))
    np.shape(a)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 2. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 3. 3. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 4. 6. 4. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: