NUS-ISS

Problem Solving Using Pattern Recognition

Convolutional neural network

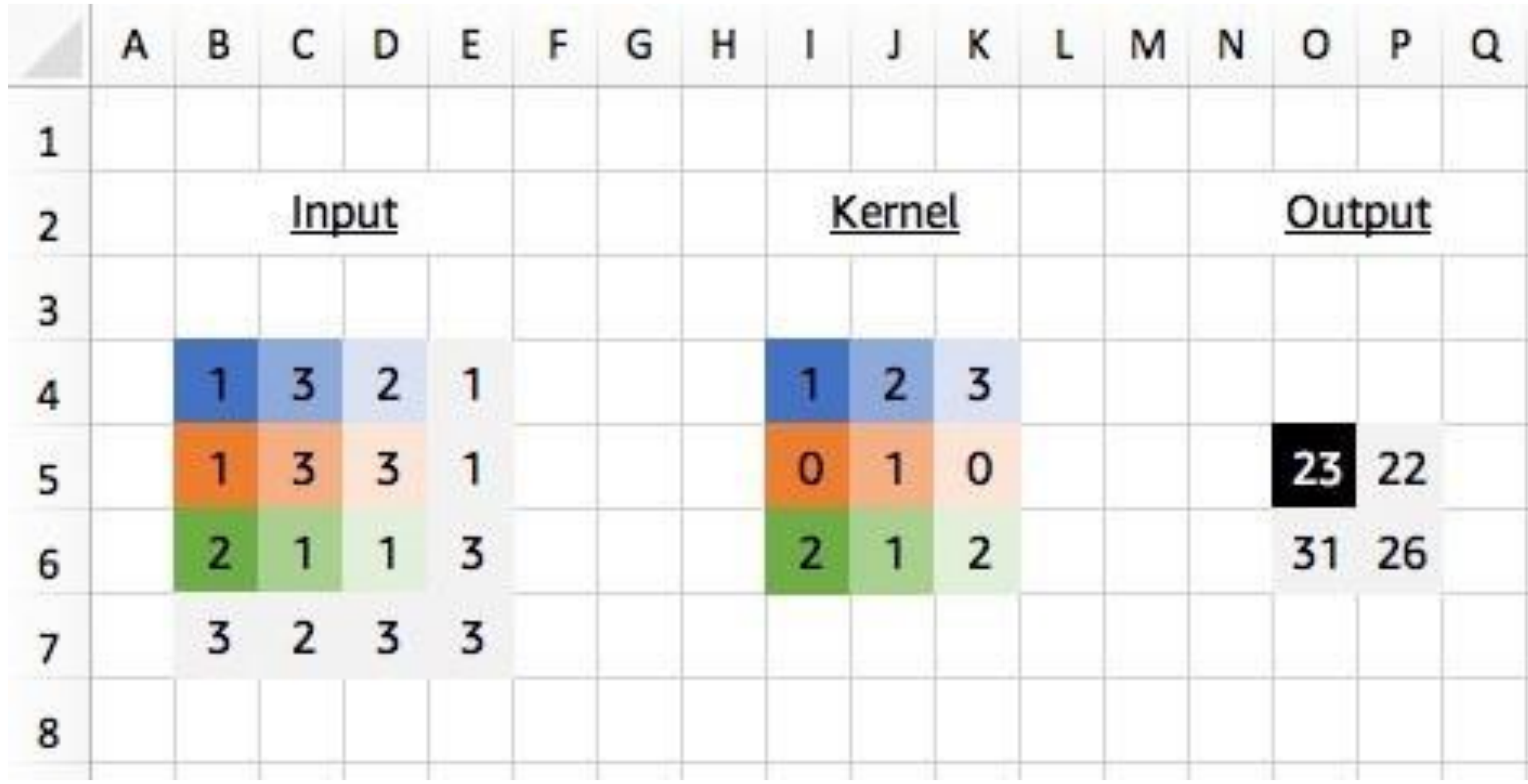
by Nicholas Ho

Recap:

Convolution = filtering

2D convolution

The original



Note:

- **Conv1D is used for input signals which are similar to the voice**
- **Conv2D is used for images**
- **Conv3D is usually used for videos where you have a frame for each time span**

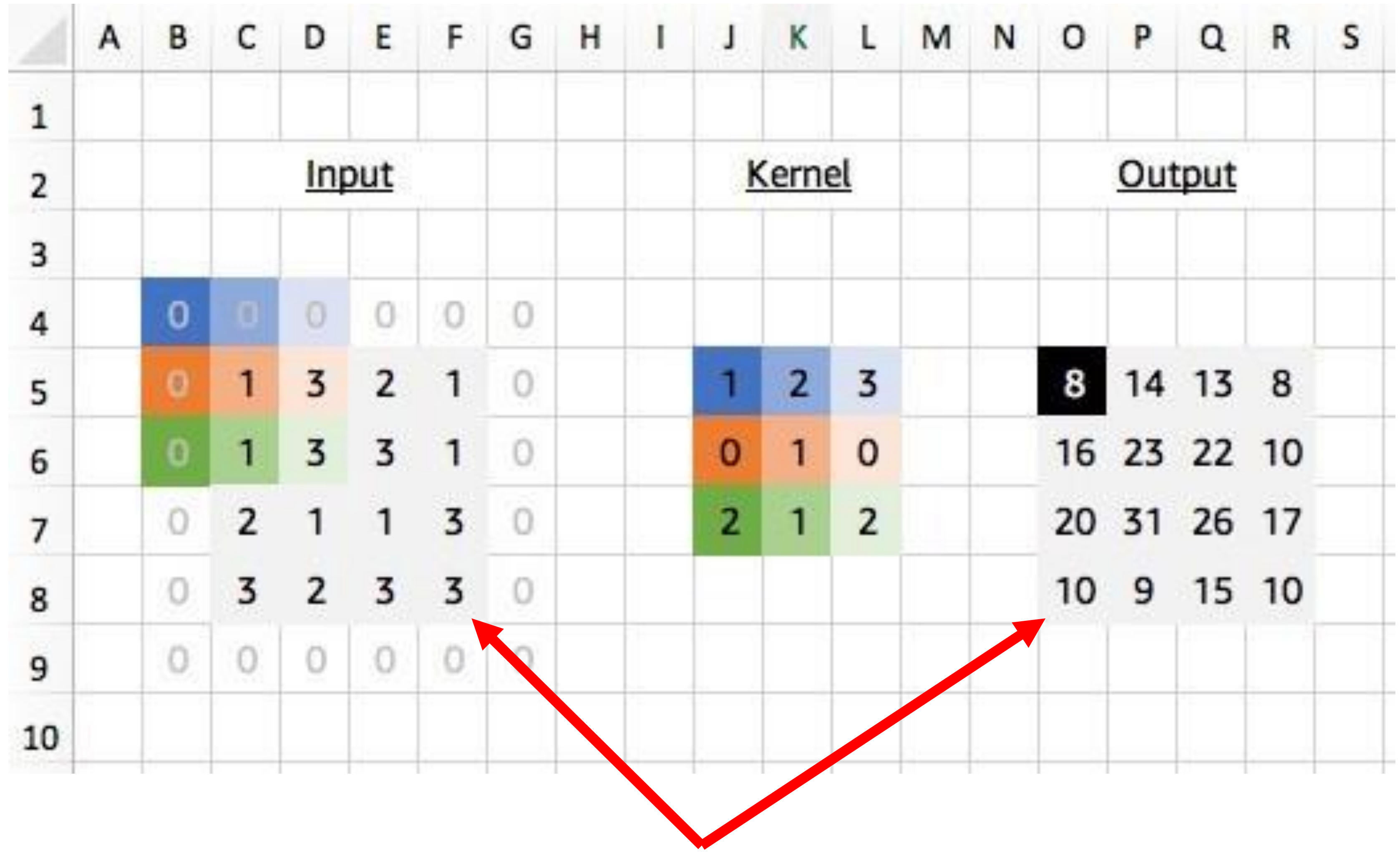
Source: https://medium.com/apache-mxnet/convolutions-explained-with-ms-excel-465d6649831c

2D convolution

The padded

Note that the Kernel's movement is determined by the stride value, which can be adjusted

(in the examples stride = 1 or [1, 1])

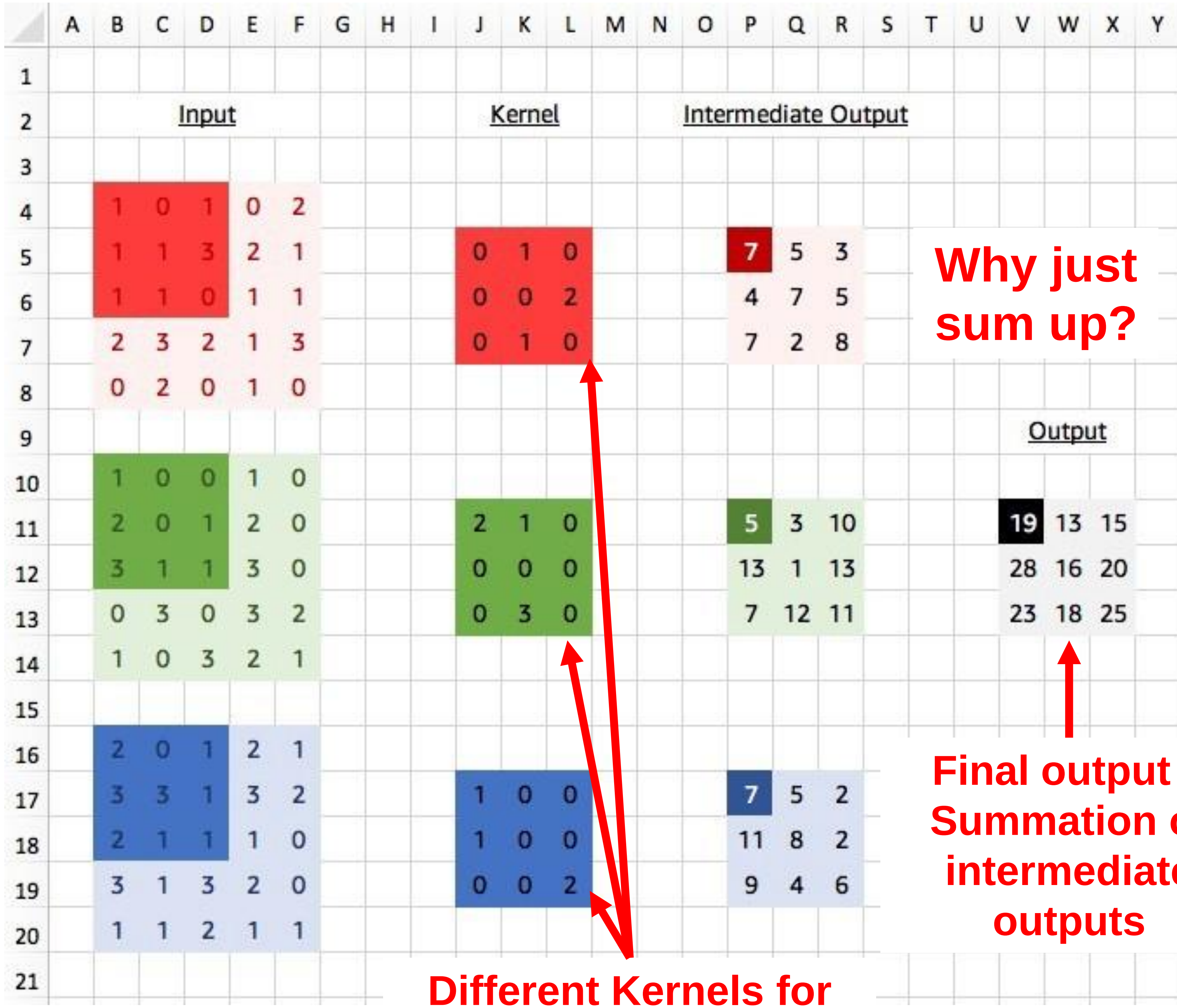


Padding retains the size of the output; input size = output size

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

2D convolution
Multi-channel

Note:
- **We do not decide the values of the Kernels**
- **The Kernel values are updated by the algorithms based on the training data**
- **We decide only the size of the Kernels**



Why just sum up?

Final output = Summation of intermediate outputs

Different Kernels for each input channel

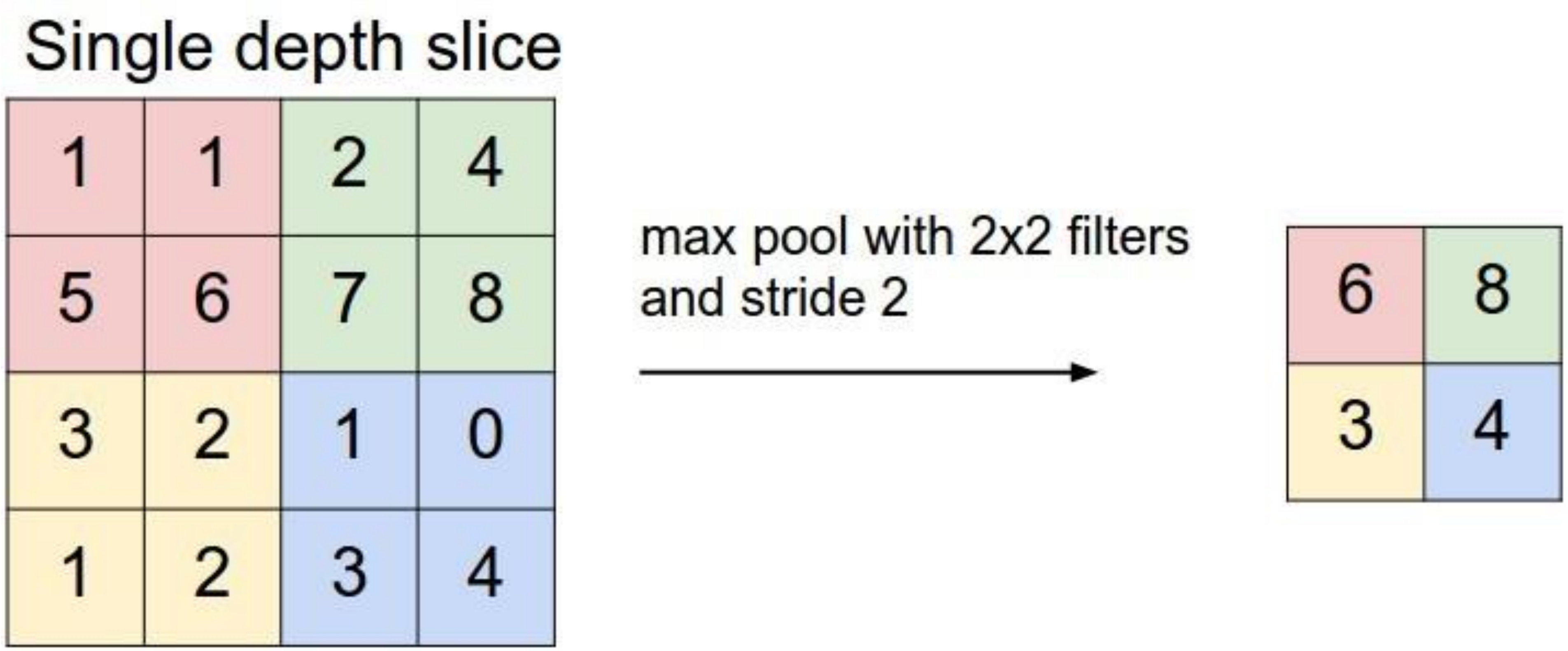
Source: https://medium.com/apache-mxnet/convolutions-explained-with-ms-excel-465d6649831c

Max pooling

The original

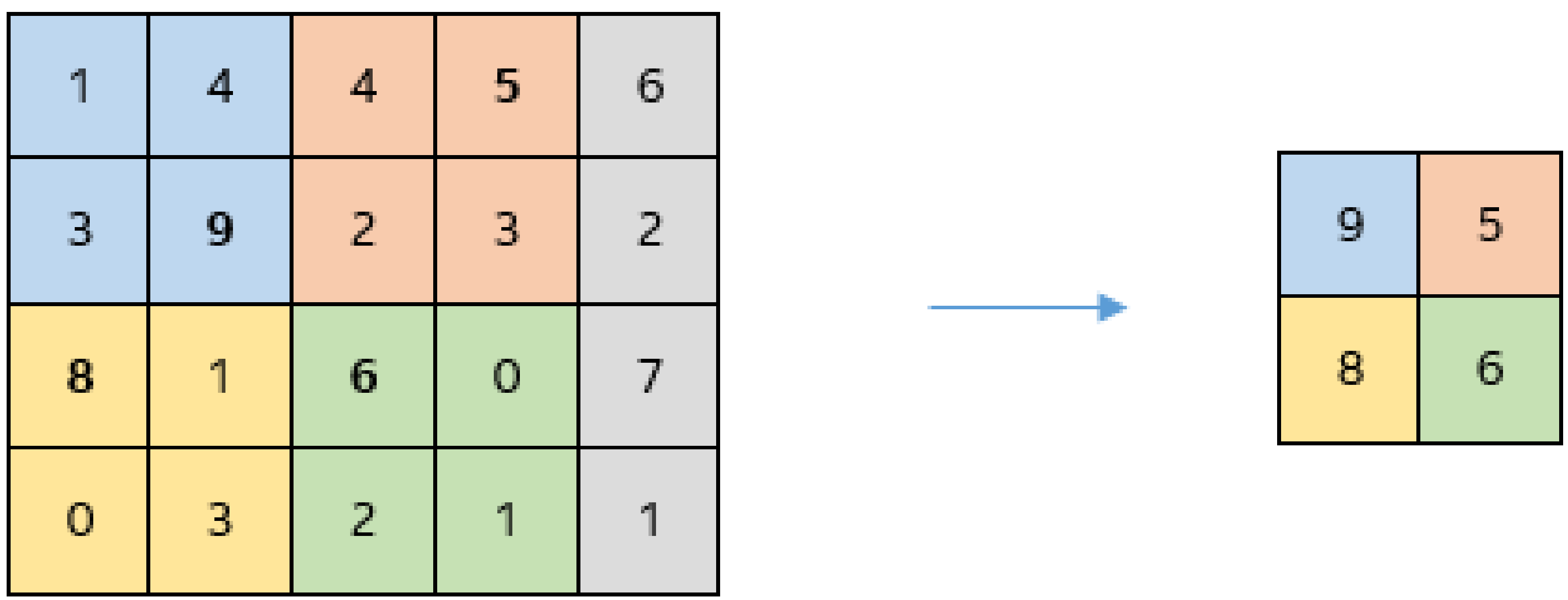
Convolutional vs max-pooling layers:

- **Convolutional layers extract features**
    - **big Kernels extract obvious features,**
    - **small ones extract more detailed features**
- **Max Pooling layers help to downscale the feature maps so that only features with significant weights are brought over to the next layer**



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

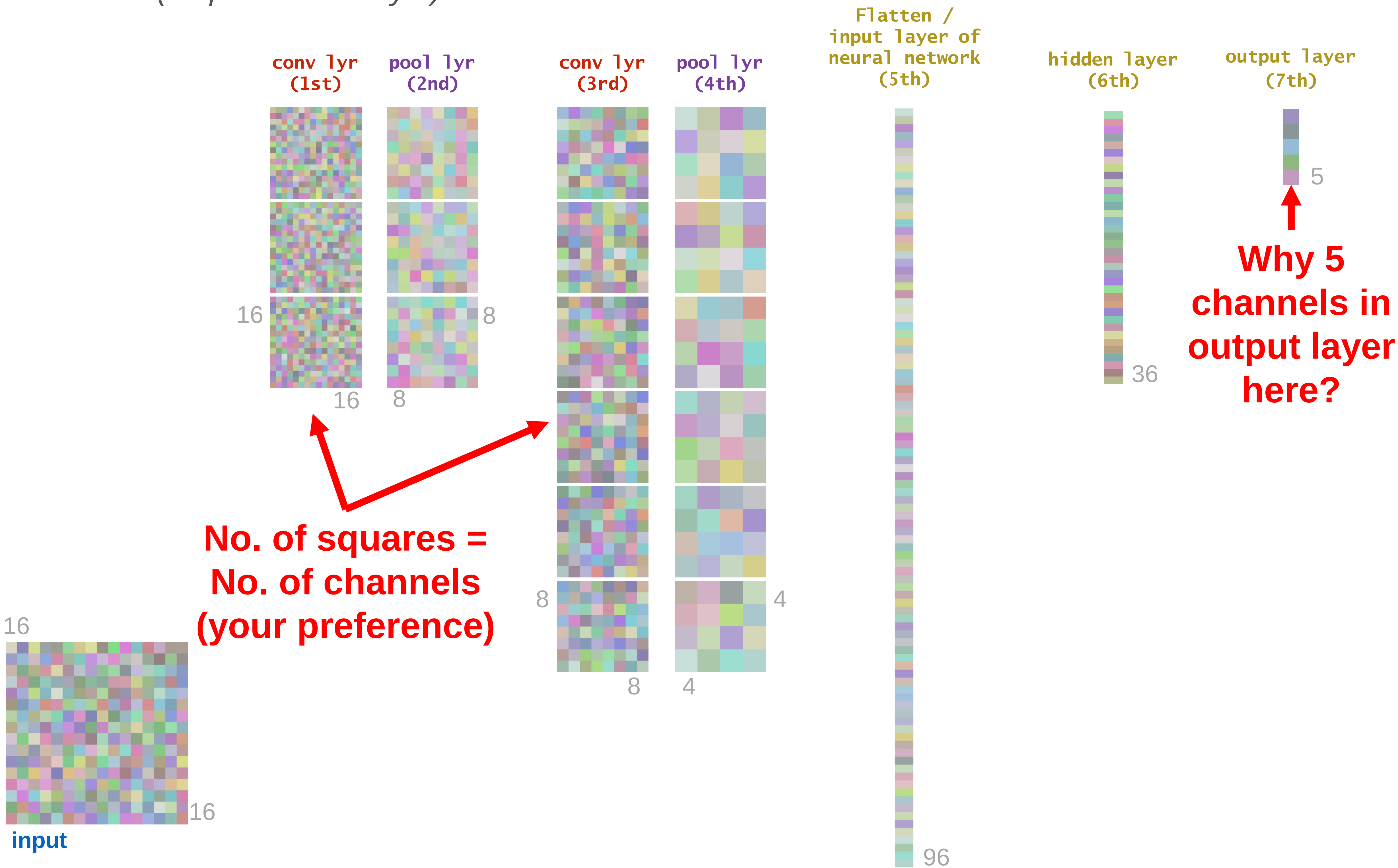
Max pooling
With situation



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

Convolutional neural network

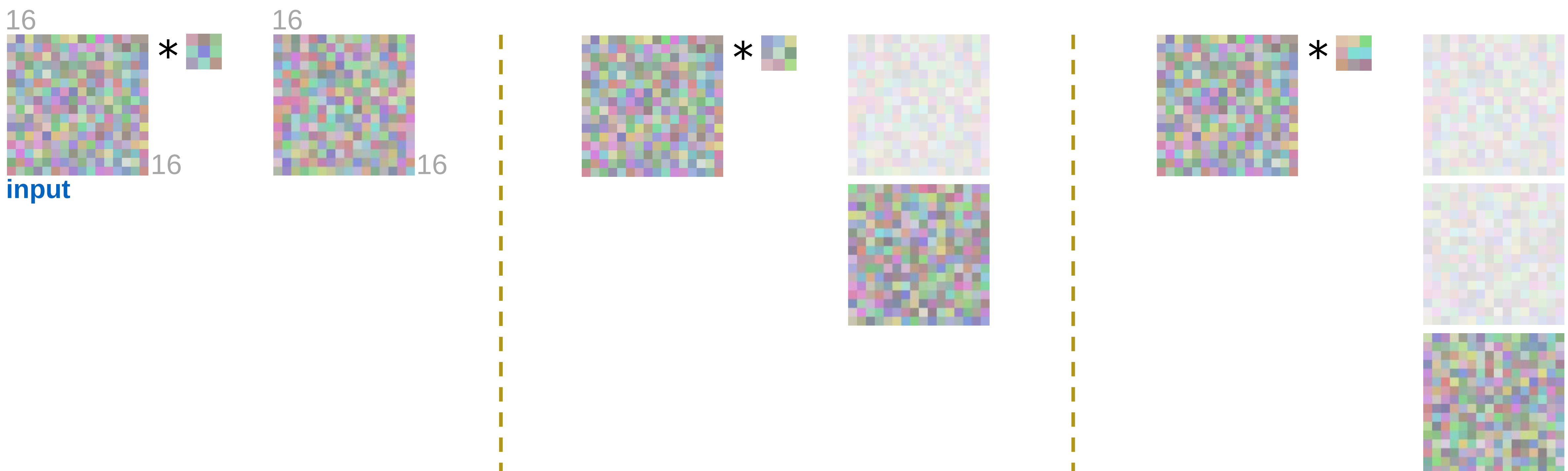
Overview *(output of each layer)*



The making of ...

The first convolutional layer (part 1)

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



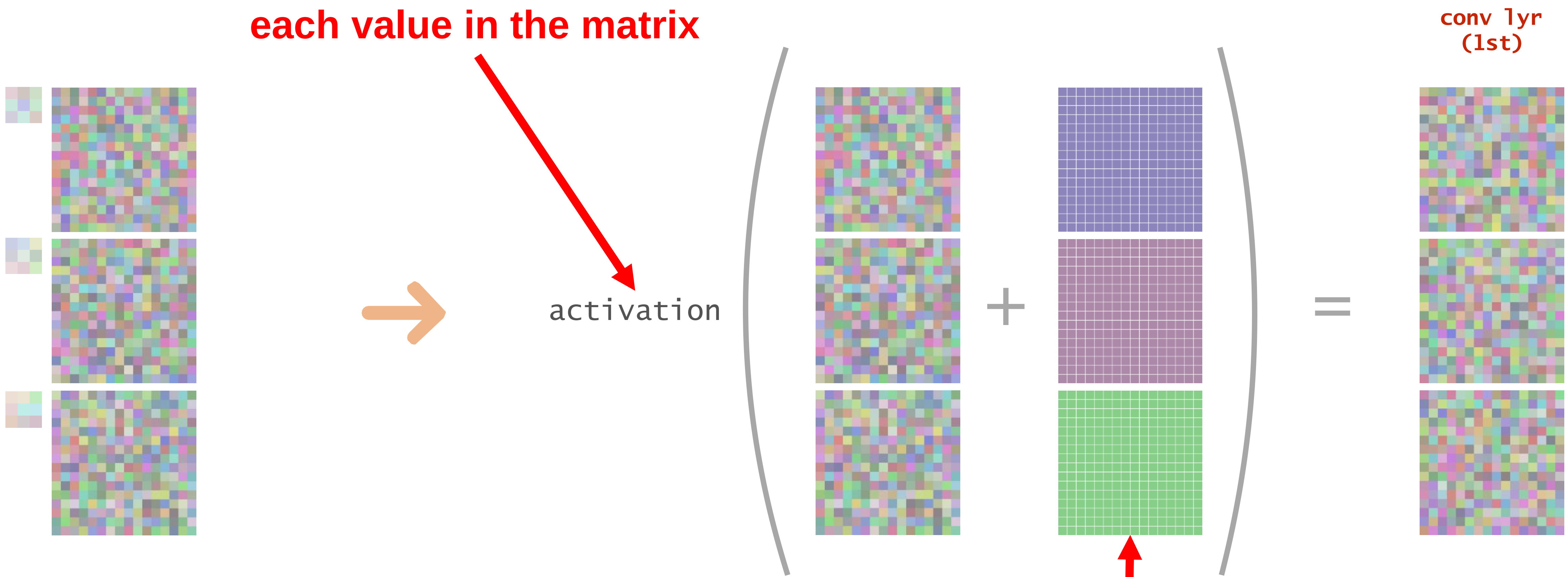
3 different Kernel types which result in 3 different channels

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

Can be ReLu or sigmoid function; the function is applied on each value in the matrix

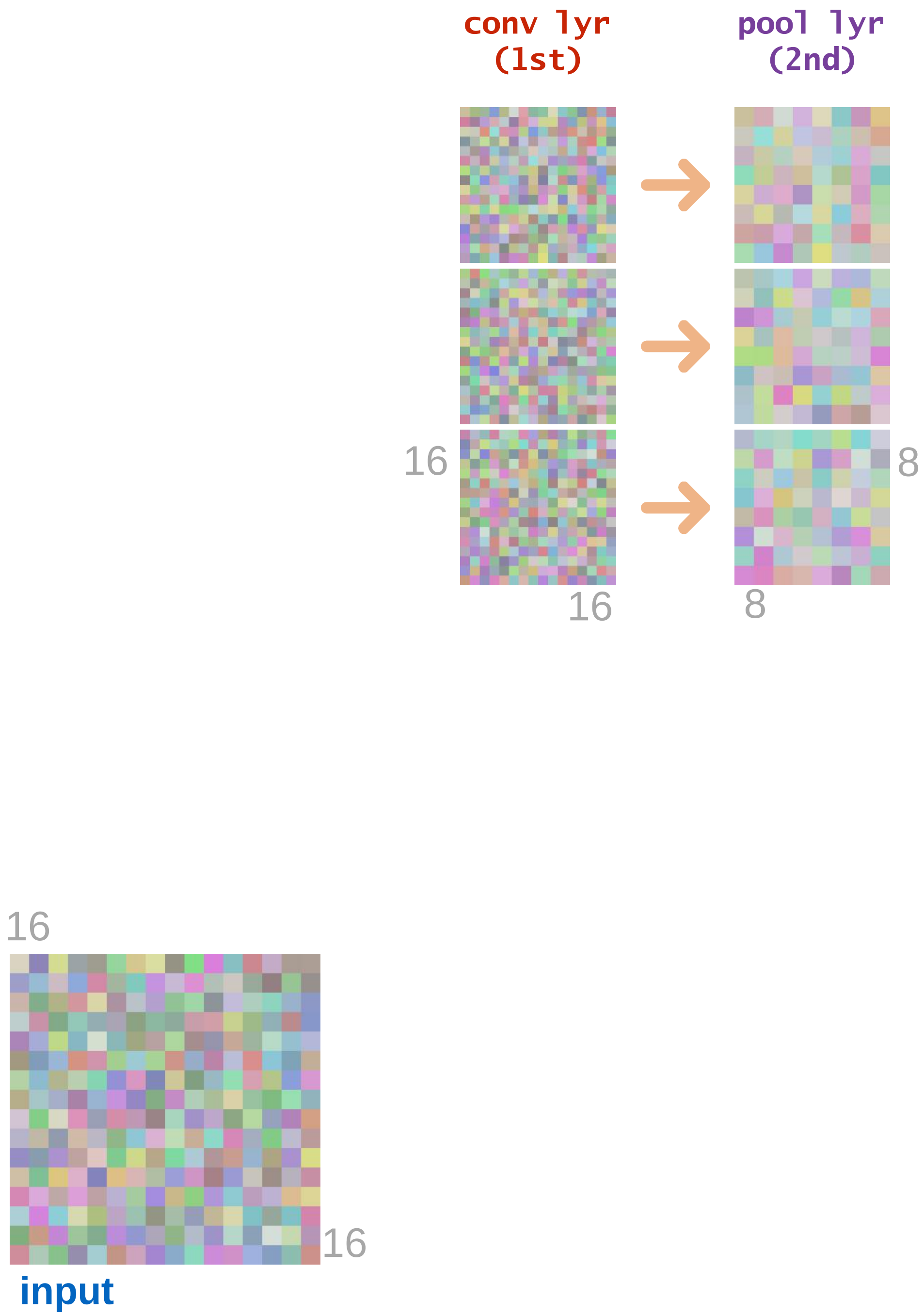


bias is a matrix and is the same throughout each matrix but different for various matrices

The making of ....

The pooling layer

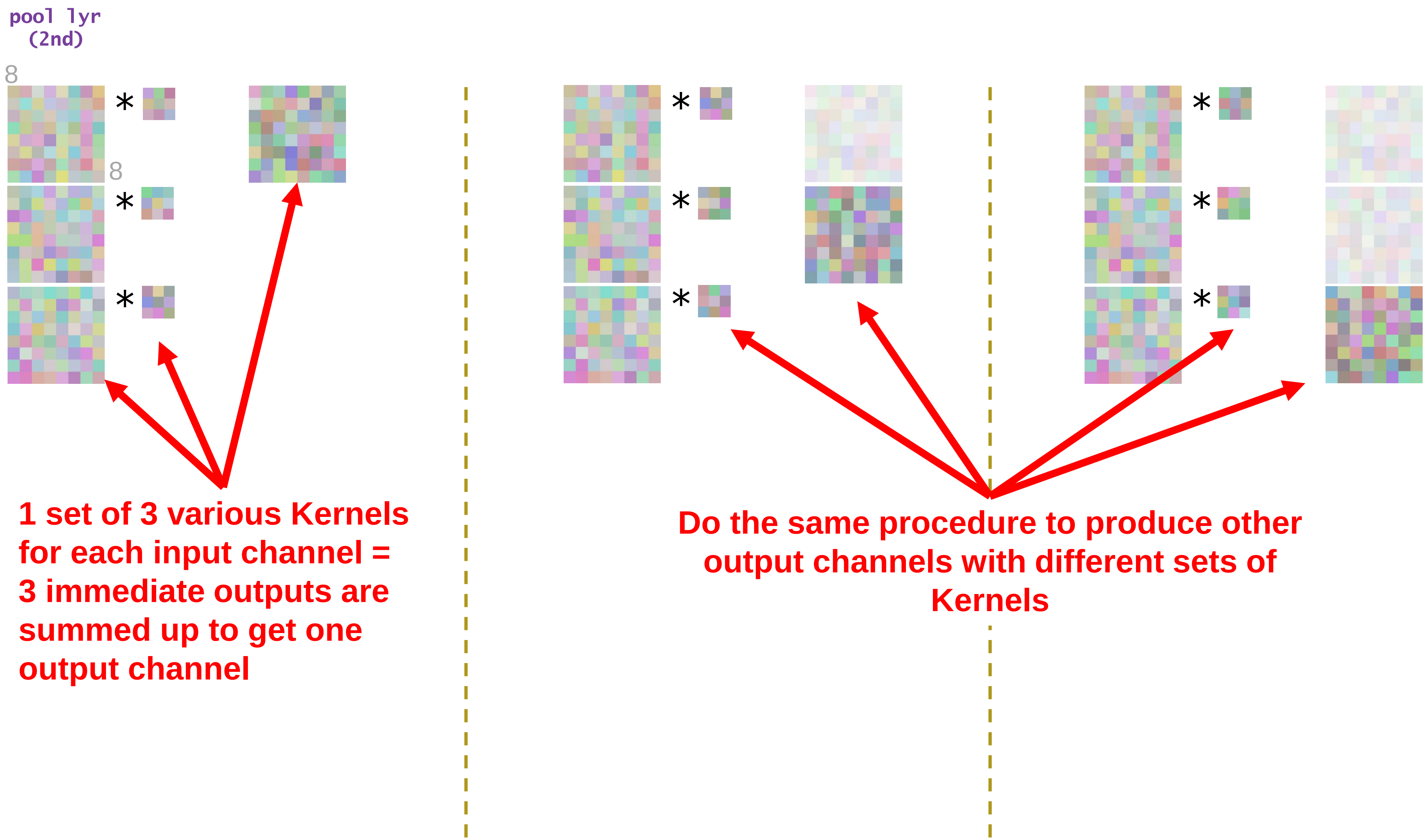
- Apply 2 x 2 max-pooling (stride 2) on the outputs from the first convolutional layer



The making of ...

The second convolutional layer (part 1)

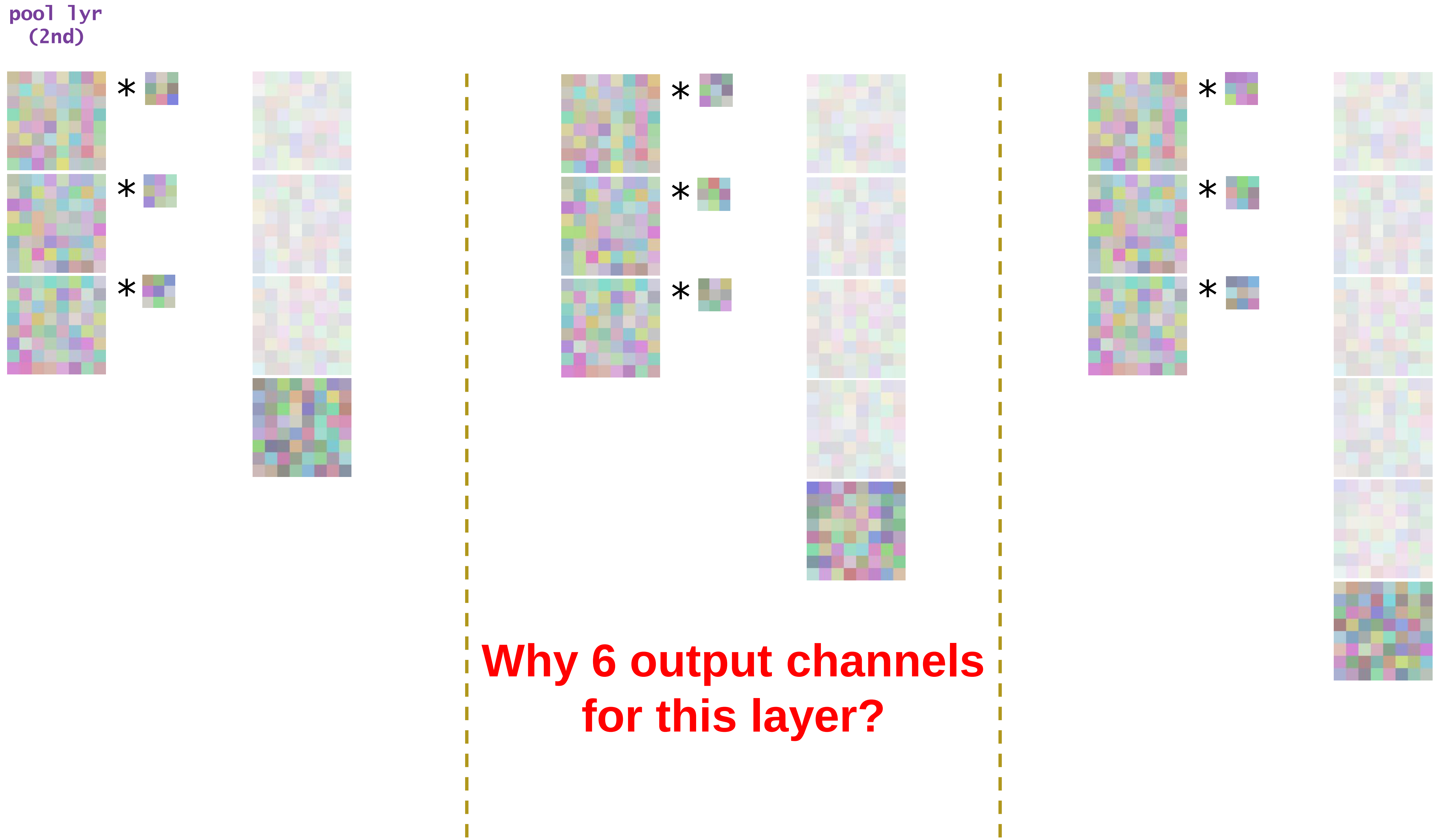
- Performs 6 separate multi-channel 2D convolutions (with padding) to generate 6 convolution outputs



The making of ...

The second convolutional layer (part 2)

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

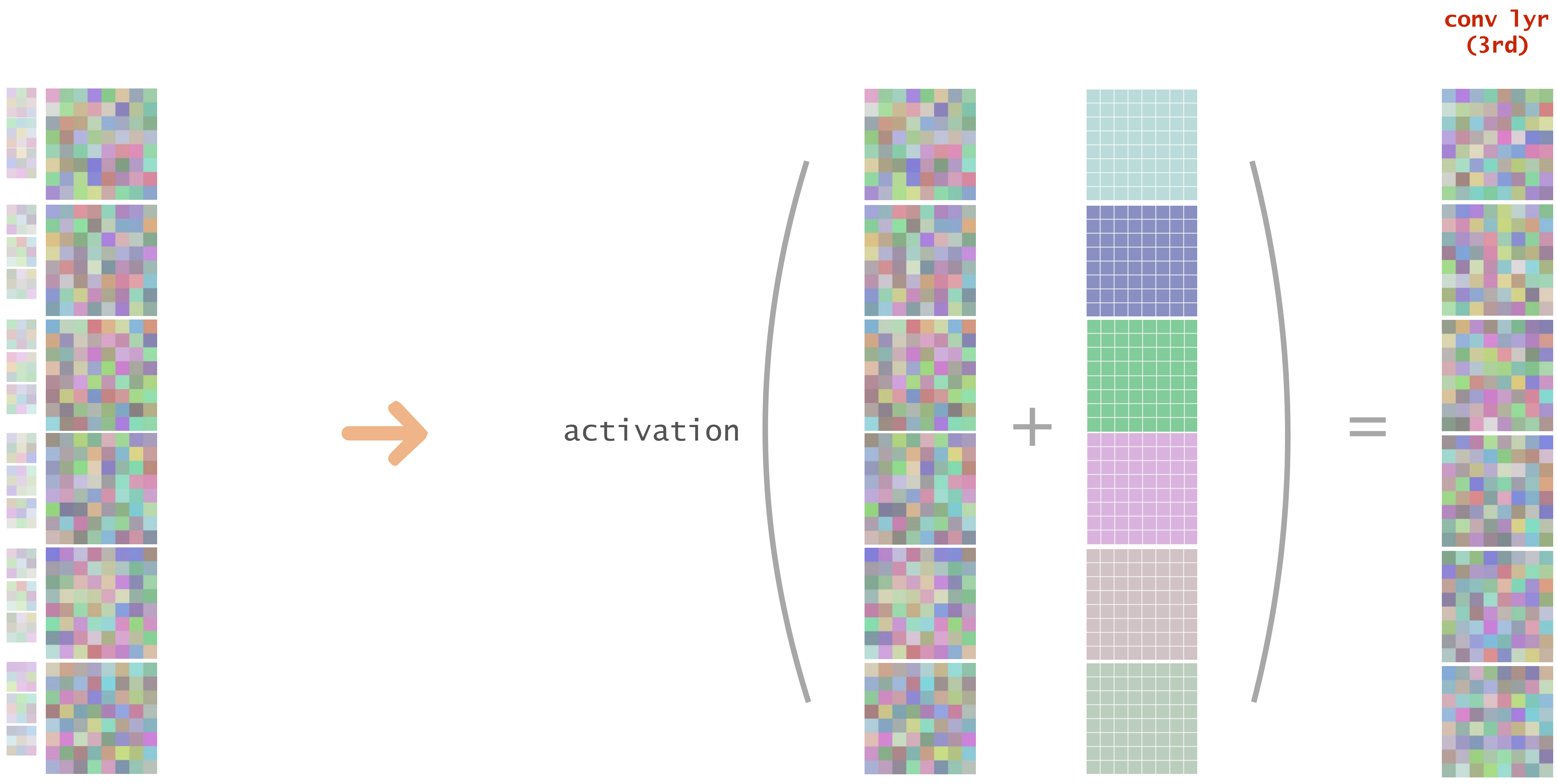


Why 6 output channels for this layer?

The making of ...

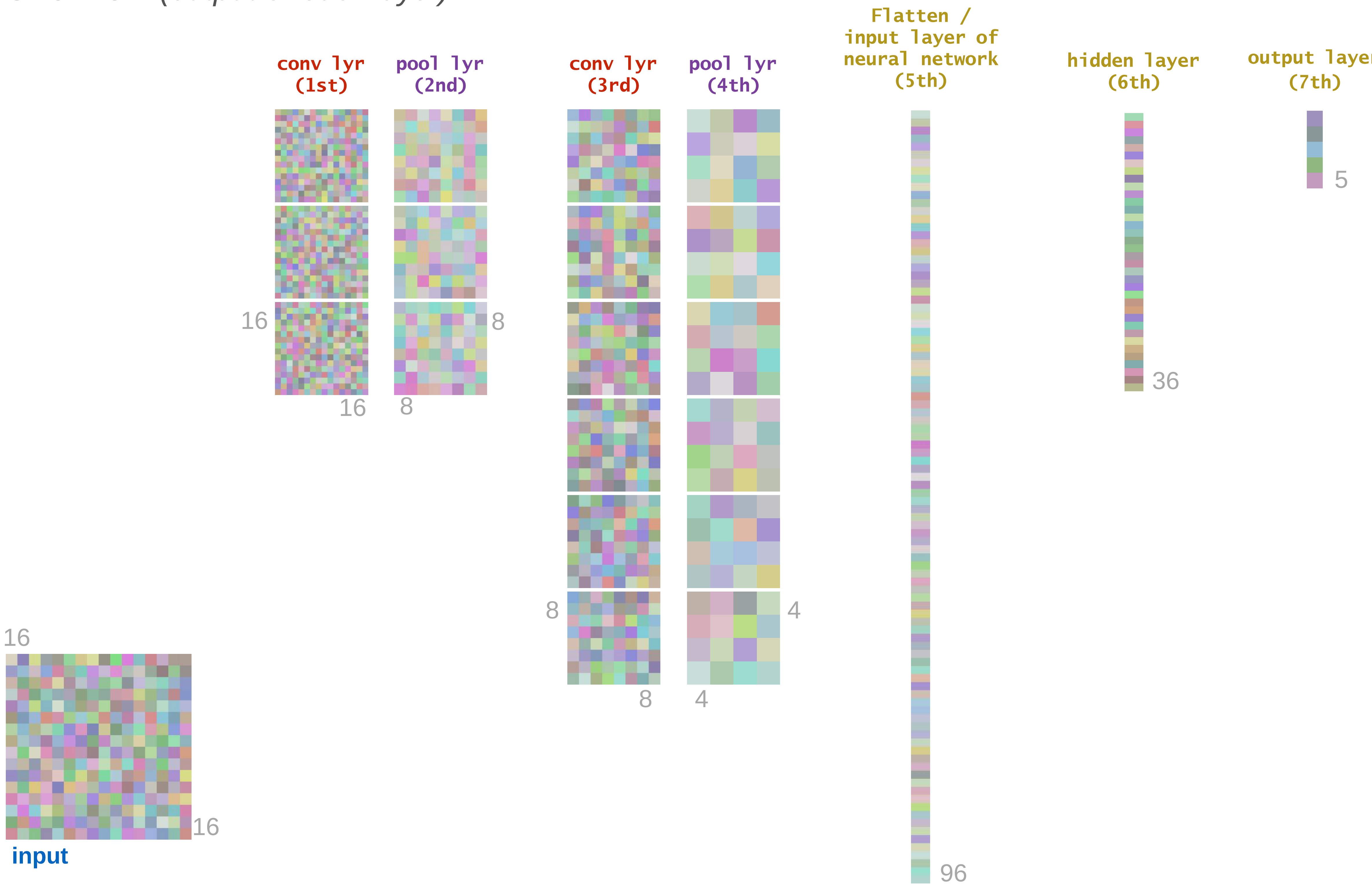
The first convolutional layer (part 3)

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

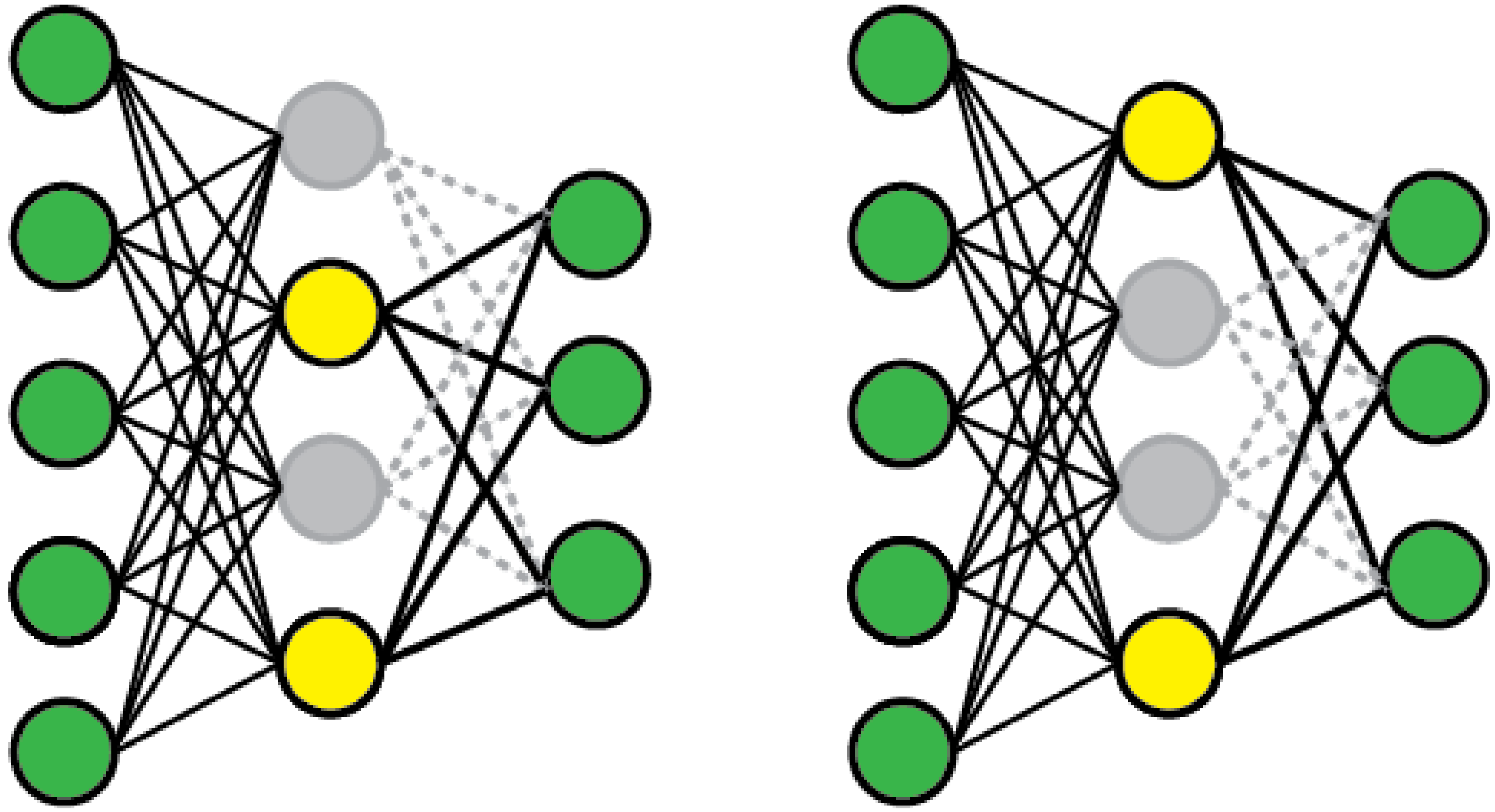


Convolutional neural network

Overview *(output of each layer)*



Convolutional neural network
Dropout

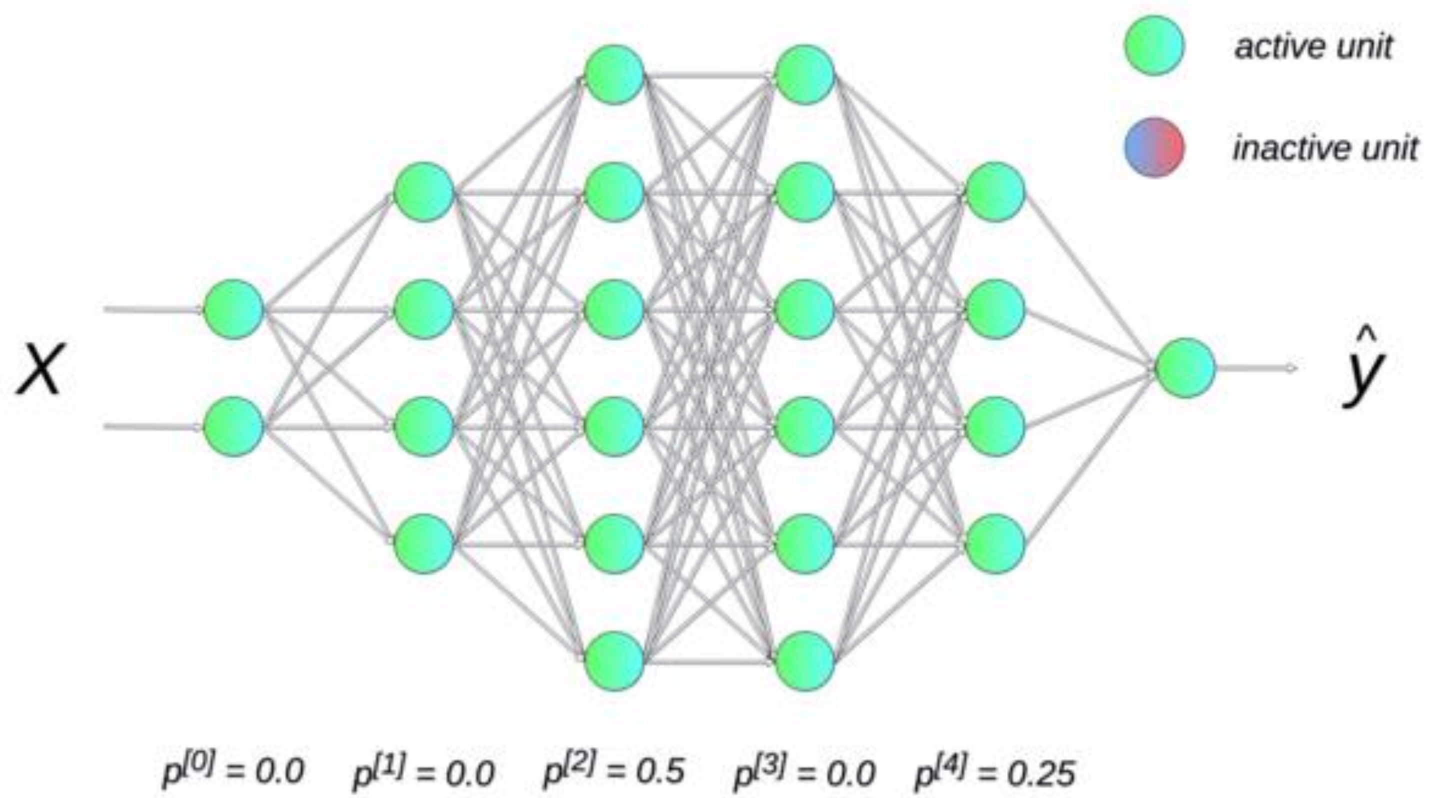


We will apply dropout in the workshop later on

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

Convolutional neural network

Dropout



- Very popular method to **regularize neural networks**; effective in preventing overfitting
- **Concept:**
  - Approximates training a large number of neural networks with different architectures in parallel
  - Every unit of the neural network (except output layer) is given the probability *p* of being temporarily ignored/muted (i.e. "dropped out") in calculations
  - Hyper parameter p is called **dropout rate** and very often its default value is set to 0.5
  - In each iteration, the neurons are randomly selected according to the assigned probability. As a result, each time we work with a smaller neural network

Source: https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a

Any fans of Japan?

RECAP:

(3 basic components in deep learning learnt)

1. **Convolutions**
2. **Pooling**
3. **Dropout**

Cursive Kuzushiji

Automated solution?

Workshop objective:
To classify hand drawn Japanese characters

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

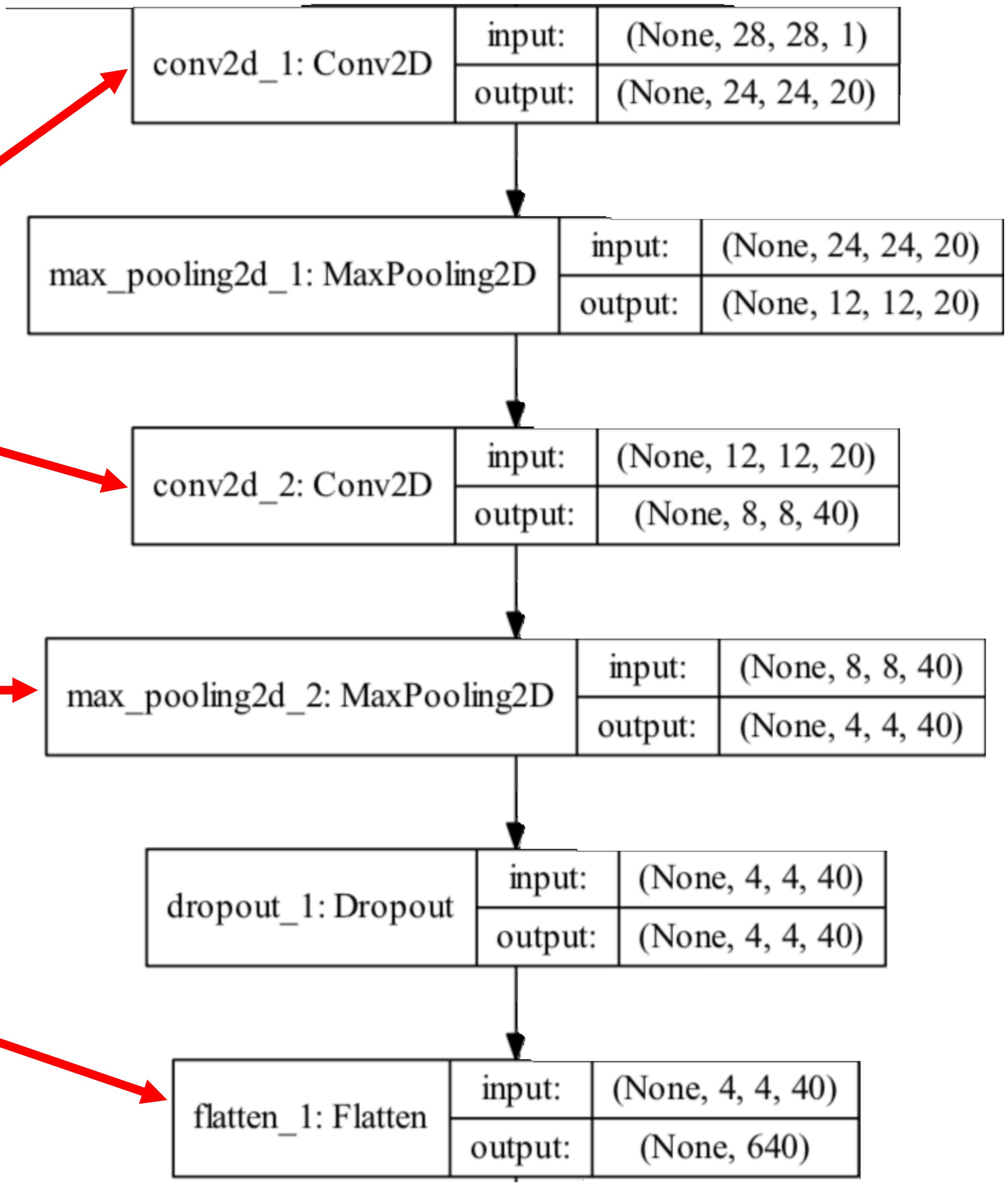
Kuzushiji MNIST

Another 'MNIST' alternative

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

Kuzushiji MNIST

The basic model, part 1



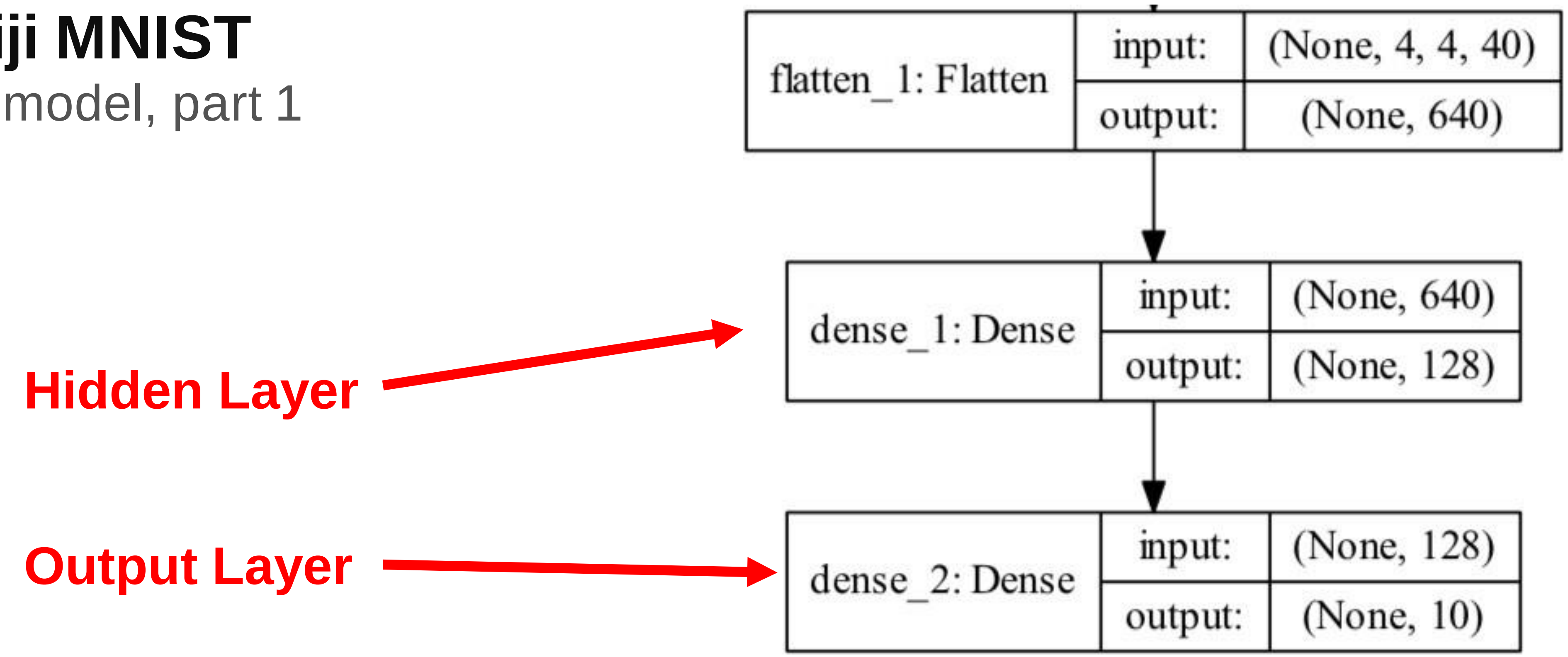
Filter Kernel size of 5x5 for all Conv Layers; to be shown later

Max pooling of size 2x2 will reduce the matrix size by half; no. of channels does not change

When flatten, the layer becomes a 1D array; 4 x 4 x 40 = 640

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; note that we are using Keras function under the tensorflow; not using the keras directly

To save the model

To train and test data using the model

```
> 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
```

Approach to build our model for this WS; other approaches will be covered in PRMLS course

Function that allows us to convert our labels from integer into a one hot encoding type

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()
```

The key is to set vmin and vmax 0 and 1 in order to display the gray scale correctly

Otherwise imshow will rescale our gray scale images, which is not desirable for this application

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

Use numpy function to load the data (in npz format in numpy

```
> 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']
```

Provided data is in uint8 format (unsigned 8-bit integer; range 0~255; this is too large!)

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

Hence, convert data type to float for both tr and ts data (rescale range to 0~1; optimal range)

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

Kuzushiji MNIST

3. Data preparation, part 2

60000 images for tr; rows & col is 28x28

10000 images for ts; rows & col is 28x28

- 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) to fit into Keras API

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

Reshaping functions to include the channel; necessary step to meet the requirements for the Keras API

Kuzushiji MNIST

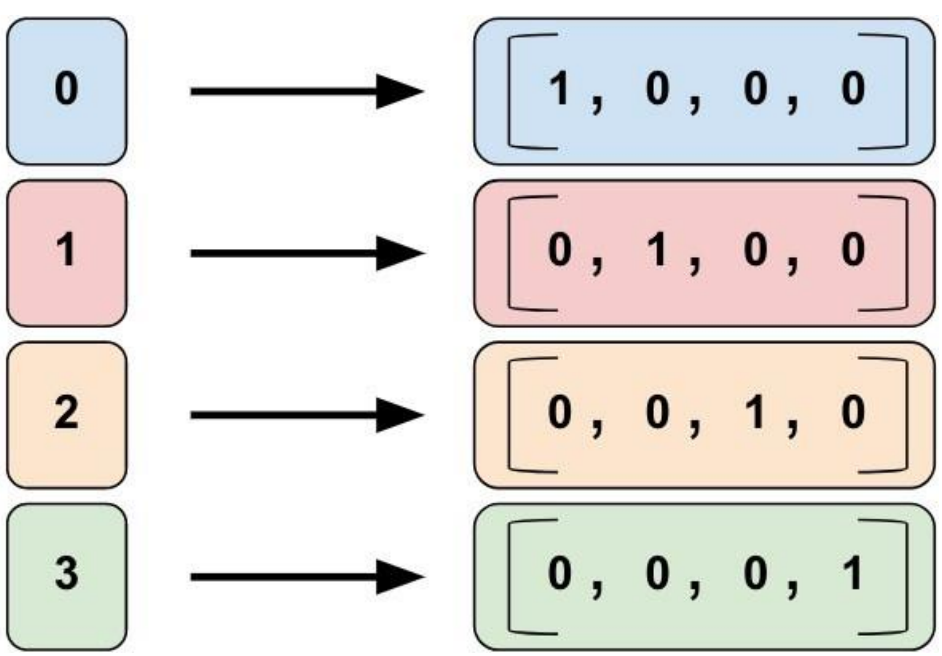
3. Data preparation, part 3

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

Use the to_categorical function to perform the one-hot encoding

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

Preparation done! Now we want to define the model

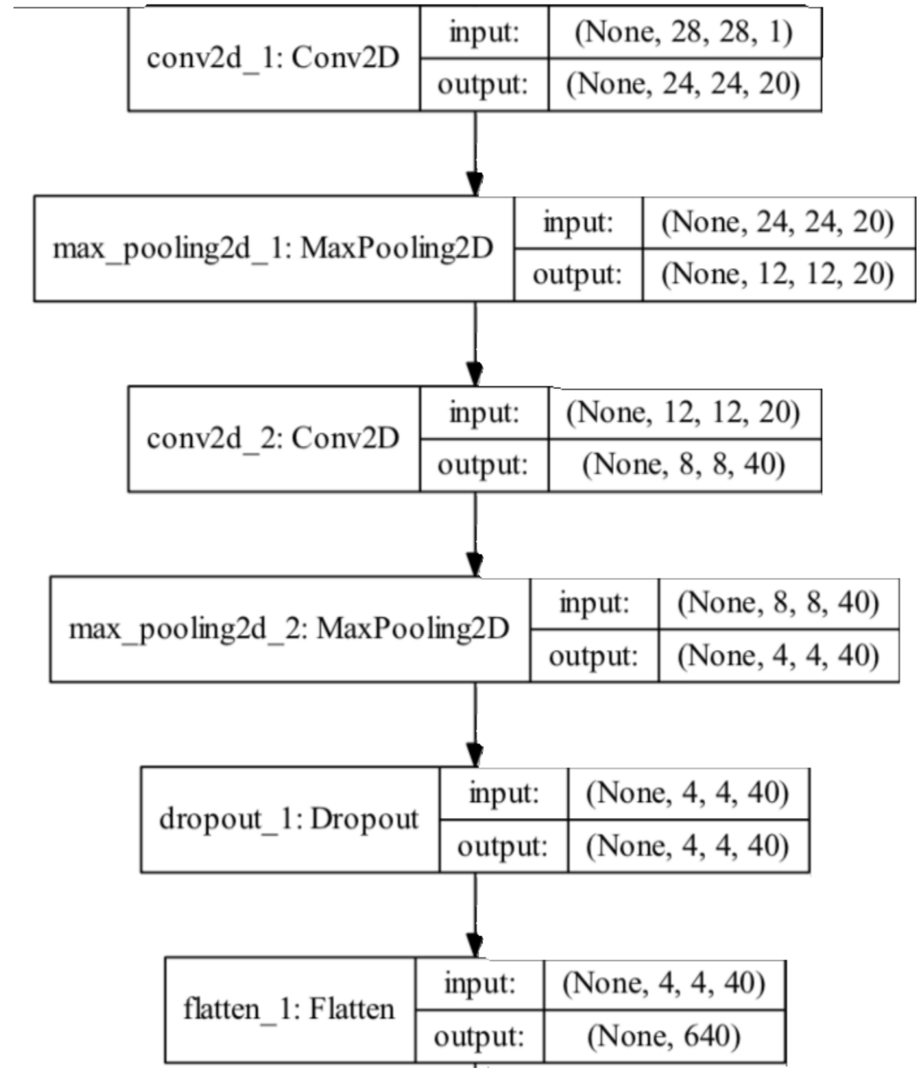


Example of a 4 class label that are converted into a one-hot encoding type

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

Kuzushiji MNIST

4. Define model, part 1



Main Thing to do for today's workshop!

```
> 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'))
      .....
```

Dropout rate = 0.2

Note that input shape has to be defined only for the 1st Conv2D layer

FAQs

1. Where to put the Dropouts?
2. Max Pooling size strictly 2x2?
3. Bigger vs Smaller Kernel size?
4. How many channels/neurons should I put for each CNN/Dense layer?
5. How many CNN/Dense layers should I put?
6. How to enable GPU on colab?
7. How to put padding? (to be answered next few slides)
8. How do I change the stride value? (to be answered next few slides)

Kuzushiji MNIST

4. Define model, part 1

Changing Padding Type

```
> 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', padding='same/valid'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Dropout(0.2))
      model.add(Flatten())
      model.add(Dense(128, activation='relu'))
      .....
```

Using padding!

No padding!

Kuzushiji MNIST

4. Define model, part 1

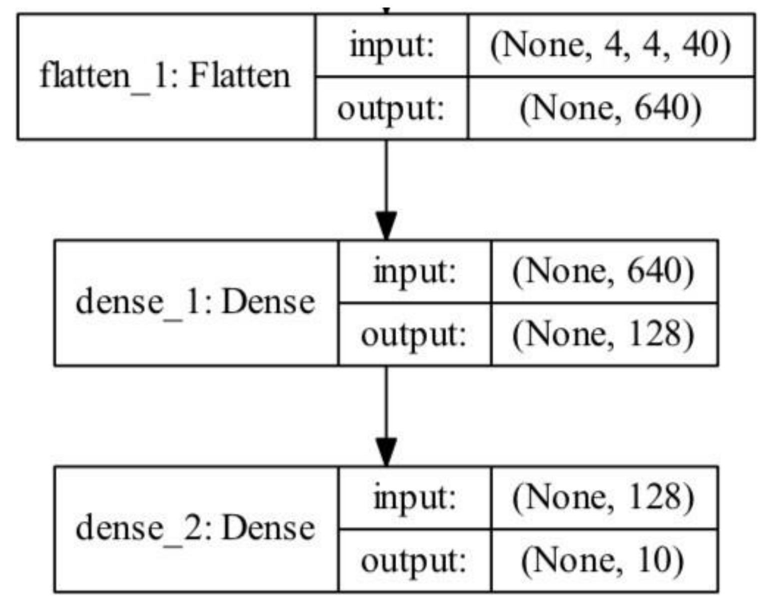
Changing Stride Value

```
> 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), strides=(2, 2)))
      model.add(Conv2D(40, (5, 5), strides=(2, 2), 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



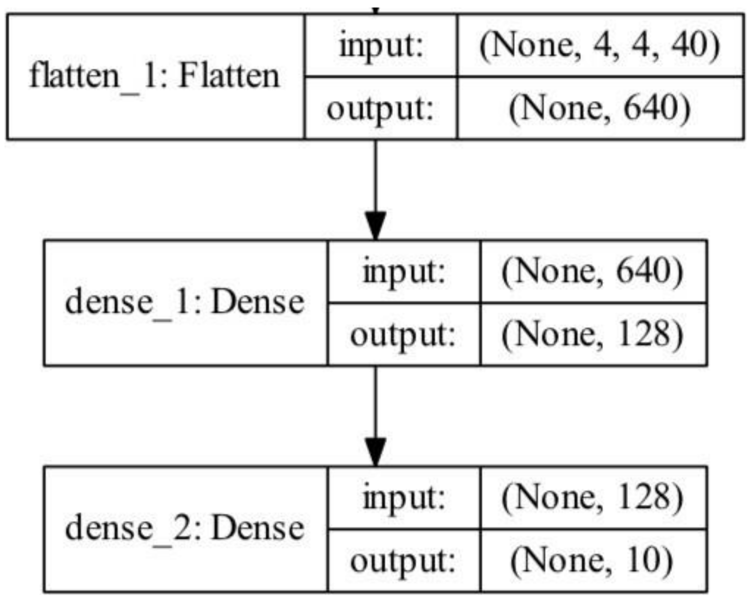

```
> seed          = 29
> np.random.seed(seed)

> 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
```

Dense layers perform the main classification tasks

Kuzushiji MNIST

4. Define model, part 1



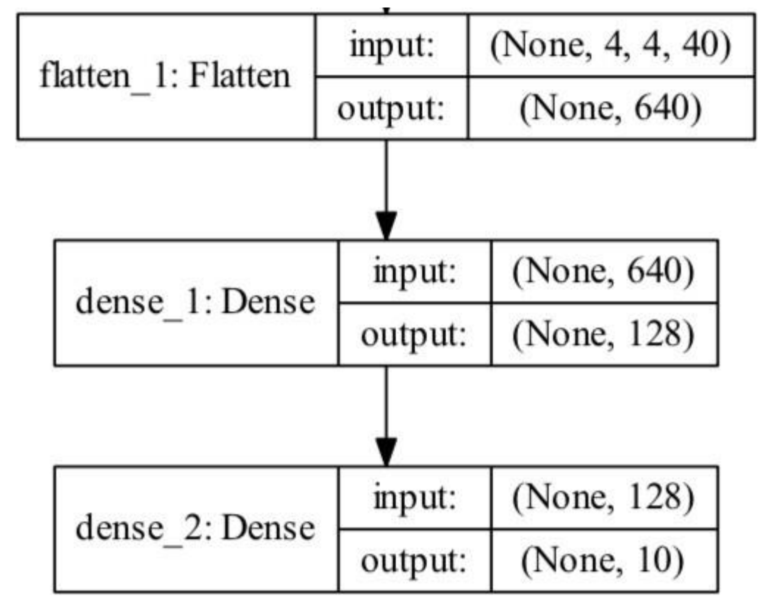

```
> seed          = 29
> np.random.seed(seed)

> 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
```

Softmax enables you to identify which output is true within the output layer (in terms of probabilities)

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
```

Optimizer = an algorithm that tells the framework how to update the weights and bias. Backbone of this is backpropagation; different optimizers have different ways to update the weights and bias

Kuzushiji MNIST

4. Define model, part 2

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

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

> model.summary()
```

Summary allows you monitor your model to check if your model construction is done correctly

Creating two models here (one for training and another for final evaluation); not necessary to do this but it is a good practice

Output example

```
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

Before training, we need to specify where/how we are saving the model

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

Monitor model set to “validation accuracy”

Use ModelCheckpoint to save the model in the middle of the training process

Save when the validation accuracy is the max (i.e. mode = max)

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

Lastly, put the 2 objects (i.e. checkpoint and csv_logger) into a list, named as callbacks_list; these 2 objects (i.e. callbacks) will be called after each training epoch

Log the training and testing information into a CSV file via CSVlogger

Kuzushiji MNIST
5. Train model

- Training is only a single line

Use the model.fit function to do the training

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

Epochs is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset

Batch size is a hyperparameter that controls each time how many samples are taken at one go to train and update the weights

```
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

After training, we need to do a final evaluation (no training) using a fresh model (i.e. modelGo); must be fresh to test the model with the trained weights

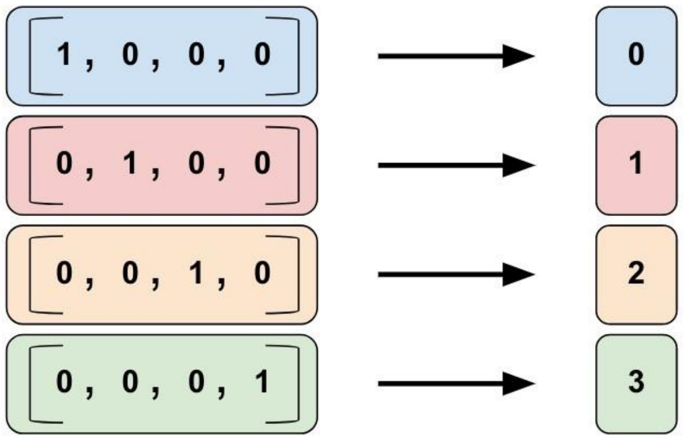
```
> modelGo.load_weights(filepath)
> modelGo.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

Weights loaded from the trained model and the fresh model is recompiled

- Test the model, calculate the accuracy and confusion matrix

Use the modelGo.predict function to test the model using the testing data set

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



Use this function to convert “predicts” output from one-hot encoded to integer type; need integer format to obtain the accuracies and to report the matrix

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

Do the same conversion for the test labels

```
> 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)
```

Calculates the test scores in accuracy

Calculates the confusion matrix

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
```

Classification Report

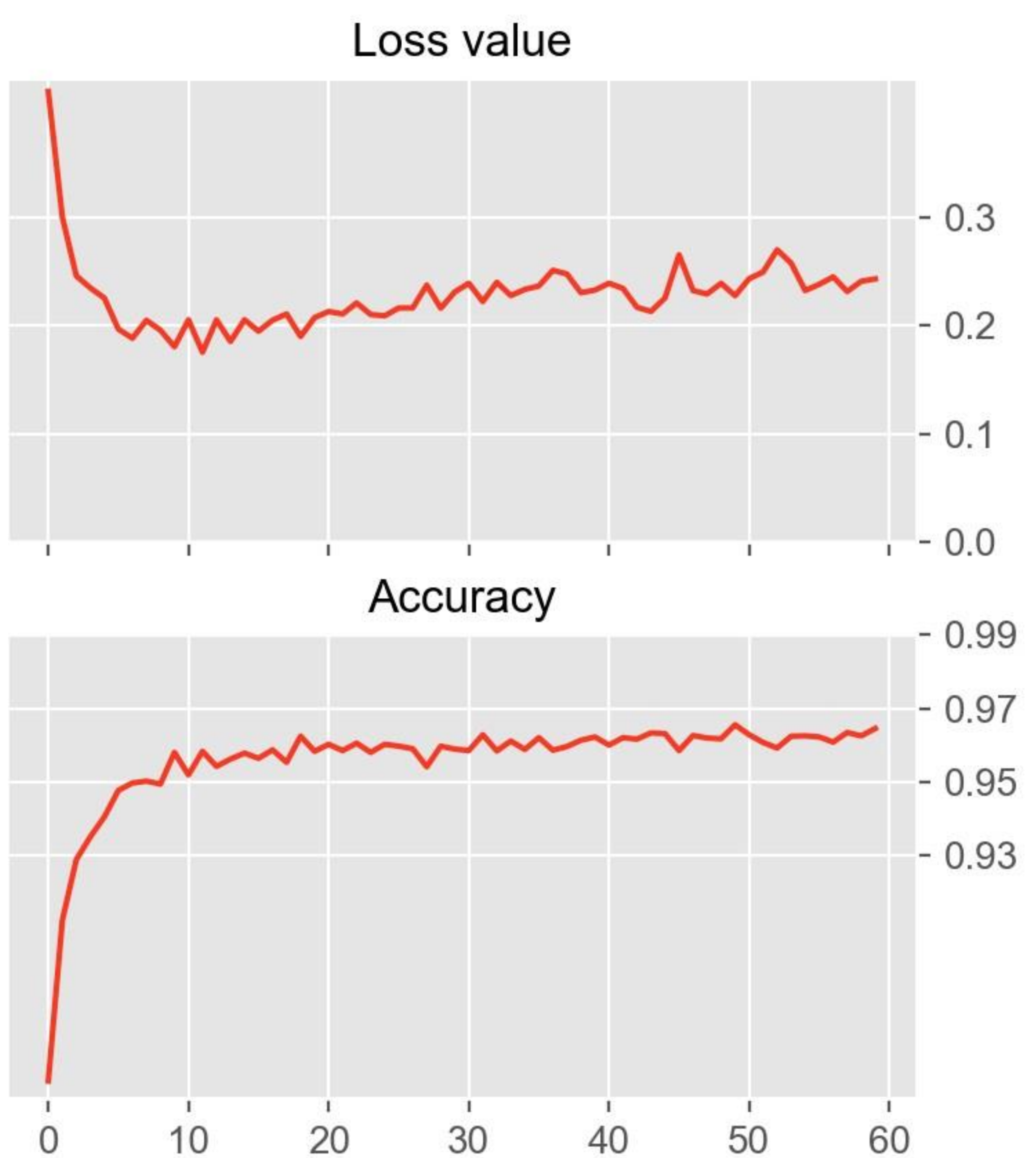
```
[[974   2   1   1  18   1   0   1   1   1]
 [  5 943   6   0   5   2  24   3   7   5]
 [  8   3 939   9   4   7  19   4   7   0]
 [  0   0   8 982   0   4   5   0   1   0]
 [ 12   2   1   9 953   4   8   2   6   3]
 [  1   3  13   4   1 960  13   0   3   2]
 [  0   2   3   0   1   2 992   0   0 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

Confusion Matrix

Kuzushiji MNIST

6. Test model, part 4

Use pandas to read the training log in csv (i.e. modelname + '.csv') that we have saved just now



- 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()
```

Project:

Try the different variations in your model and observe their performances

Rmb to enable your GPU in your COLAB!

1. The original model; given previously; this will serve as the base for the rest
2. Add 1x CNN and 1x MaxPooling layer; decide on the channels and sizes for these layers
3. Step 2 plus add 2 more Dropout layers of 20%; decide where to put them
4. Steps 2 & 3 plus add an additional dense layer of activation RELU; decide on the number of neurons yourself for this layer
5. A model of your own configuration that gives an optimal accuracy (must be more than the original model's accuracy); **this version to be submitted**

Project (Optional; only if you have time):

Try the different variations in your model and observe their performances

Rmb to enable your GPU in your COLAB!

6. From the original model, add padding for all CNN layers
7. From the original model, add 3x additional CNN layers; decide on the channels and sizes for all CNN layers
8. From the original model, add 3x additional dense layers of activation RELU; decide on the number of neurons yourself for all dense layers
9. Steps 7 and 8 together **WITHOUT** any additional Dropout layers
10. Steps 7 and 8 together **WITH** additional Dropout layers of 20%; decide how many Dropout layers you need and where to put them

For those who have finished your quiz but have not finished your workshop, please do so and upload.

For those who have finished your quiz and have uploaded your workshop, please give me a while to mark your quiz and check your submission. After which, you may leave early.