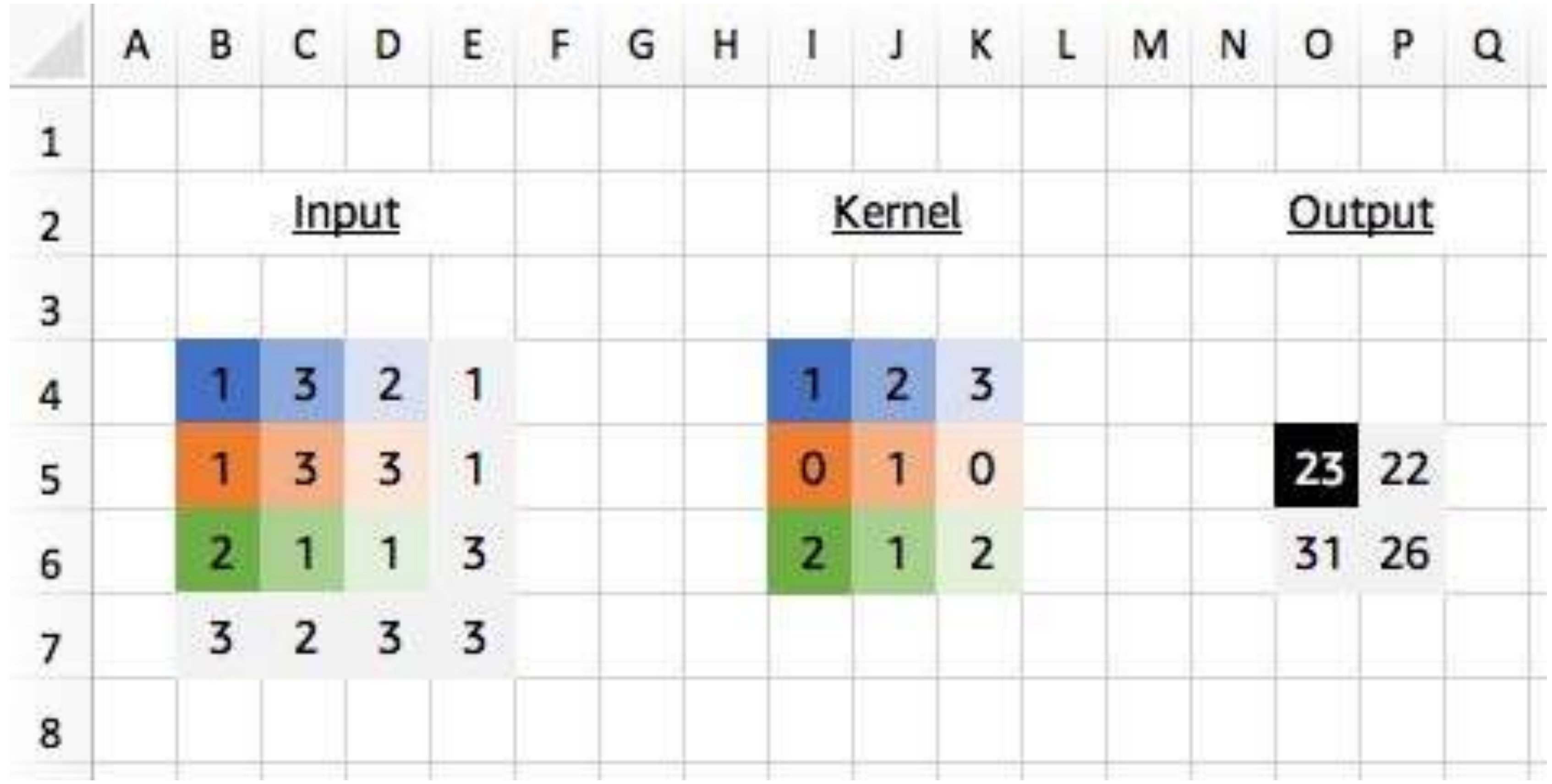
NUS-ISS
Problem Solving Using Pattern Recognition

Convolutional neural network

by Dr. Tan Jen Hong

2D convolution
The original



Source: https://medium.com/apache-mxnet/convolutions-explained-with-ms-excel-465d6649831c

2D convolution

The padded

Input

0	0	0	0	0	0
0	1	3	2	1	0
0	1	3	3	1	0
0	2	1	1	3	0
0	3	2	3	3	0
0	0	0	0	0	0

Kernel

1	2	3
0	1	0
2	1	2

Output

8	14	13	8
16	23	22	10
20	31	26	17
10	9	15	10

Source: https://medium.com/apache-mxnet/convolutions-explained-with-ms-excel-465d6649831c

2D convolution

Multi-channel

Input

1	0	1	0	2
1	1	3	2	1
1	1	0	1	1
2	3	2	1	3
0	2	0	1	0

1	0	0	1	0
2	0	1	2	0
3	1	1	3	0
0	3	0	3	2
1	0	3	2	1

2	0	1	2	1
3	3	1	3	2
2	1	1	1	0
3	1	3	2	0
1	1	2	1	1

Kernel

0	1	0
0	0	2
0	1	0

2	1	0
0	0	0
0	3	0

1	0	0
1	0	0
0	0	2

Intermediate Output

7	5	3
4	7	5
7	2	8

5	3	10
13	1	13
7	12	11

7	5	2
11	8	2
9	4	6

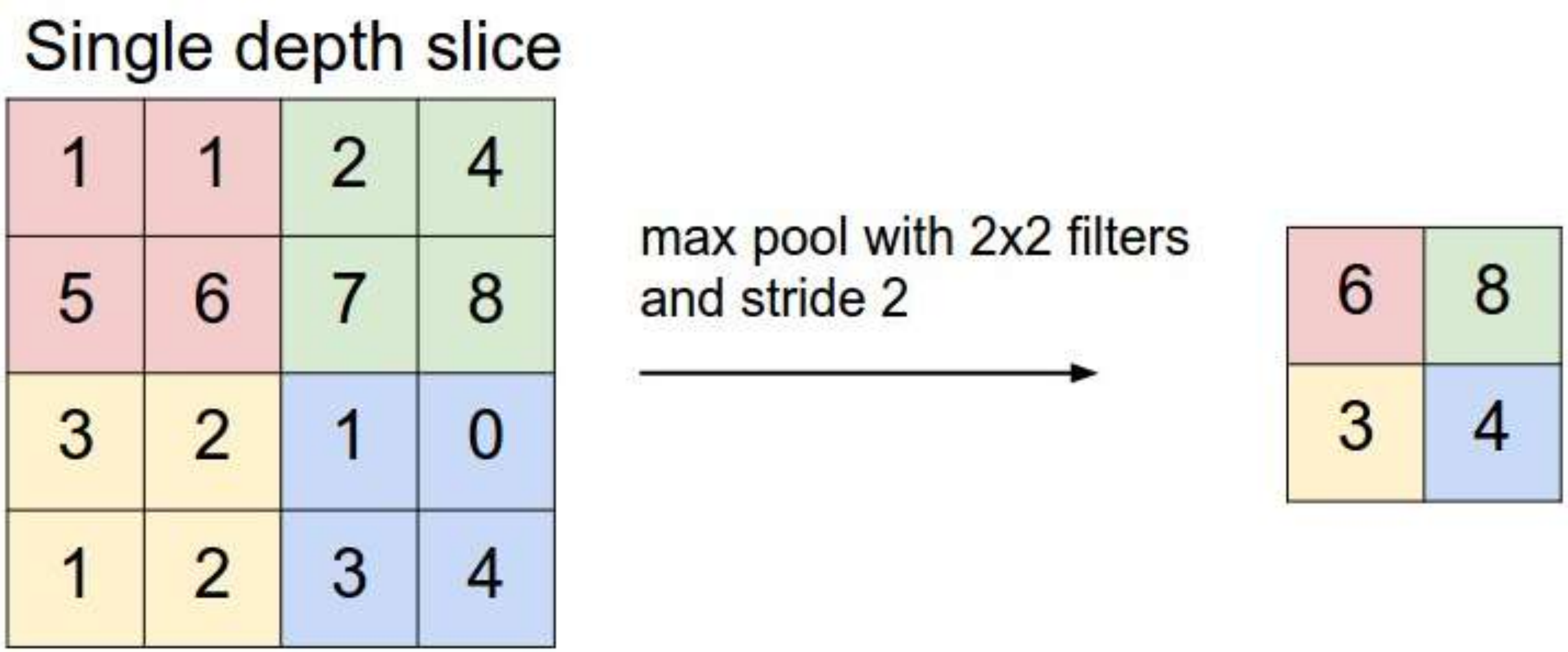
Output

19	13	15
28	16	20
23	18	25

Source: https://medium.com/apache-mxnet/convolutions-explained-with-ms-excel-465d6649831c

Max pooling

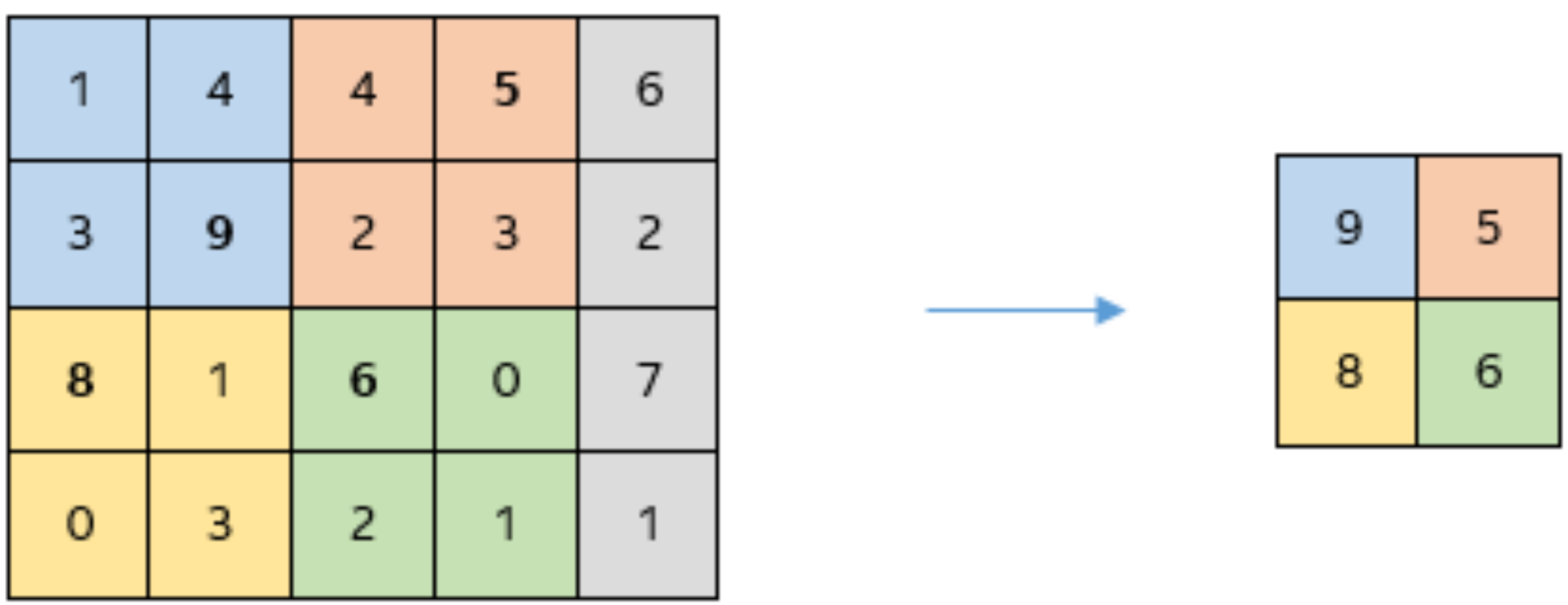
The original



Source: http://cs231n.github.io/convolutional-networks/

Max pooling

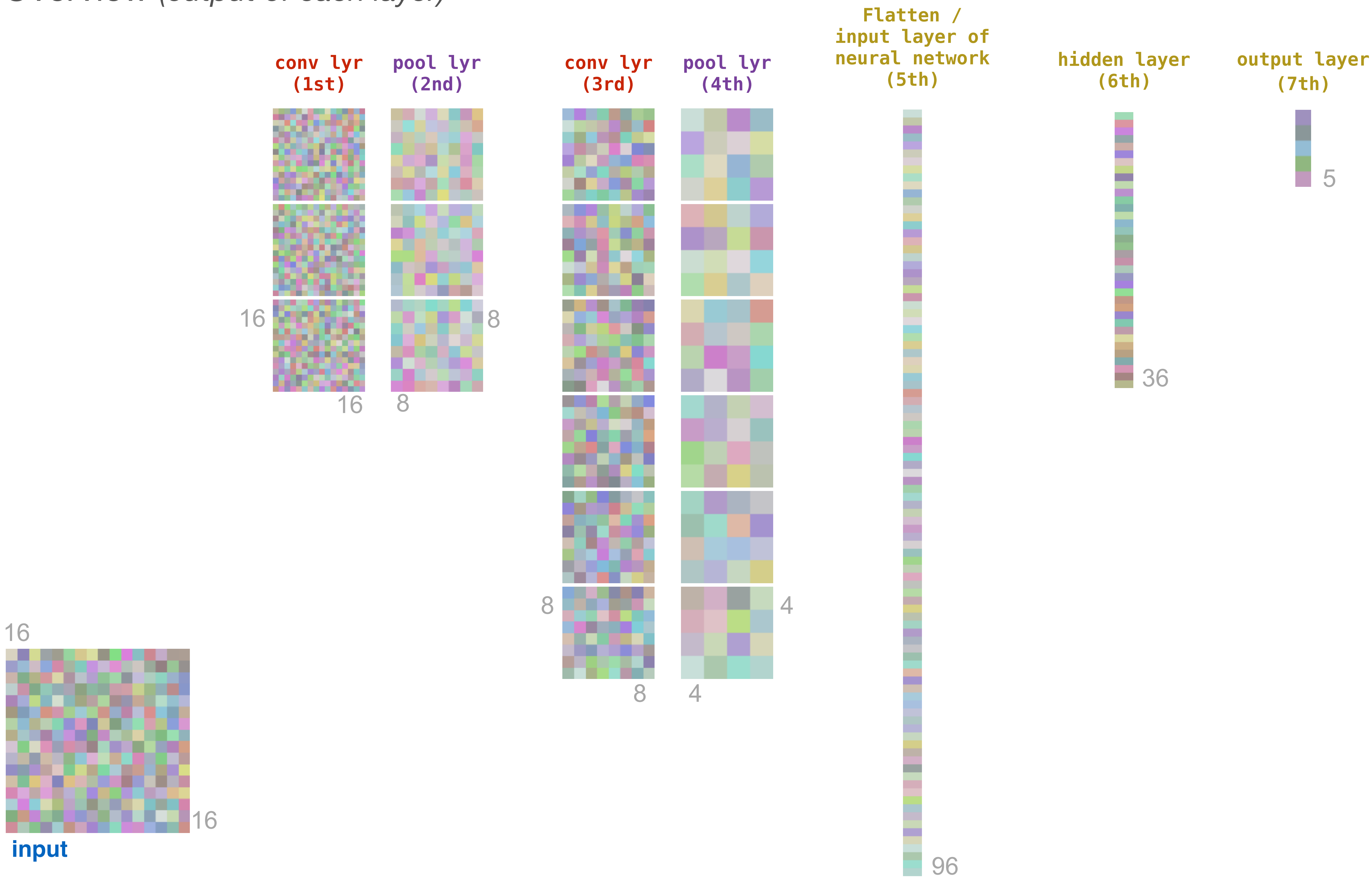
With situation



Source: https://software.intel.com/en-us/daal-programming-guide-2d-max-pooling-forward-layer

Convolutional neural network

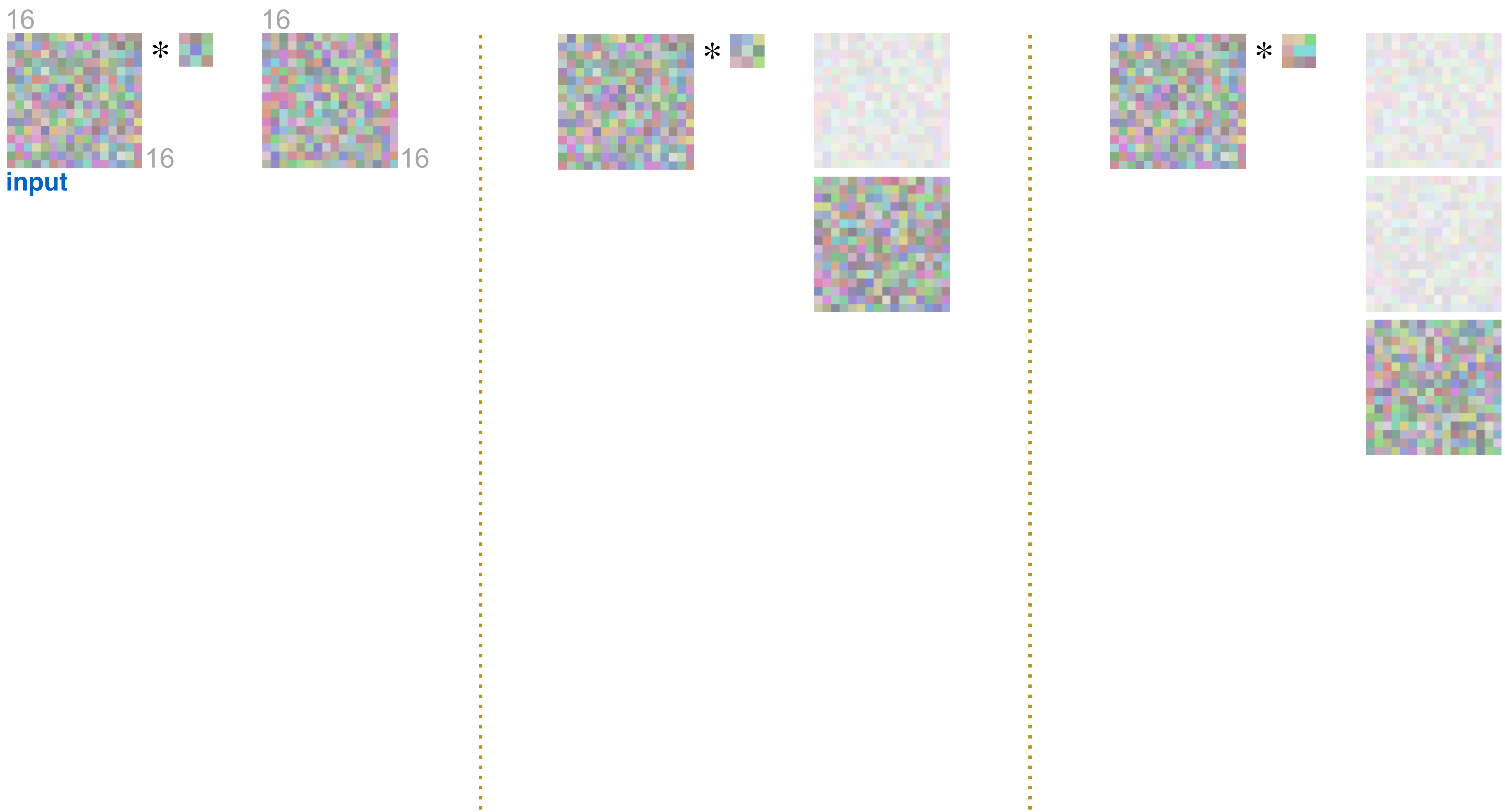
Overview *(output of each layer)*



The making of ...

The first convolutional layer (part 1)

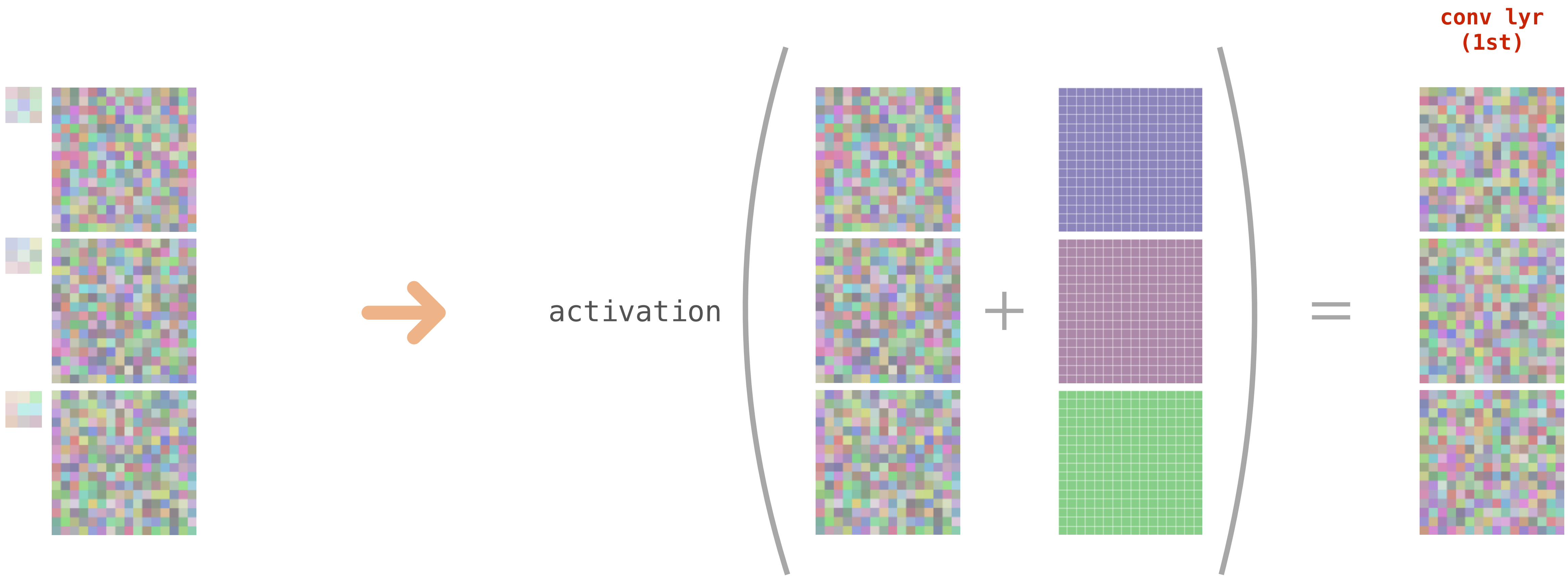
- Performs 3 separate 2D convolutions (with padding) to generate 3 intermediate outputs



The making of ...

The first convolutional layer (part 2)

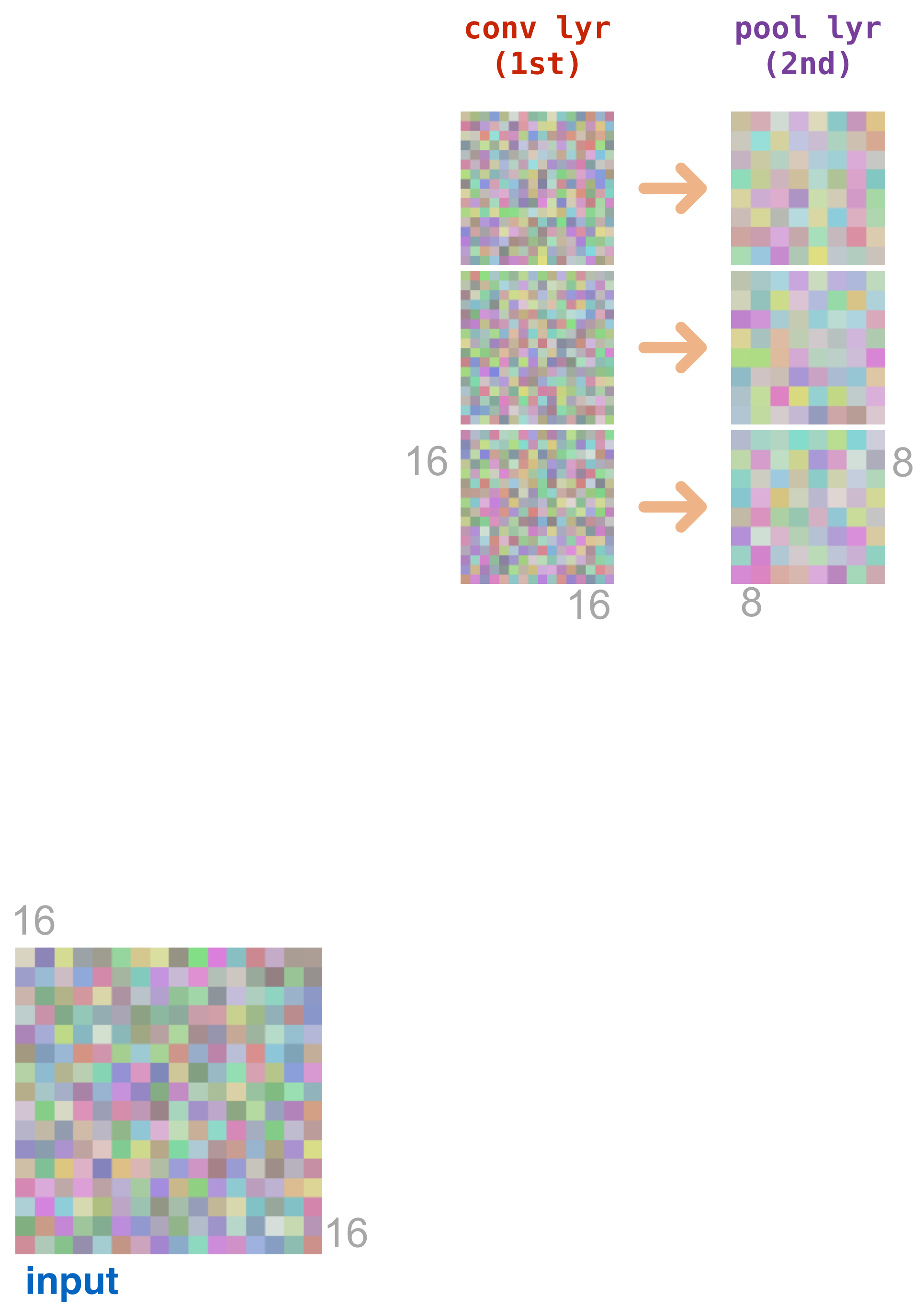
- Add bias to each convolution output, and apply activation function to get the final output for the convolutional layer



The making of ....

The pooling layer

- Apply 2 x 2 max-pooling (stride 2) on the outputs from the first convolutional layer



The making of ...

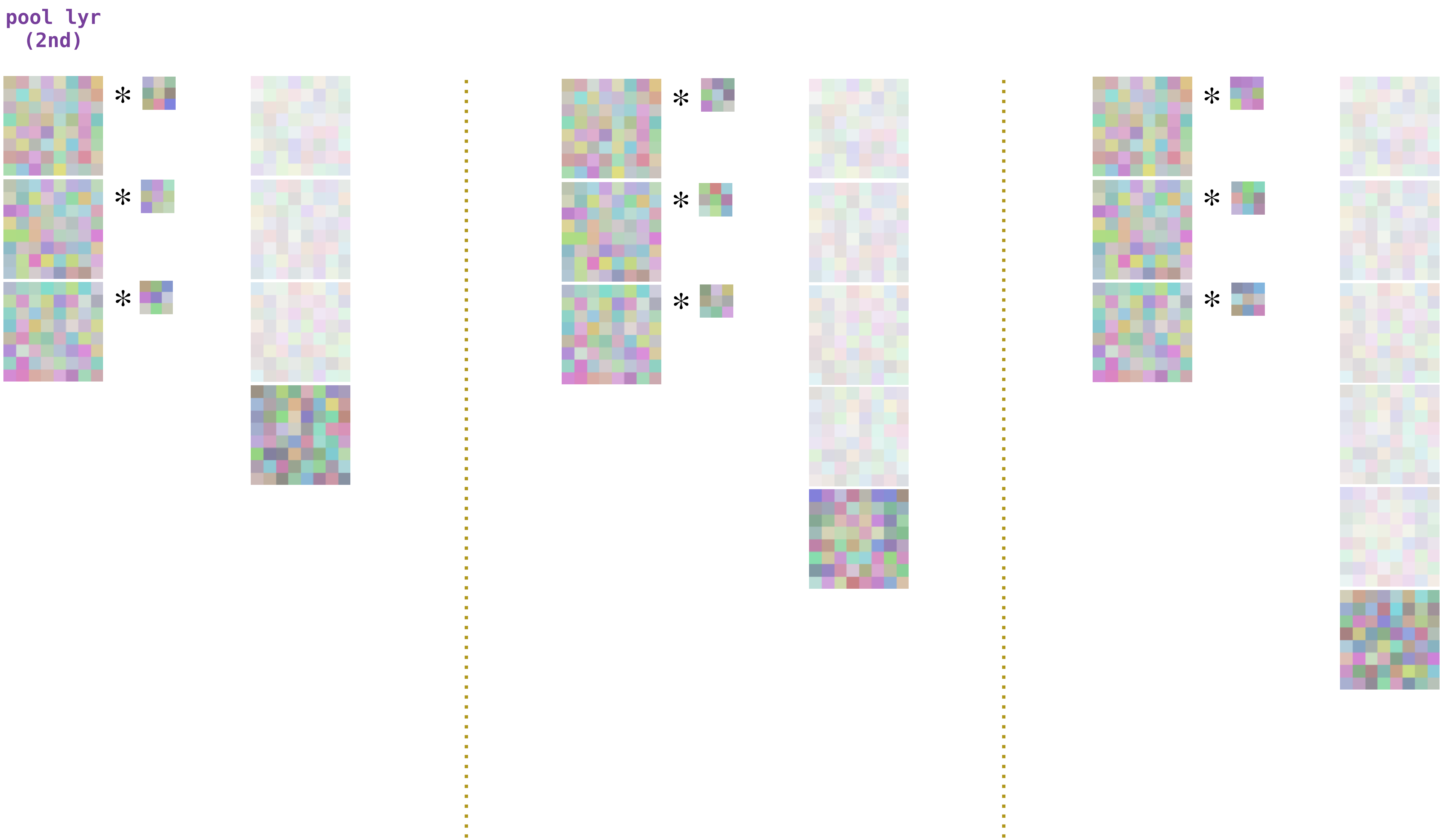
The second convolutional layer (part 1)

- Performs 6 separate multi-channel 2D convolutions (with padding) to generate 6 convolution outputs

The making of ...

The second convolutional layer (part 2)

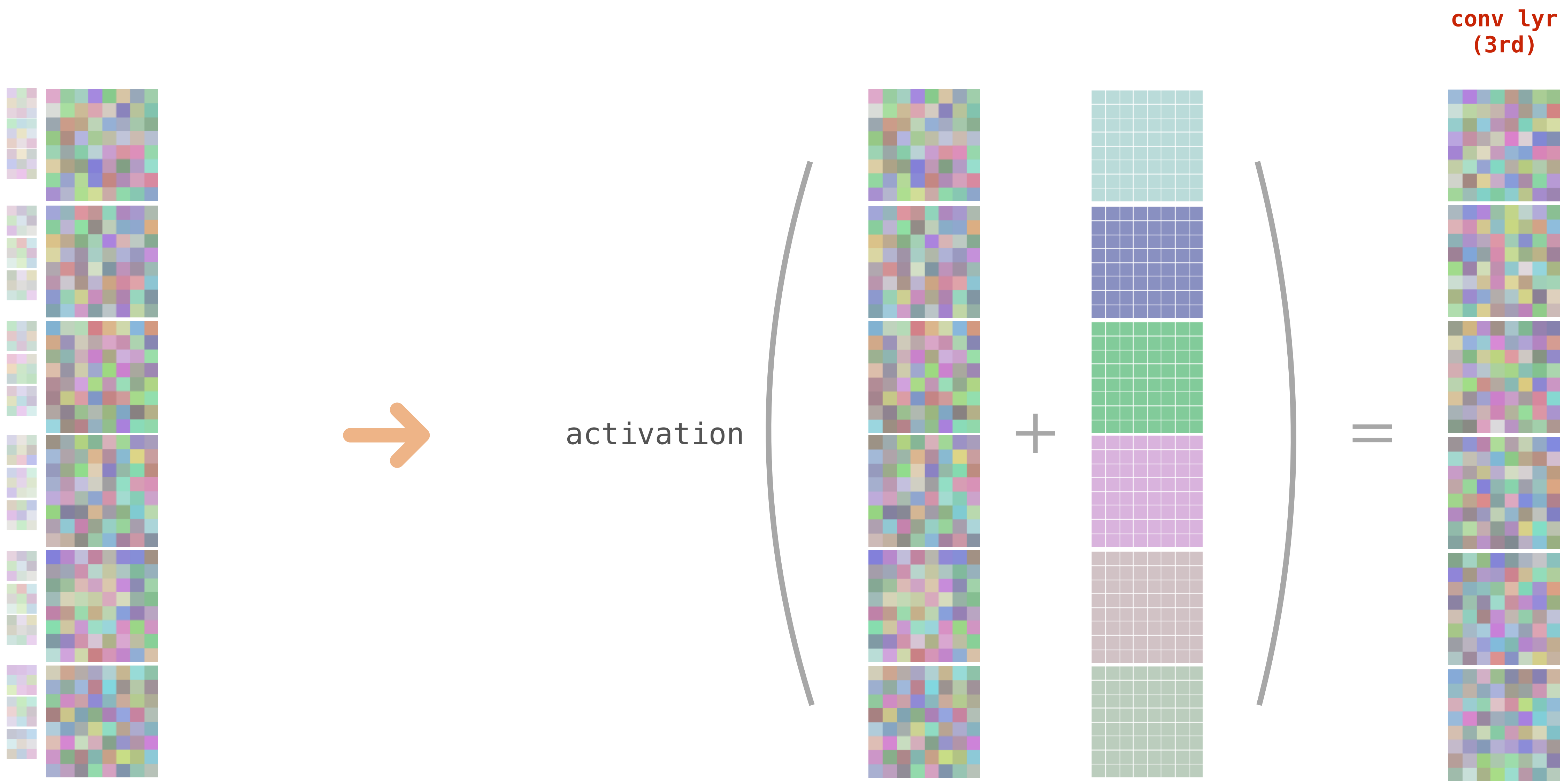
- Performs 6 separate multi-channel 2D convolutions (with padding) to generate 6 convolution outputs



The making of ...

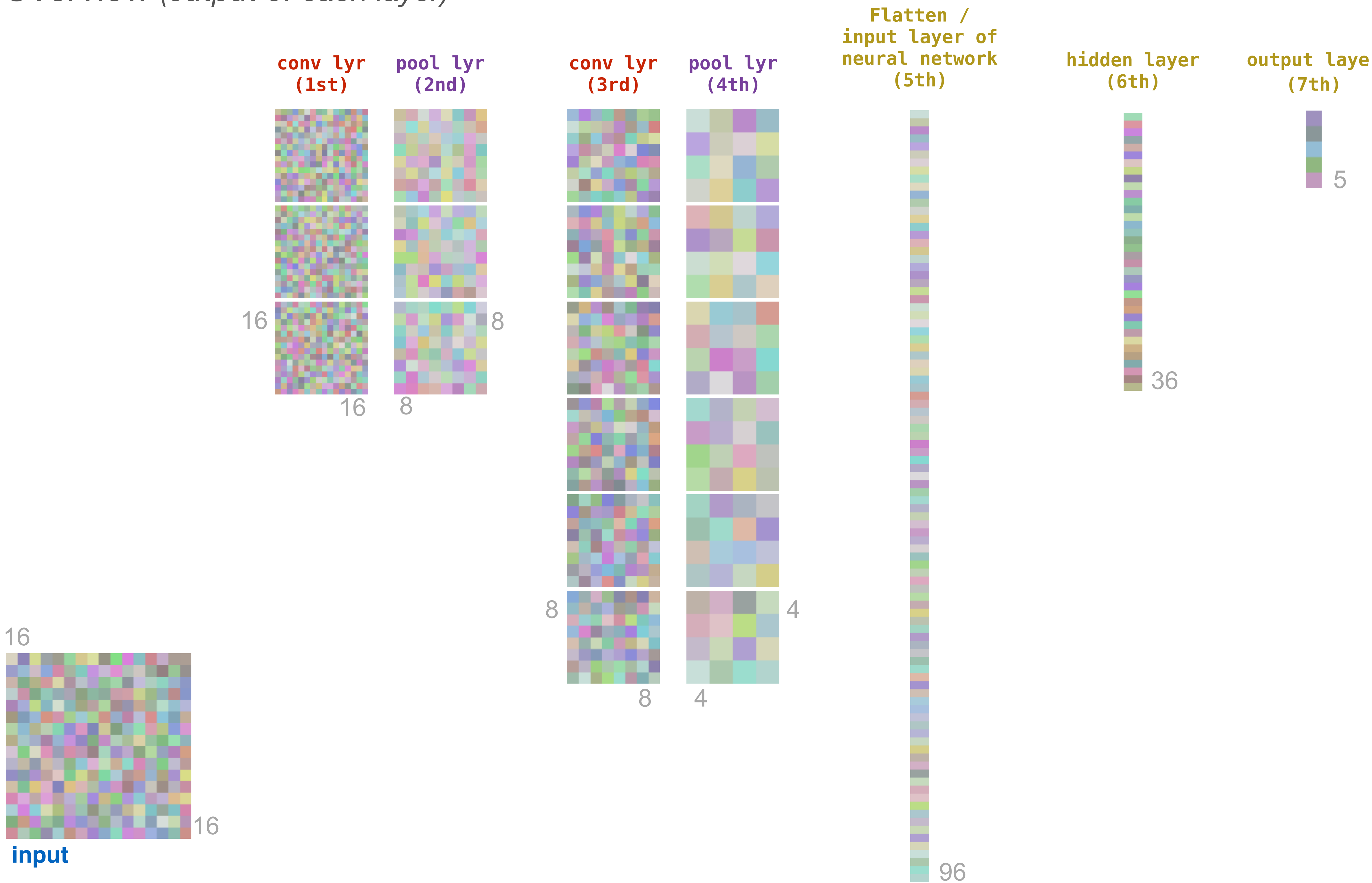
The first convolutional layer (part 3)

- Add bias to each intermediate output, and apply activation function to get the final output for the convolutional layer



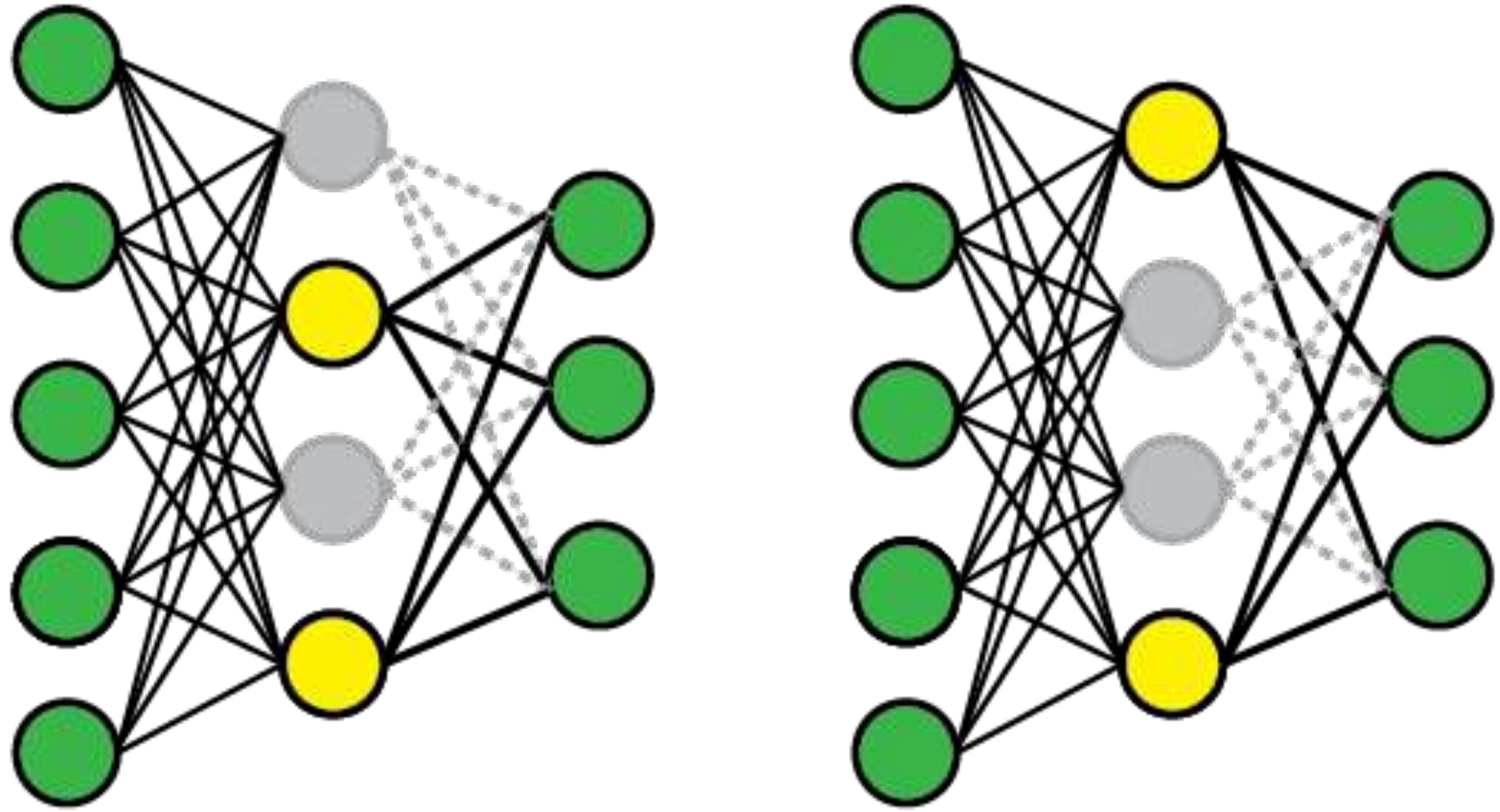
Convolutional neural network

Overview *(output of each layer)*



Convolutional neural network

Dropout



Source: https://stats.stackexchange.com/questions/201569/difference-between-dropout-and-dropconnect

Exercise

Suggest a use case of identification using machine learning method

Exercise

Suggest a use case of identification using machine learning method

- Make your proposal in powerpoint slides by answering the below questions:
- What is the problem to be solved? Why is the problem worth solving?
- Which supervised learning method are you going to use, and why did you choose that?
- What are the input(s) and output(s)? Specify the number of classes. Provide a few examples on the input and output (approximation is also OK)
- How do you plan to collect the data for your input and output, and what would you do in your data collection to assure robustness of your trained model
- How much data do you think is needed to train your model well?

Any fans of Japan?

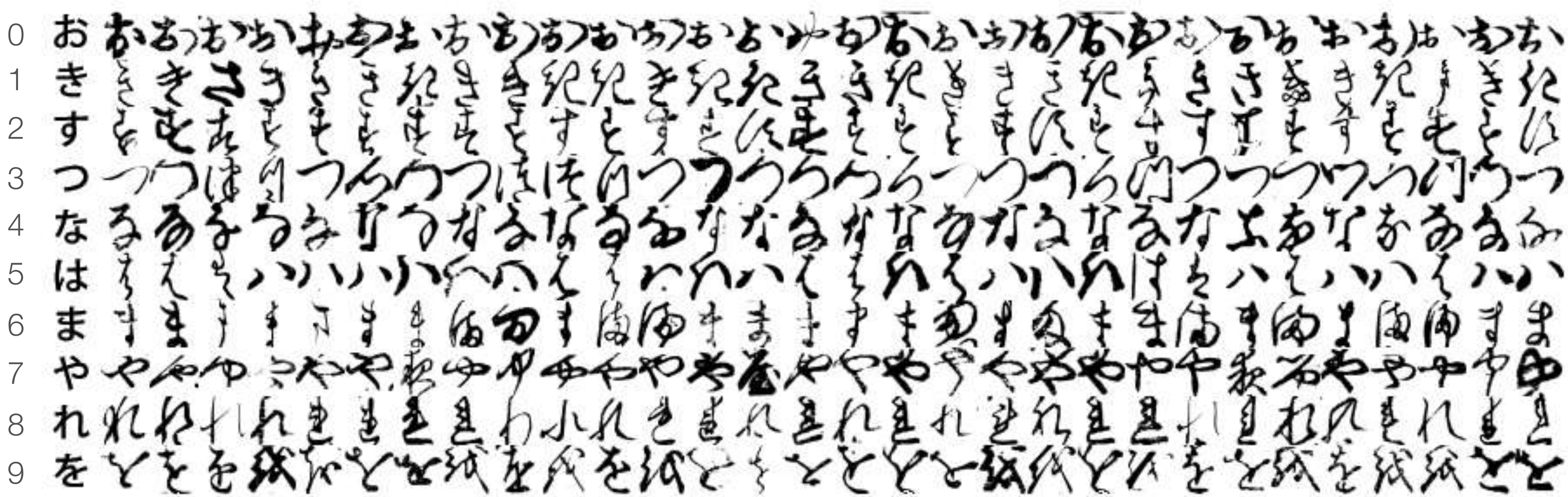
Cursive Kuzushiji

Automated solution?

Source: https://arxiv.org/pdf/1812.01718.pdf

Kuzushiji MNIST

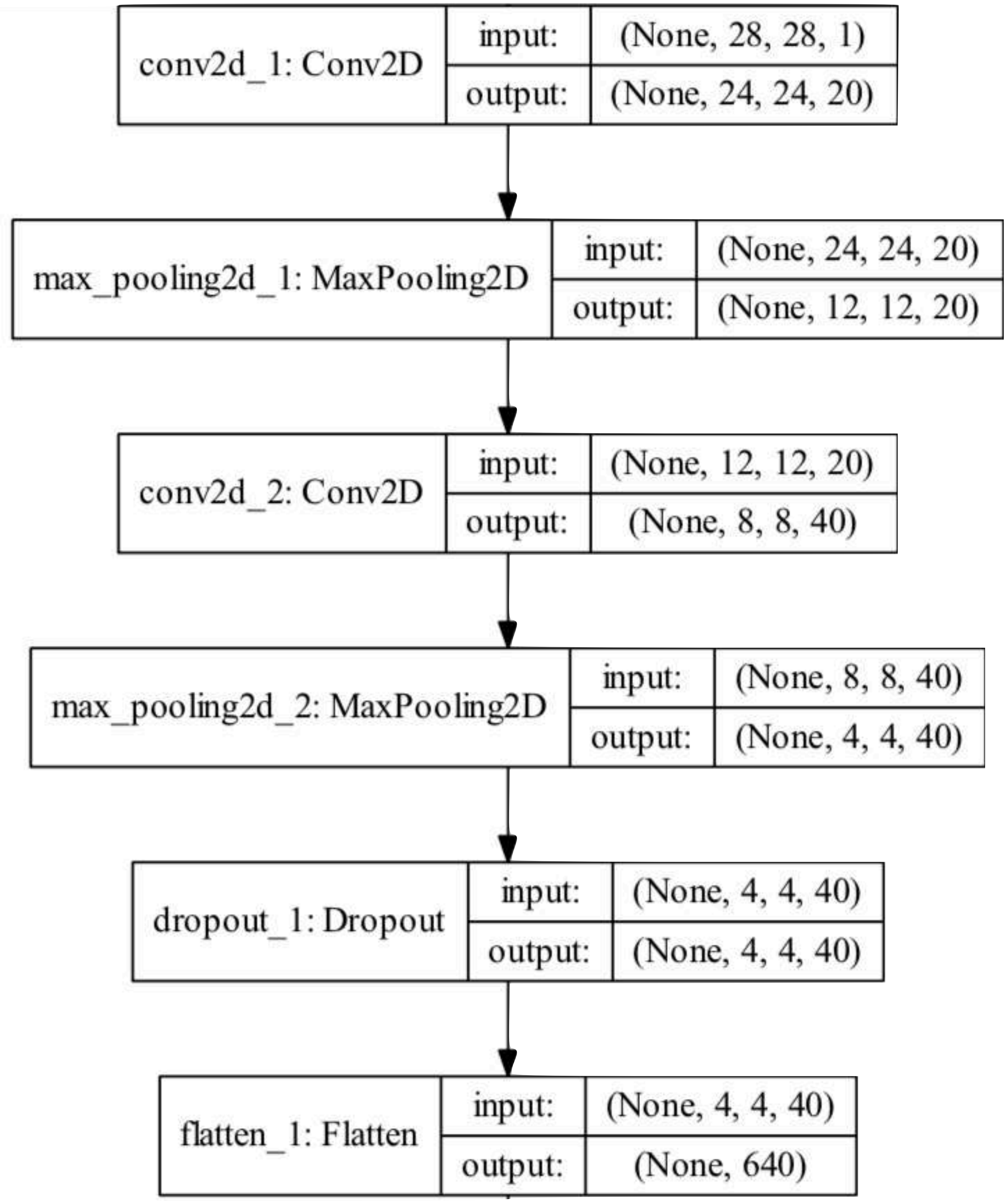
Another 'MNIST' alternative



Source: https://github.com/rois-codh/kmnist/blob/master/images/kmnist_examples.png

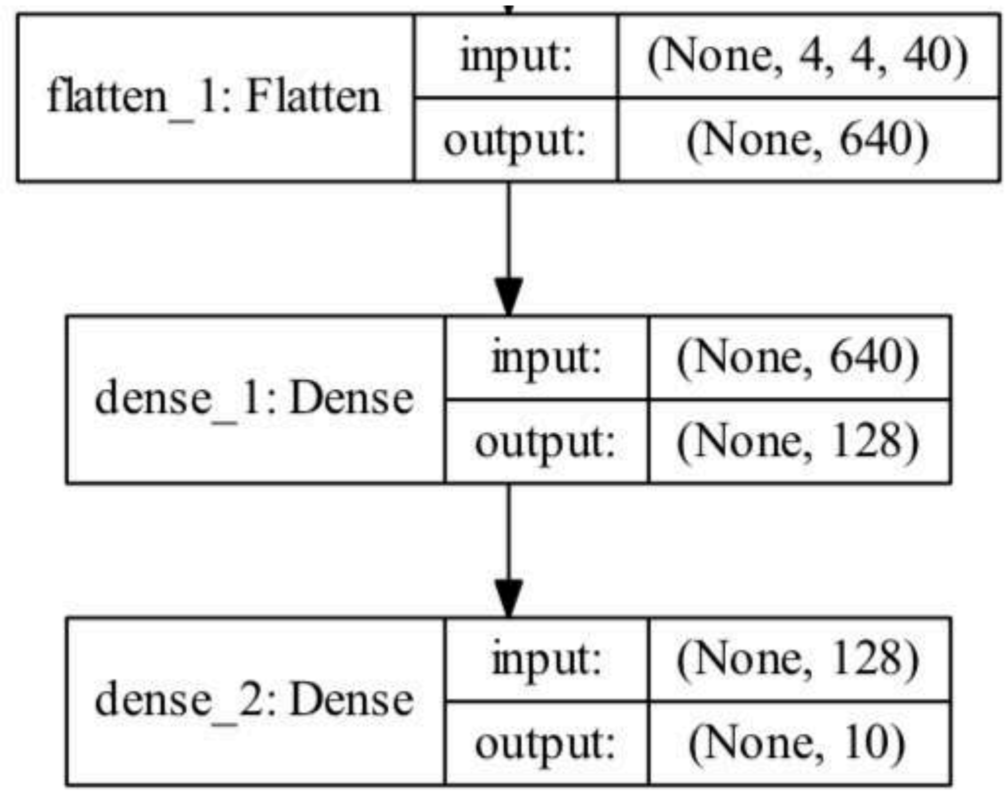
Kuzushiji MNIST

The basic model, part 1



Kuzushiji MNIST

The basic model, part 1



Kuzushiji MNIST

The main layout for the code

1. Import libraries
2. Matplotlib setup
3. Data preparation
4. Define model
5. Train model
6. Test model

Kuzushiji MNIST

1. Import libraries, part 1

- numpy for matrix manipulation; sklearn for measuring performance; matplotlib to show image and plot result; os for path manipulation

```
> import numpy as np
> import sklearn.metrics as metrics
> import matplotlib.pyplot as plt
> import os
```

Kuzushiji MNIST

1. Import libraries, part 2

- Import all the Keras functions that we are going to use in this problem

```
> from tensorflow.keras.callbacks import ModelCheckpoint,CSVLogger
> from tensorflow.keras.models import Sequential
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.layers import Dropout
> from tensorflow.keras.layers import Flatten
> from tensorflow.keras.layers import Conv2D
> from tensorflow.keras.layers import MaxPooling2D
> from tensorflow.keras.utils import to_categorical
```

Kuzushiji MNIST

2. Matplotlib setup, part 1

- First three lines setup the font manager, so that we can display Japanese words correctly in later usage
- Use 'ggplot' style to plot our training and testing result

```
> from matplotlib import font_manager as fm
> fpath       = os.path.join(os.getcwd(), "ipam.ttf")
> prop        = fm.FontProperties(fname=fpath)
```

```
> plt.style.use('ggplot')
> plt.rcParams['ytick.right']     = True
> plt.rcParams['ytick.labelright']= True
> plt.rcParams['ytick.left']      = False
> plt.rcParams['ytick.labelleft'] = False
> plt.rcParams['font.family']     = 'Arial'
```

Kuzushiji MNIST

2. Matplotlib setup, part 2

- Create a function that can display gray scale image correctly

```
> def grayplt(img,title=''):
      plt.axis('off')
      if np.size(img.shape) == 3:
         plt.imshow(img[:,:,0],cmap='gray',vmin=0,vmax=1)
      else:
         plt.imshow(img,cmap='gray',vmin=0,vmax=1)
      plt.title(title, fontproperties=prop)
      plt.show()
```

Kuzushiji MNIST

3. Data preparation, part 1

- Load train and test data; load train and test labels
- Rescale data to float, range from 0 to 1

```
> trDat       = np.load('kmnist-train-imgs.npz')['arr_0']
> trLbl       = np.load('kmnist-train-labels.npz')['arr_0']
> tsDat       = np.load('kmnist-test-imgs.npz')['arr_0']
> tsLbl       = np.load('kmnist-test-labels.npz')['arr_0']

> trDat       = trDat.astype('float32')/255
> tsDat       = tsDat.astype('float32')/255

> imgrows     = trDat.shape[1]
> imgclms     = trDat.shape[2]
```

Kuzushiji MNIST
3. Data preparation, part 2

- The current shape for trDat is `(60000, 28, 28)`
- The current shape for tsDat is `(10000, 28, 28)`
- Need to be reshaped into the form of (samples, width, height, channel)

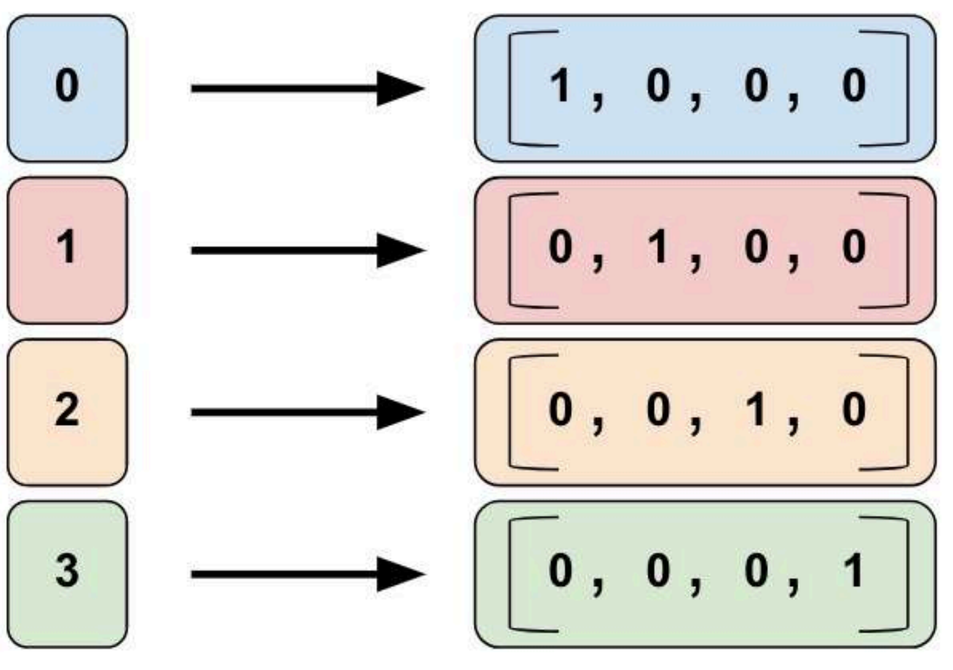
```
> trDat       = trDat.reshape(trDat.shape[0],
                               imgrows,
                               imgclms,
                               1)
> tsDat       = tsDat.reshape(tsDat.shape[0],
                              imgrows,
                              imgclms,
                              1)
```

Kuzushiji MNIST

3. Data preparation, part 3

- One-hot encode the train and test label information; get the number of classes in the labels

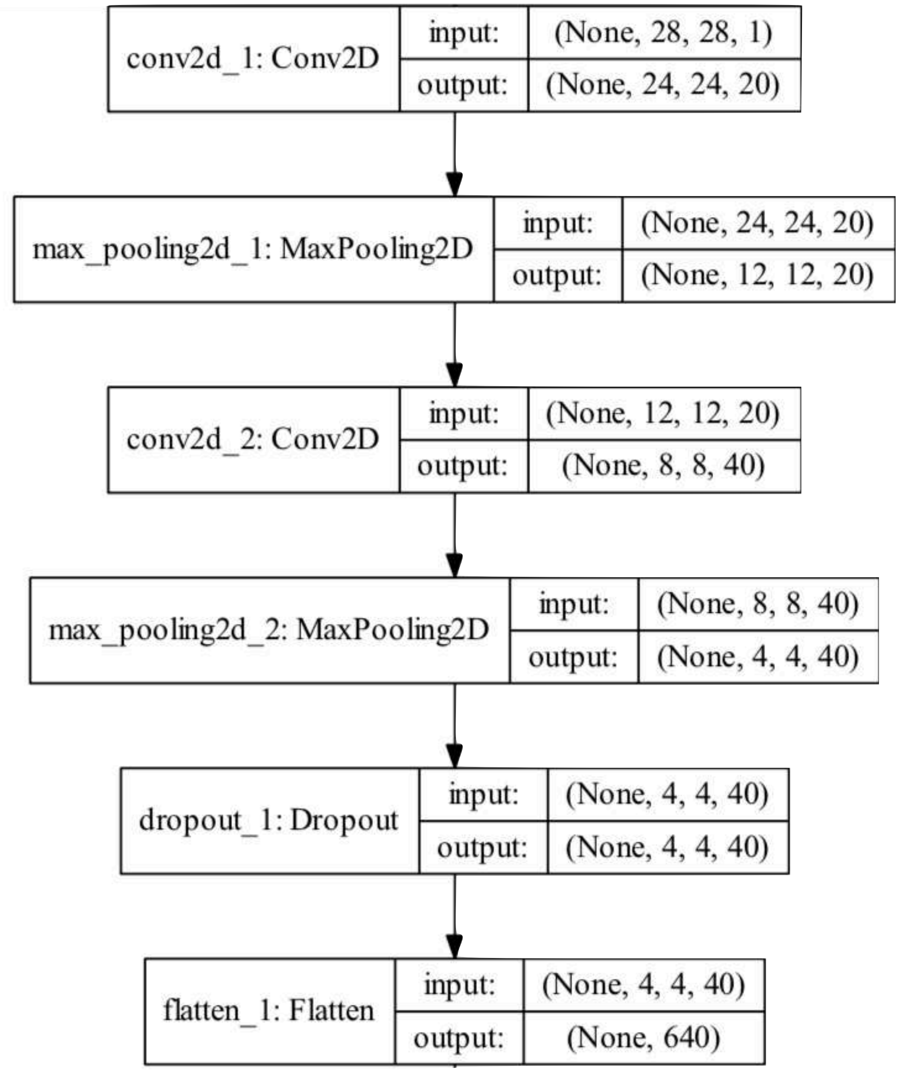
```
> trLbl       = to_categorical(trLbl)
> tsLbl       = to_categorical(tsLbl)
> num_classes = tsLbl.shape[1]
```



Source: https://arxiv.org/pdf/1812.01718.pdf

Kuzushiji MNIST

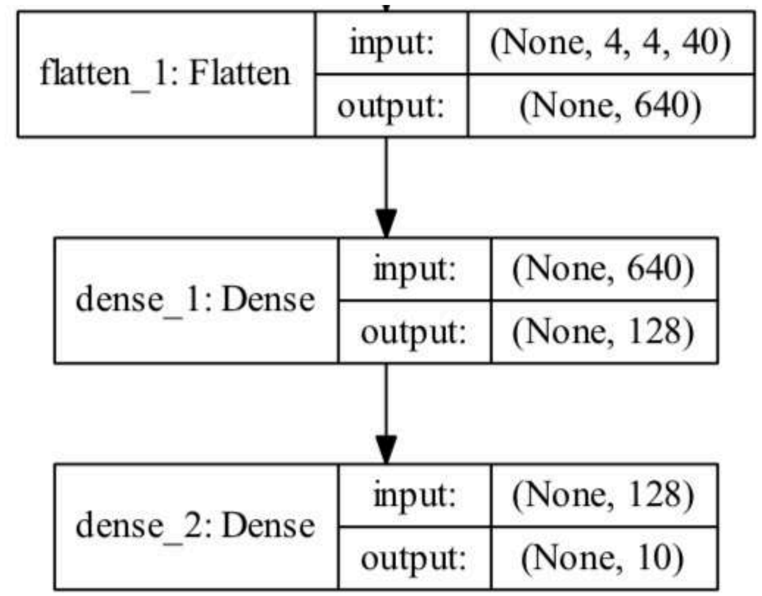
4. Define model, part 1




```
> seed          = 29
> np.random.seed(seed)

> modelname    = 'wks5_1a'
> def createModel():
      model = Sequential()
      model.add(Conv2D(20, (5, 5), input_shape=(28, 28, 1), activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Conv2D(40, (5, 5), activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Dropout(0.2))
      model.add(Flatten())
      model.add(Dense(128, activation='relu'))
      .....
```

Kuzushiji MNIST

4. Define model, part 1




```
> seed          = 29
> np.random.seed(seed)

> modelname     = 'wks5_1a'
> def createModel():
      model = Sequential()
      model.add(Conv2D(20, (5, 5), input_shape=(28, 28, 1), activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Conv2D(40, (5, 5), activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Dropout(0.2))
      model.add(Flatten())
      model.add(Dense(128, activation='relu'))
      model.add(Dense(num_classes, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer='adam',
                    metrics=['accuracy'])
      return model
```

Kuzushiji MNIST

4. Define model, part 2

- 'model' for training; 'modelGo' for final evaluation

```
> model        = createModel()
> modelGo      = createModel()

> model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 20)        520
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 20)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 40)          20040
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 40)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 40)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 640)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               82048
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 103,898
Trainable params: 103,898
Non-trainable params: 0
_________________________________________________________________
```

Kuzushiji MNIST
4. Define model, part 3

- Create checkpoints to save model during training and save training data into csv

```
> filepath        = modelname + ".hdf5"
> checkpoint      = ModelCheckpoint(filepath,
                                    monitor='val_acc',
                                    verbose=0,
                                    save_best_only=True,
                                    mode='max')


> csv_logger      = CSVLogger(modelname + '.csv')
> callbacks_list  = [checkpoint,csv_logger]
```

Kuzushiji MNIST

5. Train model

- Training is only a single line

```
> model.fit(trDat,
            trLbl,
            validation_data=(tsDat, tsLbl),
            epochs=60,
            batch_size=128,
            callbacks=callbacks_list)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/60
60000/60000 [==============================] - 68s 1ms/sample - loss: 0.4707 - acc: 0.8539 - val_loss: 0.4163 - val_acc: 0.8689
Epoch 2/60
60000/60000 [==============================] - 66s 1ms/sample - loss: 0.1603 - acc: 0.9509 - val_loss: 0.3003 - val_acc: 0.9125
Epoch 3/60
60000/60000 [==============================] - 66s 1ms/sample - loss: 0.1068 - acc: 0.9673 - val_loss: 0.2459 - val_acc: 0.9290
Epoch 4/60
60000/60000 [==============================] - 65s 1ms/sample - loss: 0.0798 - acc: 0.9751 - val_loss: 0.2348 - val_acc: 0.9352
Epoch 5/60
60000/60000 [==============================] - 65s 1ms/sample - loss: 0.0653 - acc: 0.9794 - val_loss: 0.2254 - val_acc: 0.9406
......
```

Kuzushiji MNIST

6. Test model, part 1

- Use a new object to load the weights, and check the best accuracy

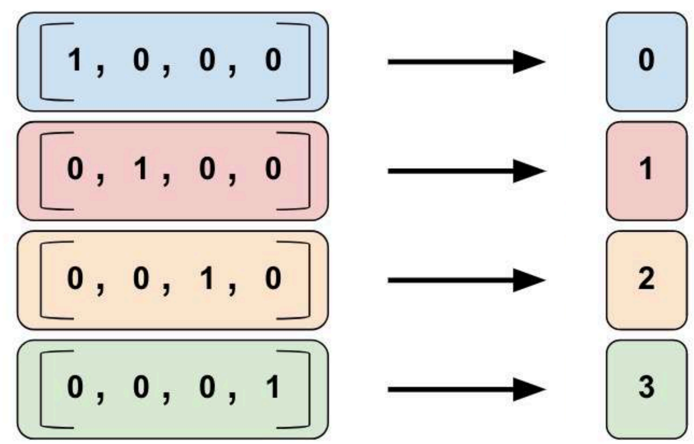
```
> modelGo.load_weights(filepath)
> modelGo.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

Kuzushiji MNIST

6. Test model, part 2

- Test the model, calculate the accuracy and confusion matrix

```
> predicts      = modelGo.predict(tsDat)
> predout       = np.argmax(predicts,
                            axis=1)
```




```
> testout       = np.argmax(tsLbl,axis=1)

> labelname     = ['お O','き Ki','す Su','つ Tsu','な Na',
                   'は Ha','ま Ma','や Ya','れ Re','を Wo']

> testScores    = metrics.accuracy_score(testout,predout)
> confusion     = metrics.confusion_matrix(testout,predout)
```

Kuzushiji MNIST

6. Test model, part 3

- Test the model, calculate the accuracy and confusion matrix

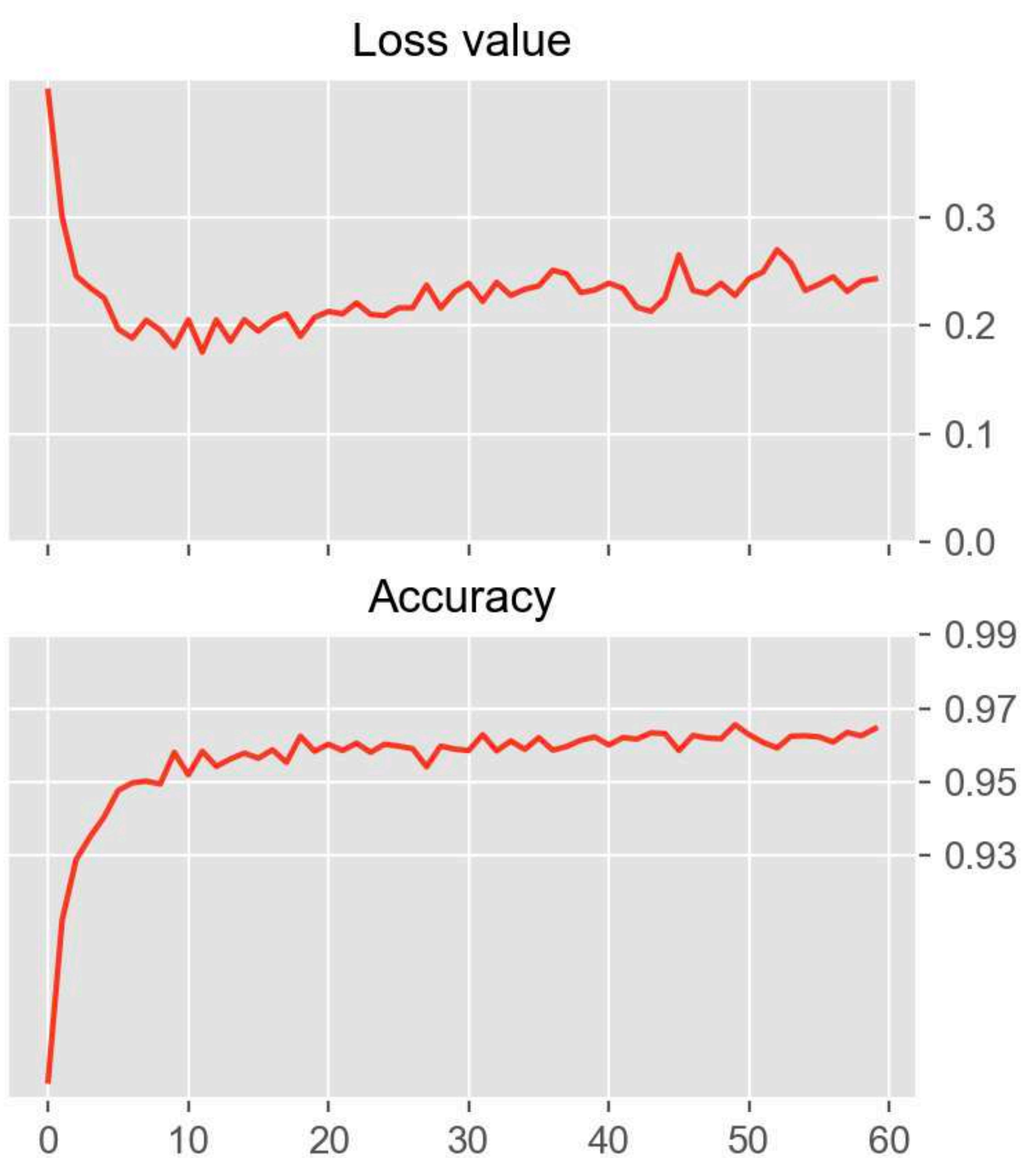
```
> print("Best accuracy (on testing dataset): %.2f%%" % (testScores*100))
> print(metrics.classification_report(testout,predout,target_names=labelname,digits=4))
> print(confusion)
```

```
Best accuracy (on testing dataset): 96.56%
             precision    recall  f1-score   support

       お O     0.9615    0.9740    0.9677      1000
      き Ki     0.9772    0.9430    0.9598      1000
      す Su     0.9562    0.9390    0.9475      1000
     つ Tsu     0.9732    0.9820    0.9776      1000
      な Na     0.9588    0.9530    0.9559      1000
      は Ha     0.9707    0.9600    0.9653      1000
      ま Ma     0.9245    0.9920    0.9571      1000
      や Ya     0.9877    0.9620    0.9747      1000
      れ Re     0.9665    0.9800    0.9732      1000
      を Wo     0.9838    0.9710    0.9774      1000

avg / total     0.9660    0.9656    0.9656     10000
```

```
[[974   2   1   1  18   1   0   1   1   1]
 [  5 943   6   0   5   2  24   3   7   5]
 [  8   3 939   9   4   7  19   4   7   0]
 [  0   0   8 982   0   4   5   0   1   0]
 [ 12   2   1   9 953   4   8   2   6   3]
 [  1   3  13   4   1 960  13   0   3   2]
 [  0   2   3   0   1   2 992   0   0   0]
 [  7   5   5   0   5   2   5 962   5   4]
 [  2   1   4   3   5   3   1   0 980   1]
 [  4   4   2   1   2   4   6   2   4 971]]
```

Kuzushiji MNIST

6. Test model, part 4



- Plot the result

```
> import pandas as pd

> records       = pd.read_csv(modelname +'.csv')
> plt.figure()
> plt.subplot(211)
> plt.plot(records['val_loss'])
> plt.yticks([0.00,0.10,0.20,0.30])
> plt.title('Loss value',fontsize=12)

> ax            = plt.gca()
> ax.set_xticklabels([])

> plt.subplot(212)
> plt.plot(records['val_acc'])
> plt.yticks([0.93,0.95,0.97,0.99])
> plt.title('Accuracy',fontsize=12)
> plt.show()
```