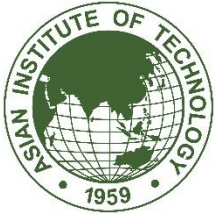


Convolutional Neural Network

Dr. Mongkol Ekpanyapong

Fully Connected Network (FCN)

- Fully connected layers that takes in a 256x256 images and maps to 10 output neural will have 256x256x10 = 655,360 parameters per node
- Hence, the FCN model is more complex
- FNC is tend to be more overfit

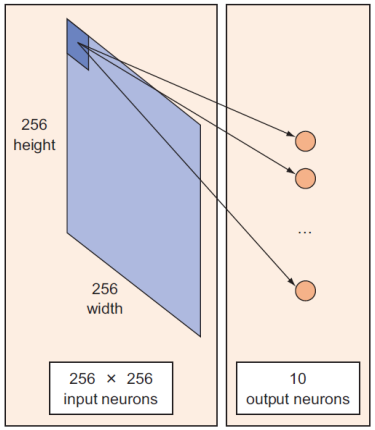


Image from Machine Learning with TensorFlow book

FCN/CNN layer Comparison

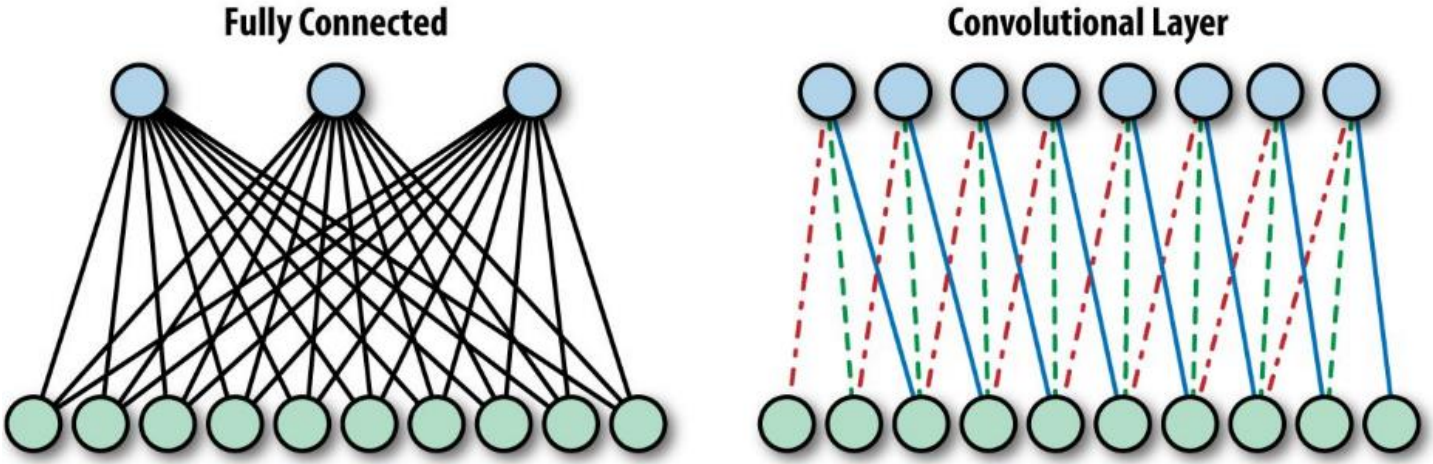


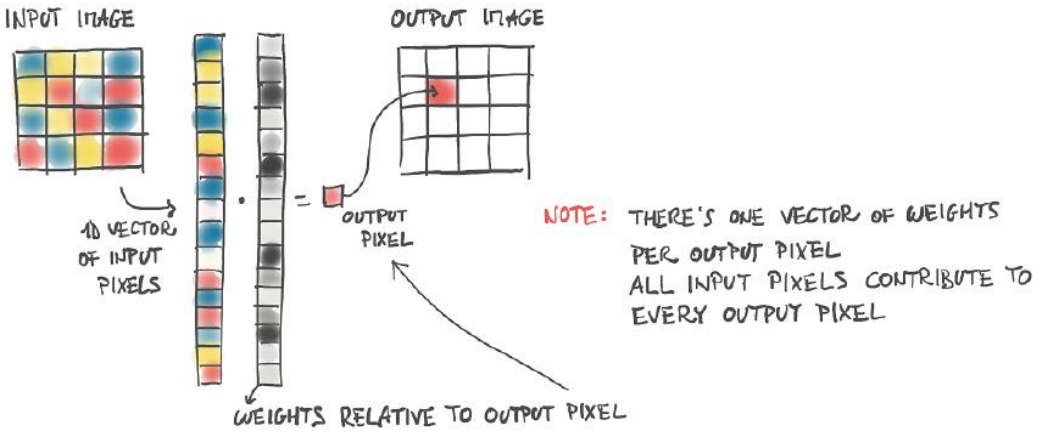
Image from Learning TensorFlow book

The Limit of Fully Connected

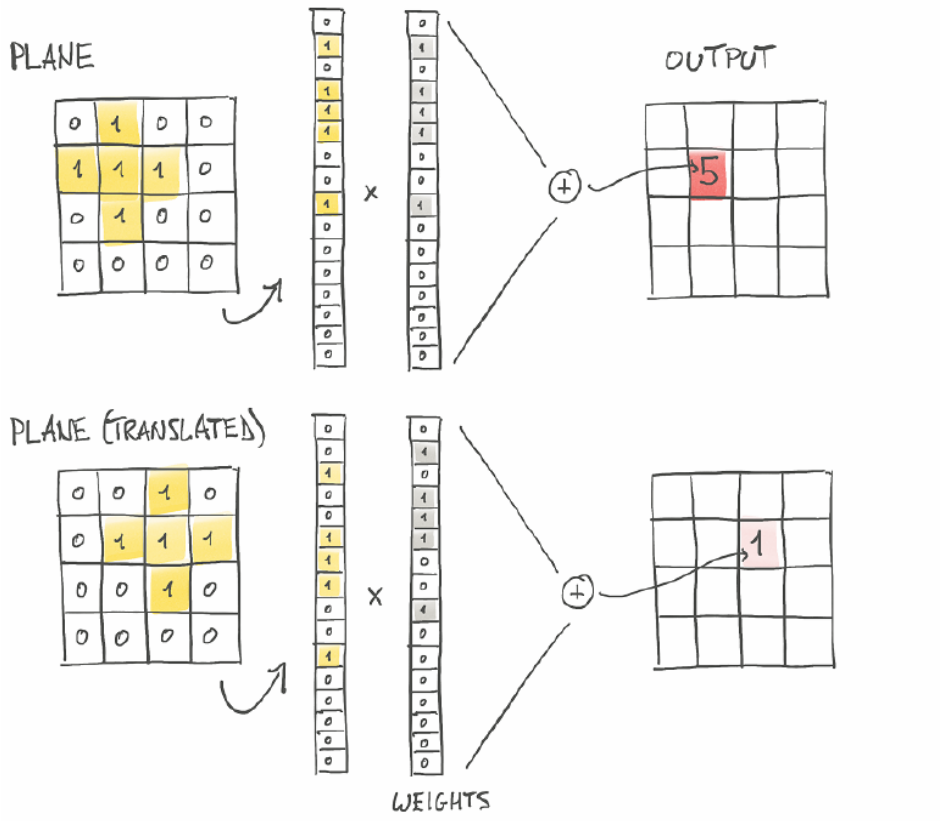
- Higher number of parameters
- We use information from far-away pixel during the prediction
- Not-translation invariant

The Limit of Fully Connected

- Every input pixel is combined with every other to produce the output results in large number of parameters



Not-Translation Invariant



CNN

- A convolution is a weighted sum of the pixel values of the image as the window slides across the whole image
- The difference between a fully connected layer and a convolution layer is that the fully connected layers learn global patterns whereas convolution layers learn local pattern
- Convolution operates over 3D tensor called feature map

Key Characteristic of CNN

- The patterns they learn are translation invariant (FCN is not)
- They can learn spatial hierarchies of patterns
- Also, it should learn about the filter automatically

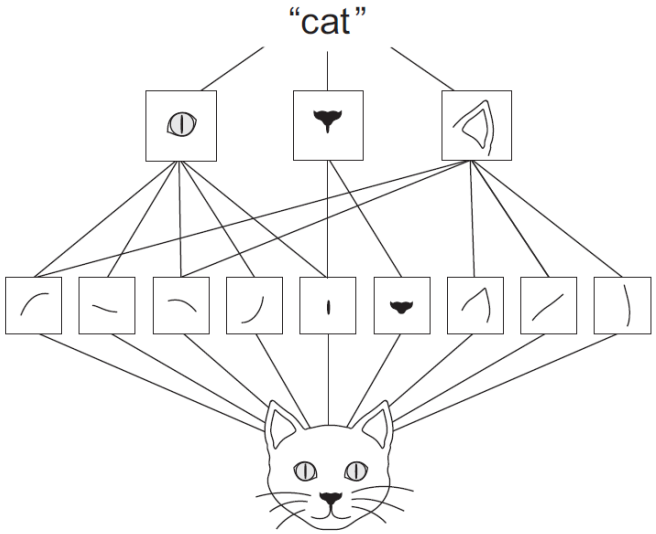
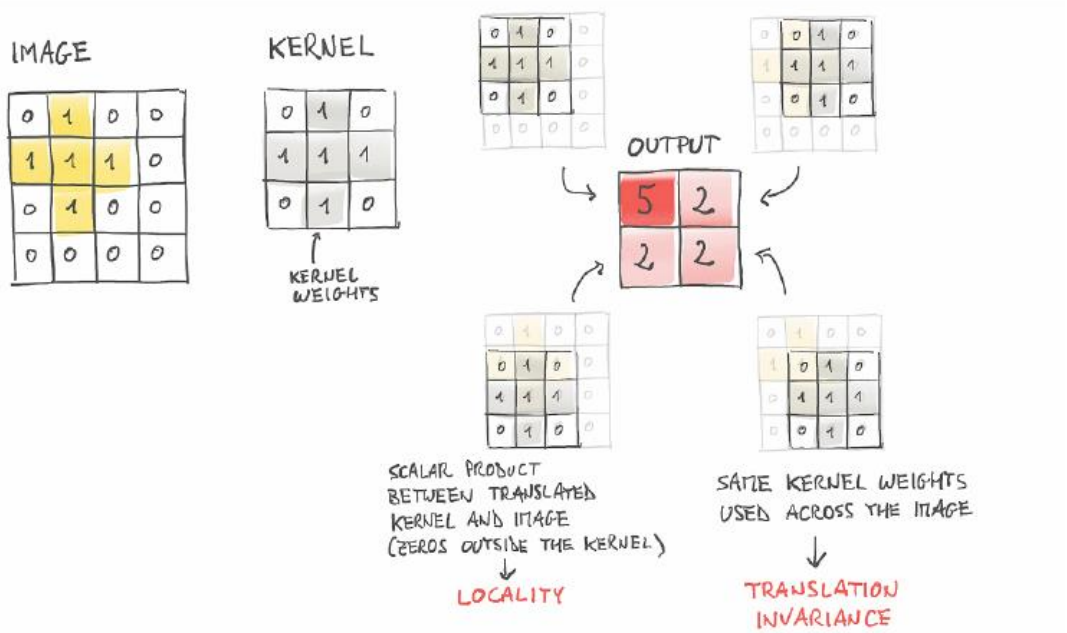


Image from Deep Learning with Python

CNN

- Locality and translation invariance



Convolutional Neural Networks

- CNN has just enough weights to look at a small patch of the image
- The number of parameters are reduced to just 5 x 5 = 25 parameters per node

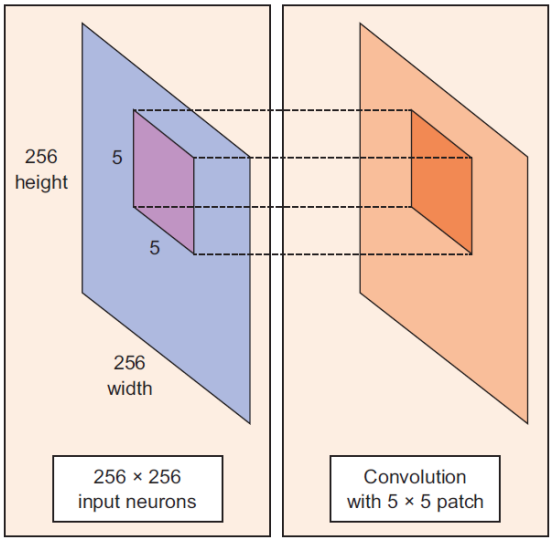


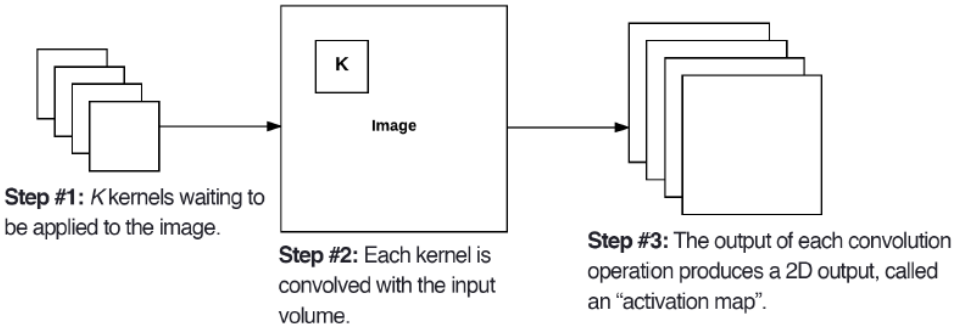
Image from Machine Learning with TensorFlow book

CNN Layer Types

- Convolutional (CONV)
- Activation (ACT) e.g., RELU or SOFTMAX
- Pooling (POOL)
- Fully-connected (FC)
- Batch Normalization (BN)
- DropOut (DO)

CONV layers

- Instead of fully connected layer, we can use convolutional layer instead
- Convolutional layer introduces the local connectivity and reduces the number of parameters for training



Example

- Given a grayscale image of size 256x256,

It connects to 10 output neurons, the number of parameters is

256x256x25 = 1,638,400 for a

fully connected layer

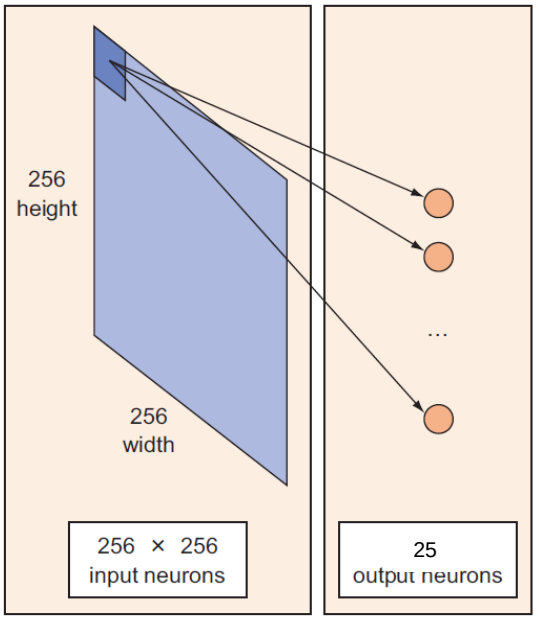


Image from Machine Learning TensorFlow book

Example with CONV

- Given a grayscale image of size 256x256,

It connects to 5x5 patch, the number of parameters is

5x5 = 25 for a

convolutional layer

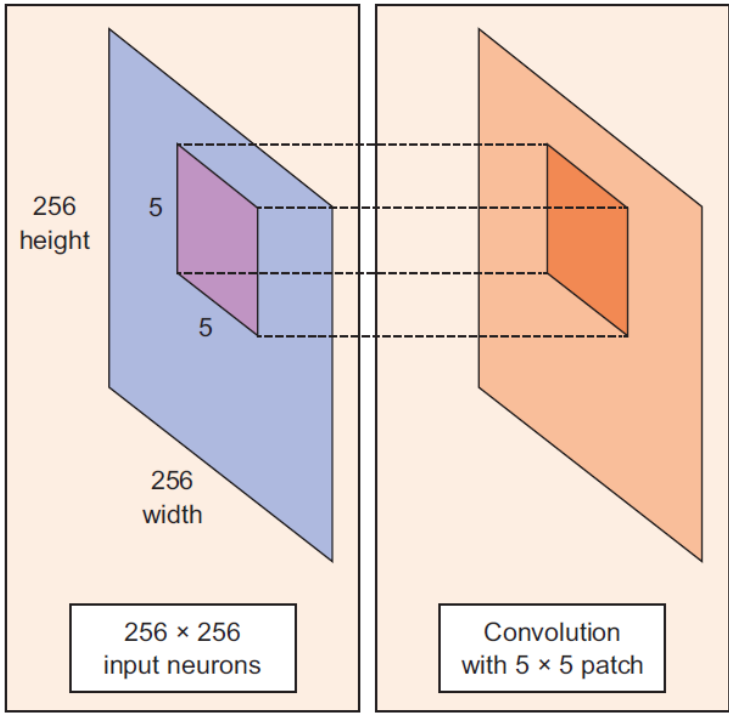


Image from Machine Learning TensorFlow book

Backpropagation

- We can consider the weight/parameter of each kernel similar to fully connected layer as shown below in which sigma is the activation function

$$\sigma\left(b+\sum_{l=0}^{4}\sum_{m=0}^{4}w_{l,m}a_{j+l,k+m}\right)$$

- Hence, the same back propagation can be applied in which now the convolution layer will learn the filters to be used

Multiple Feature map

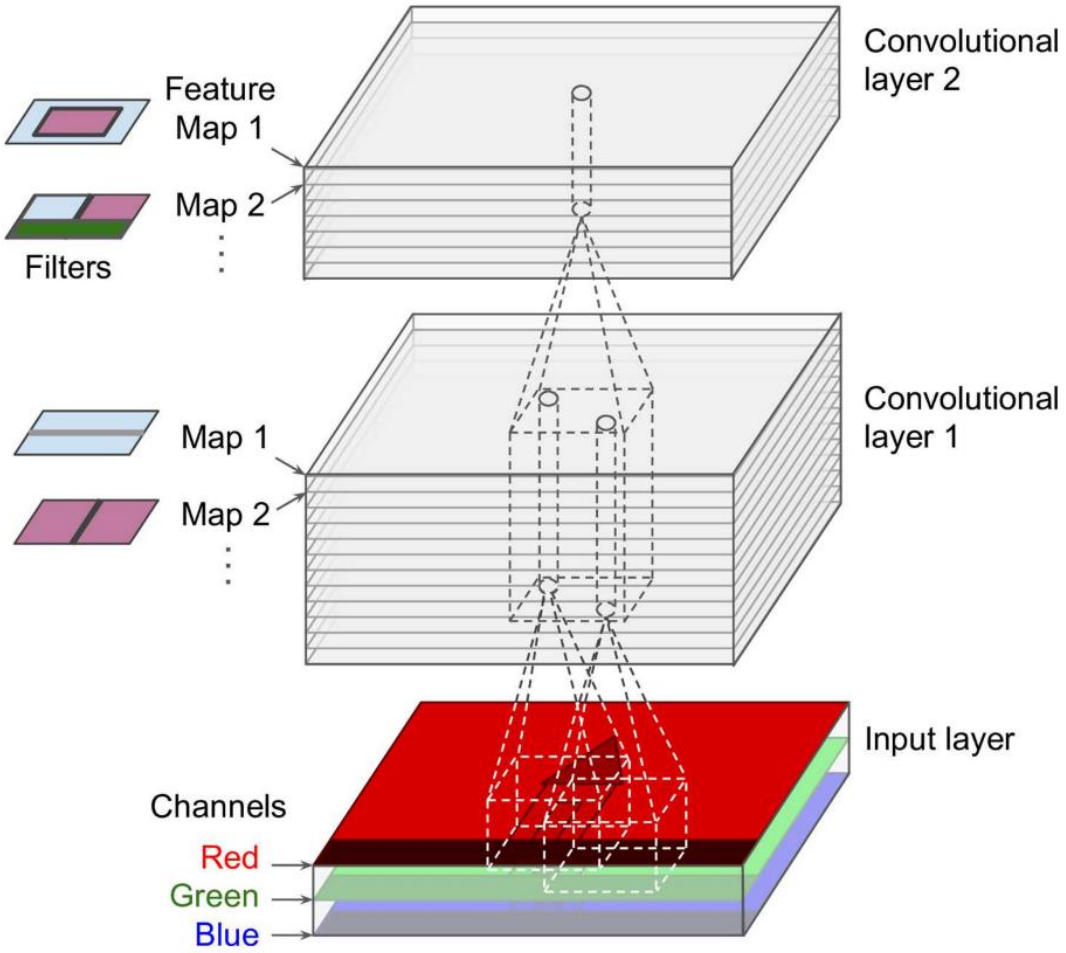


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Local Patterns in Image

- Image can be broken into local pattern such as edges, textures

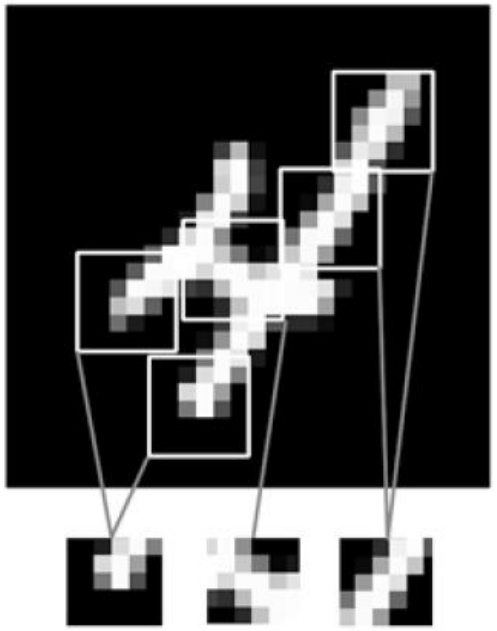


Image from Deep Learning with Python book

CNN Hyper-Parameters

- Number of convolution layers
- Convolution window size
- Convolution filter mask (Filter Depth=K)
- Number of stride
- Padding

Example of random initialized matrices for 32 filter

- The filters to be learnt by CNN

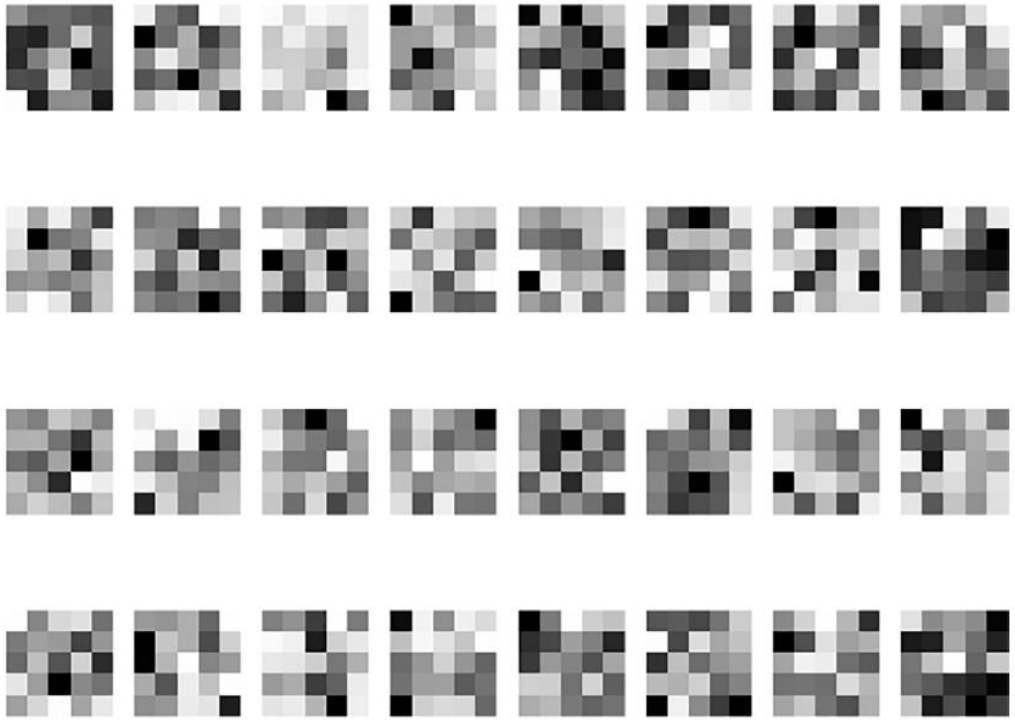
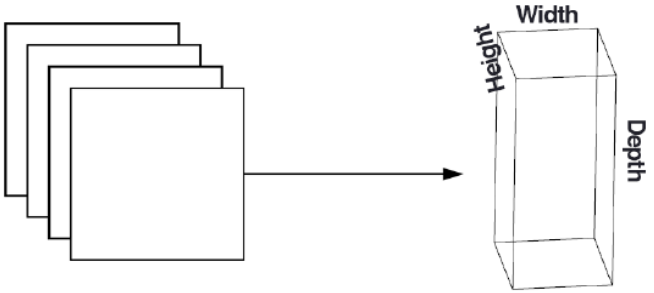


Image from Learning TensorFlow book

K filter/Activation Map

- After we apply K filter, we get the the volume of activation/feature map for the next layer
- Note that the depth can be more than K, as the input can have many channels



K

Filter Results

- We call it Feature Map or Activation Map

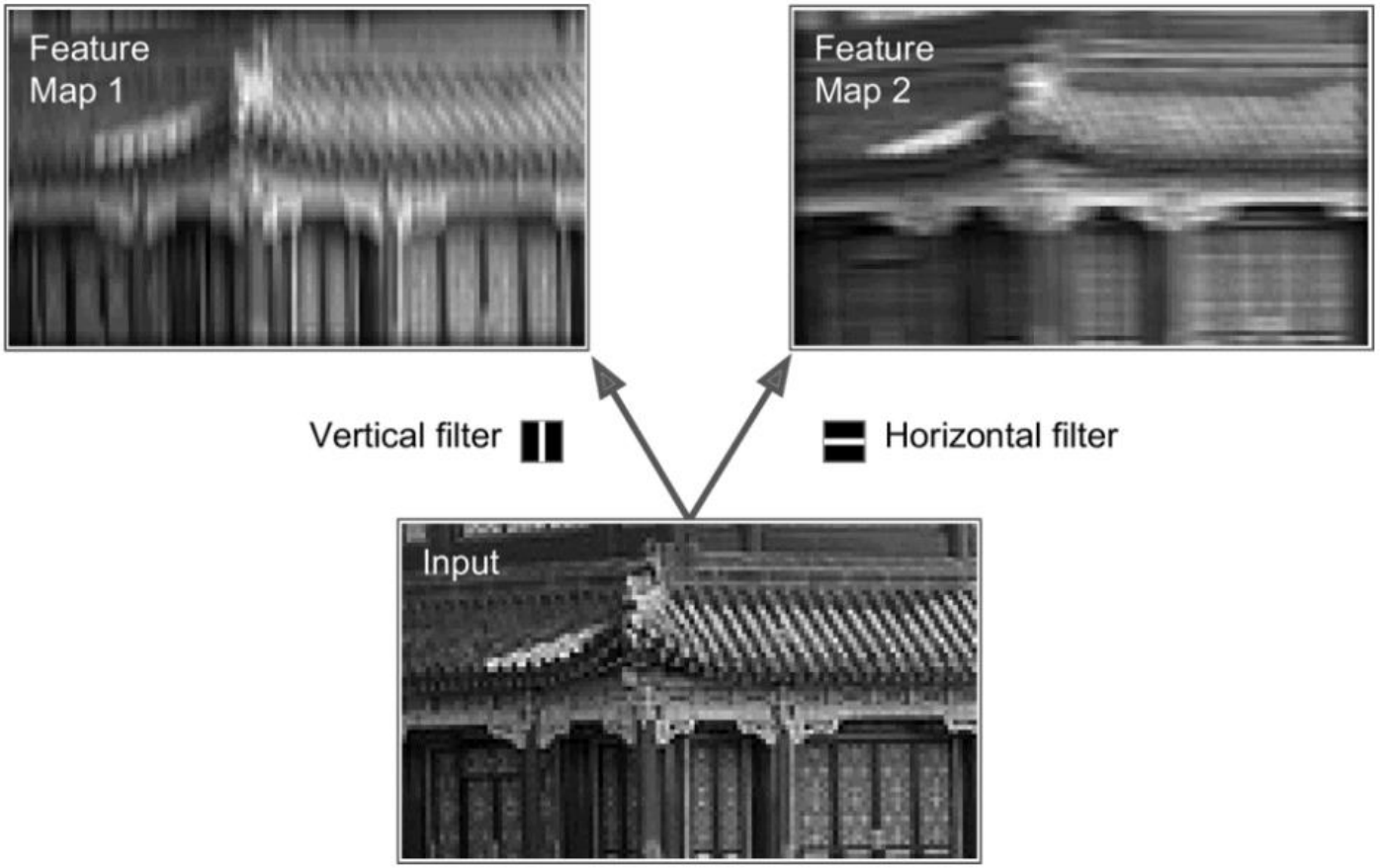
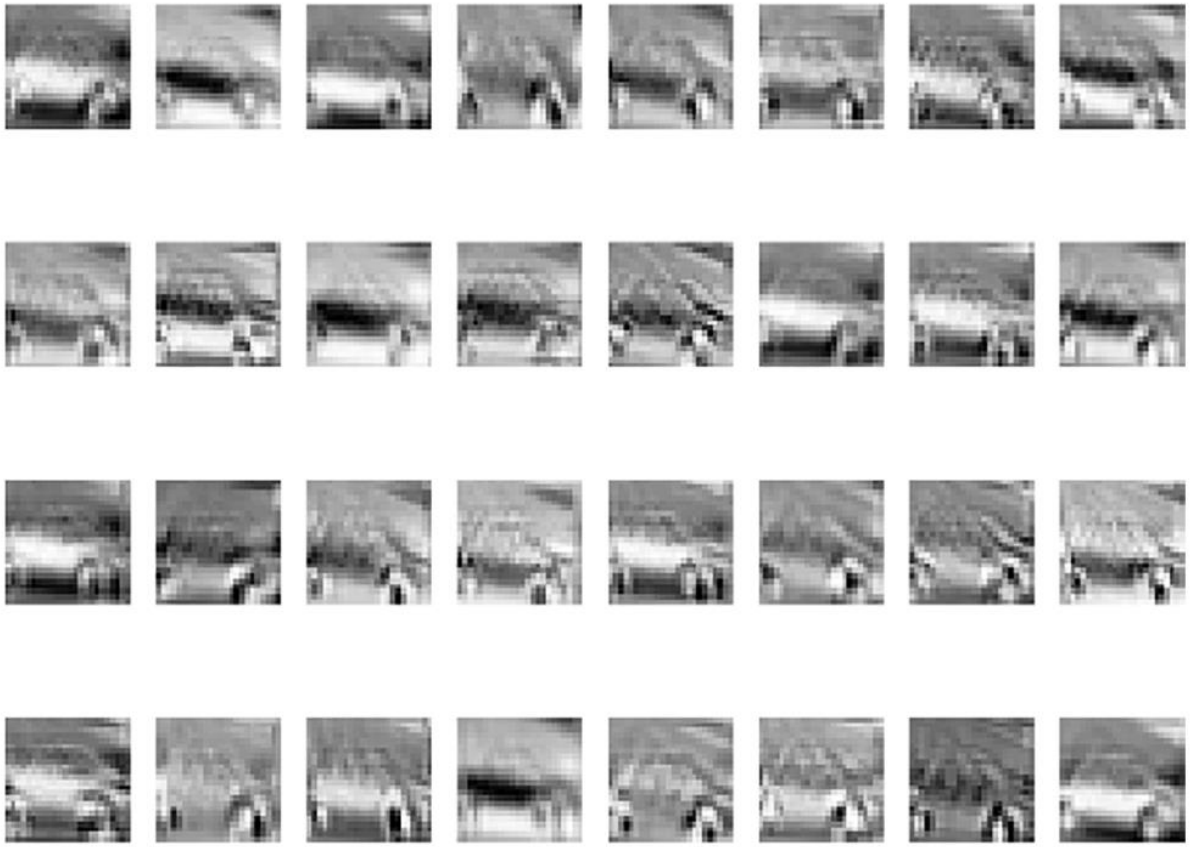
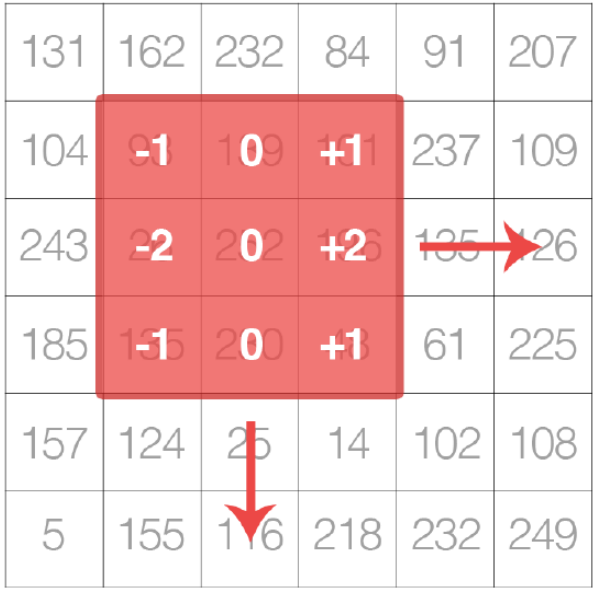


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Filter Result by applying on a car



Stride



- The stride is a step of sliding of small matrix over the image (big matrix)
- Example of image (left) and filter (right)

95	242	186	152	39
39	14	220	153	180
5	247	212	54	46
46	77	133	110	74
156	35	74	93	116

0	1	0
1	-4	1
0	1	0

- Output of a convolution with 1x1 stride (left) and 2x2 stride (right)

692	-315	-6
-680	-194	305
153	-59	-86

692	-6
153	-86

Padding

- Padding is a technique to retain the original image size when applying a convolution
- In Tensorflow framework, only zero padding is provided
- From the figure, top left is the kernel, top right is the image

692	-315	-6
-680	-194	305
153	-59	-86

0	0	0	0	0	0	0
0	95	242	186	152	39	0
0	39	14	220	153	180	0
0	5	247	212	54	46	0
0	46	77	133	110	74	0
0	156	35	74	93	116	0
0	0	0	0	0	0	0

-99	-673	-130	-230	176
-42	692	-315	-6	-482
312	-680	-194	305	124
54	153	-59	-86	-24
-543	167	-35	-72	-297

Padding Example

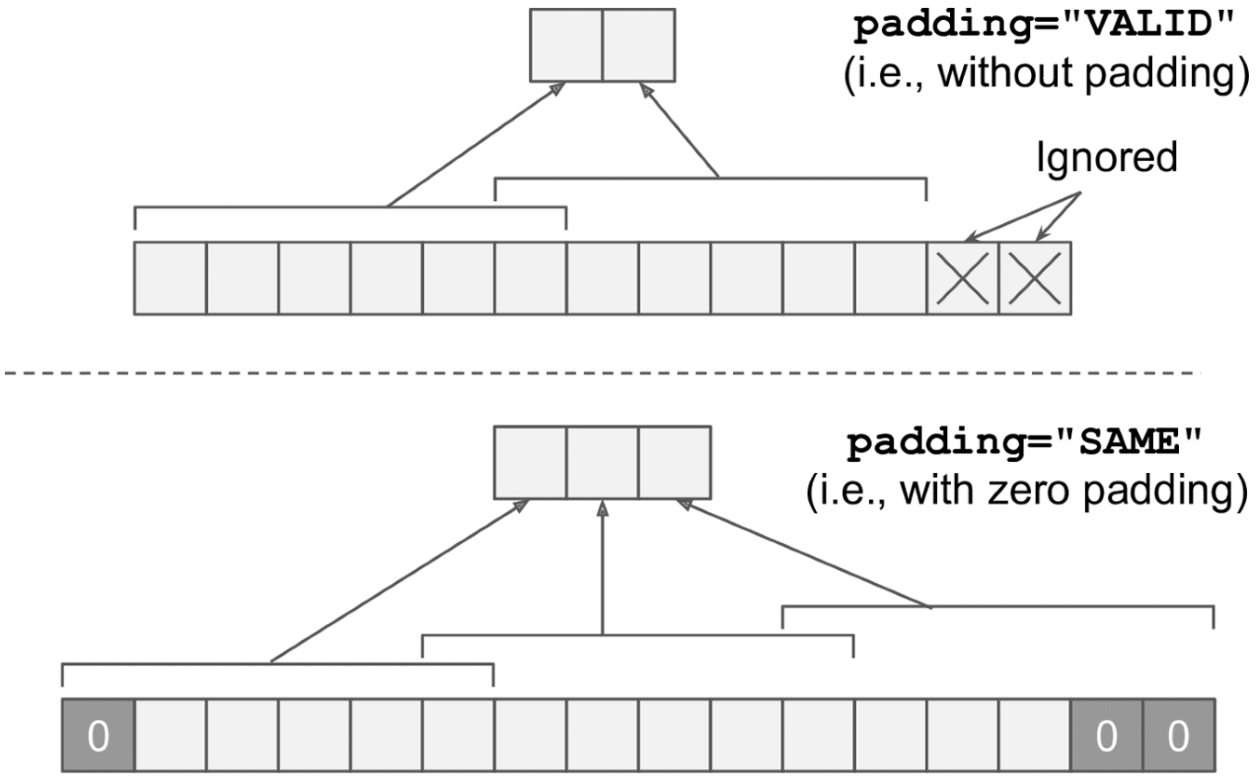


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Output Size

- To guarantee, the integer output size, the following equation can be used to check:

$$((W - F + 2P)/S) + 1$$

When W is the Image size (square)

F is the kernel size (square)

S is stride

P is the padding

Example

- Alexnet model

The image size is 227x227.

The kernel size is 11x11.

No padding.

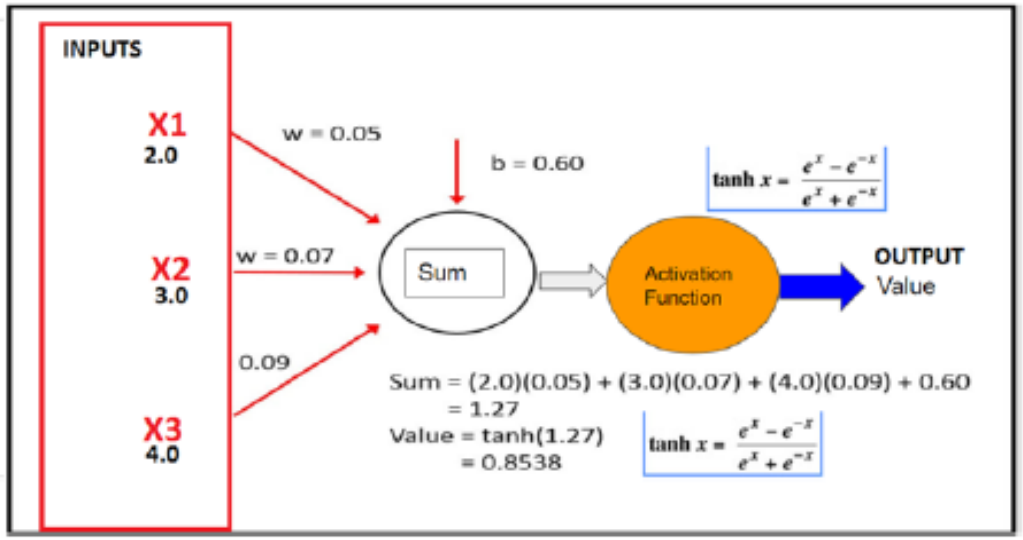
Stride is 4.

We obtain number below which is integer

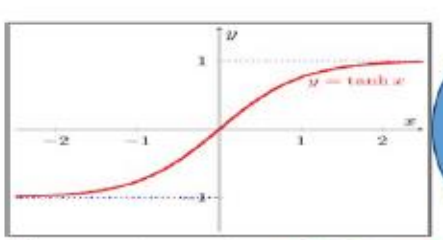
$$((227 - 11 + 2(0))/4) + 1 = 55$$

Activation Layer

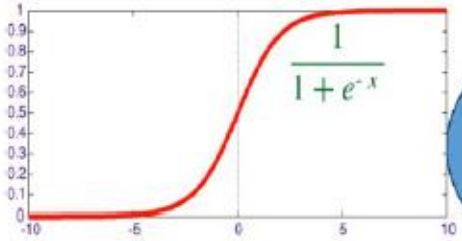
- Activation function is used to introduce non-linearity in the system



- Example of activation function



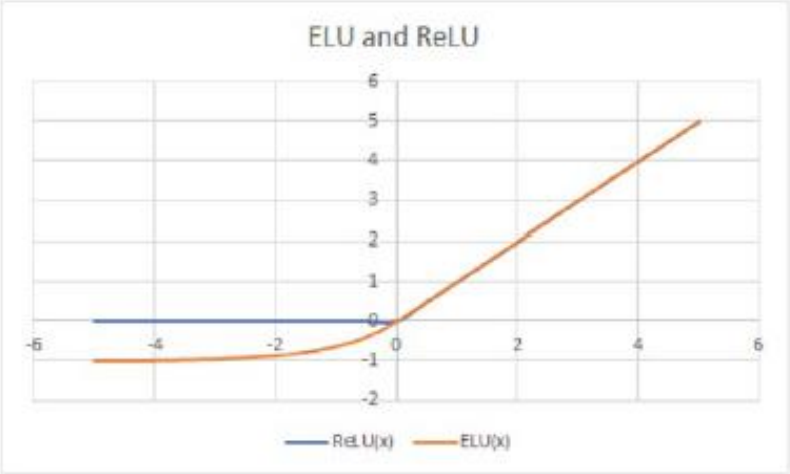
Tangent Hyperbolic: Its output always lies between -1 and 1 no matter what inputs are



Sigmoid--: Its output always lies between 0 and 1 no matter what inputs are

Activation Function

- RELU (Rectifier Linear Unit) and Exponential Linear Unit (ELU)

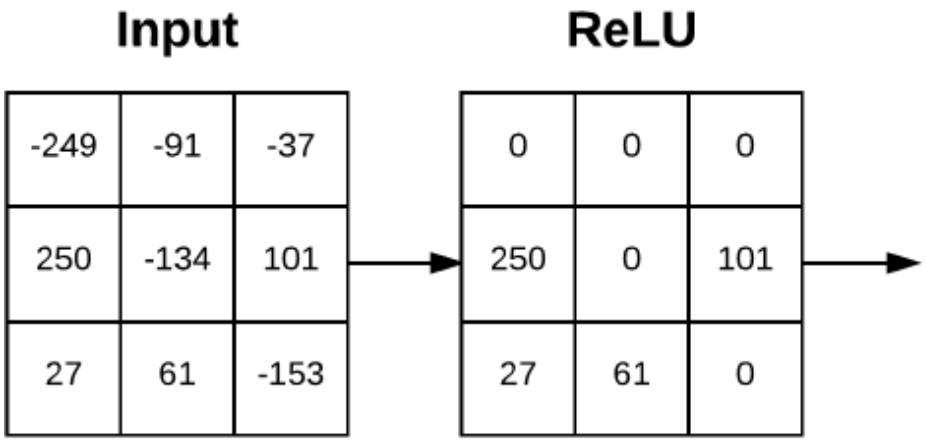


- The Model:

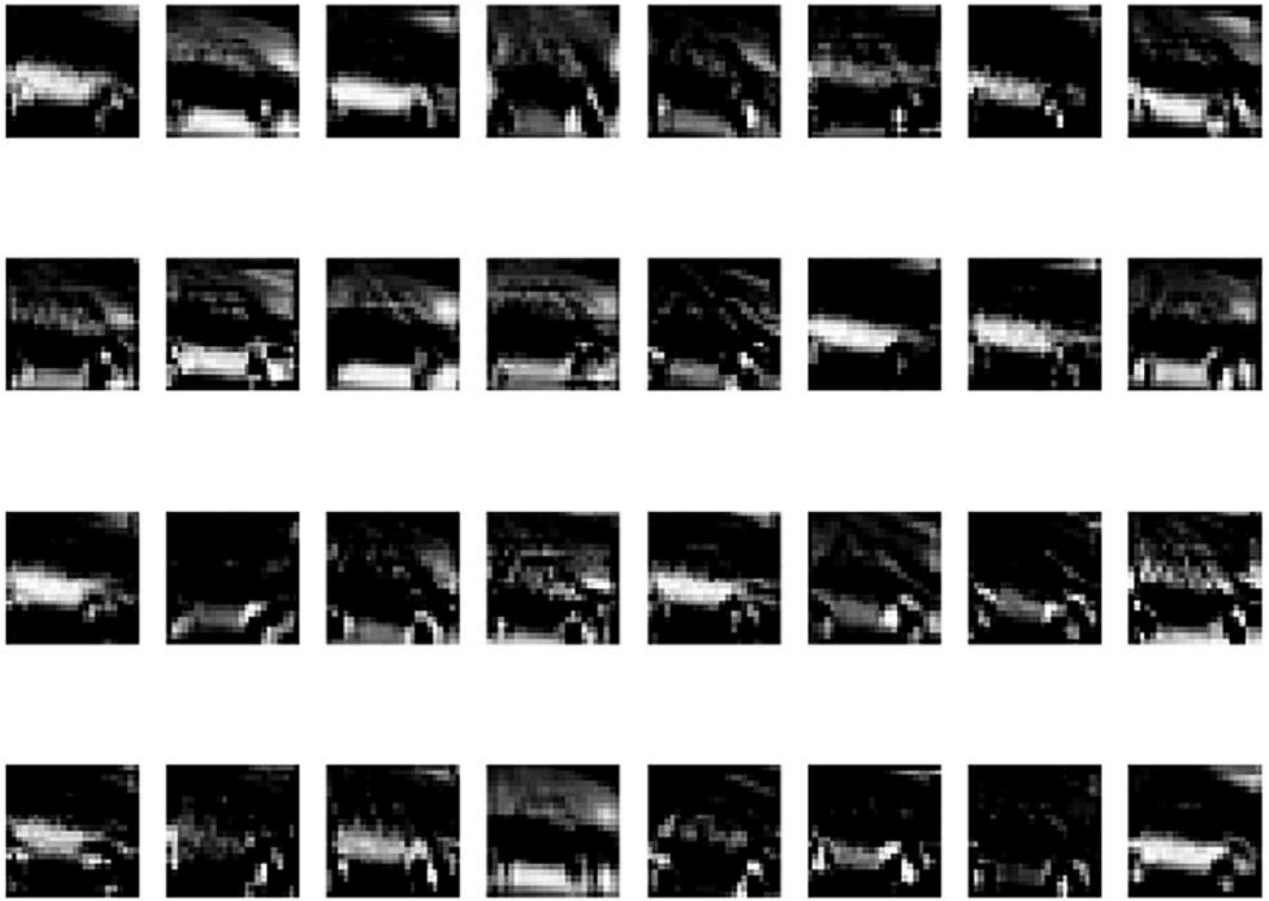
  INPUT => CONV => RELU => FC

Example

- RELU Activation Function



Result after Activation Function



Reduce the Input Size

There are two ways to reduce the input size

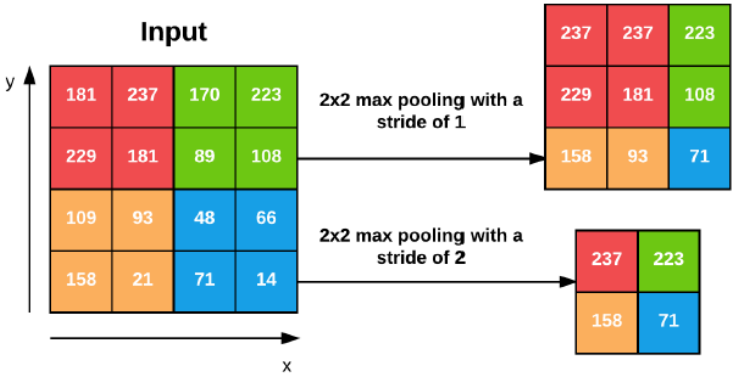
- Convolution with stride > 1

- Pooling layer

Pooling

- The pooling is a process of subsample (shrink the image) to reduce the computation
- There are many functions that can be applied such as max pooling, min pooling, mean pooling

Example


Input
y
x
181 237 170 223
229 181 89 108
109 93 48 66
158 21 71 14
2x2 max pooling with a stride of 1
237 237 223
229 181 108
158 93 71
2x2 max pooling with a stride of 2
237 223
158 71

Pooling Example

- Example of pooling size using a 2x2 kernel with a stride of 2 and no padding

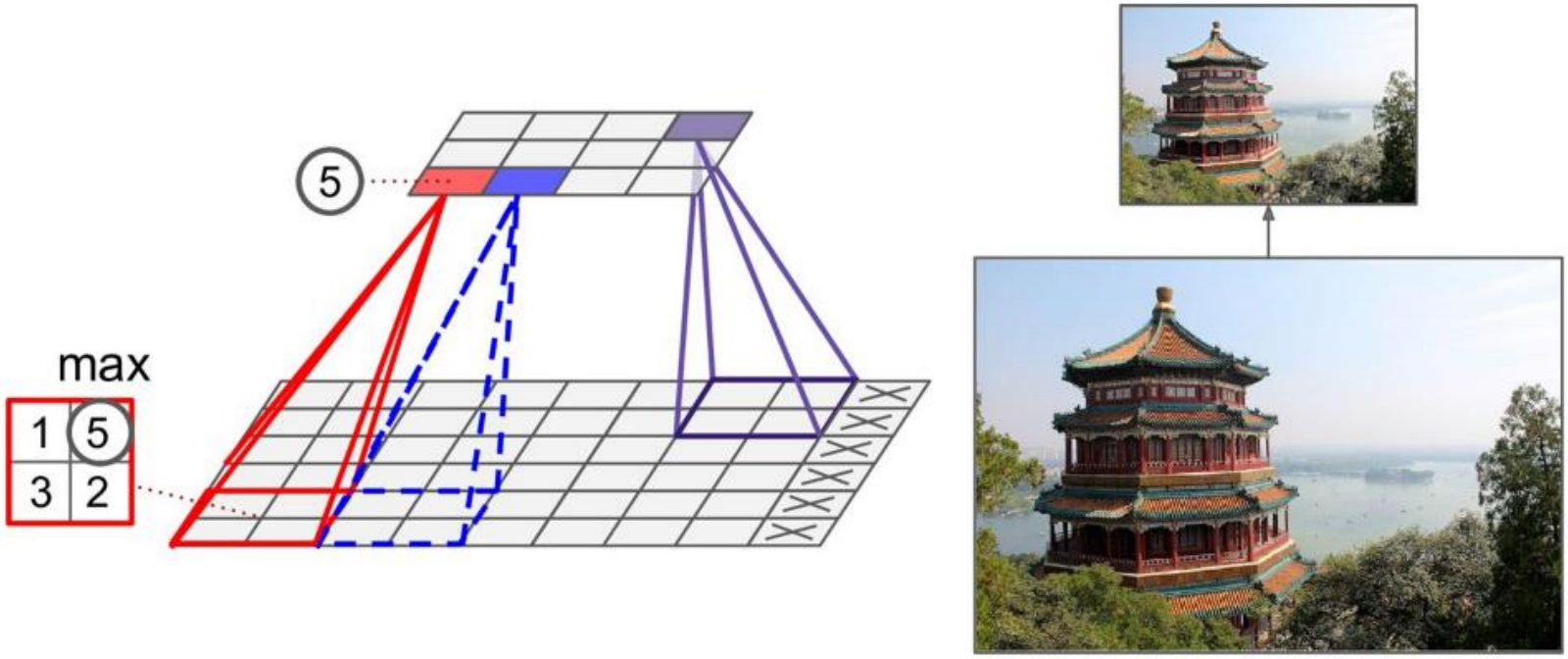


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Pooling Example

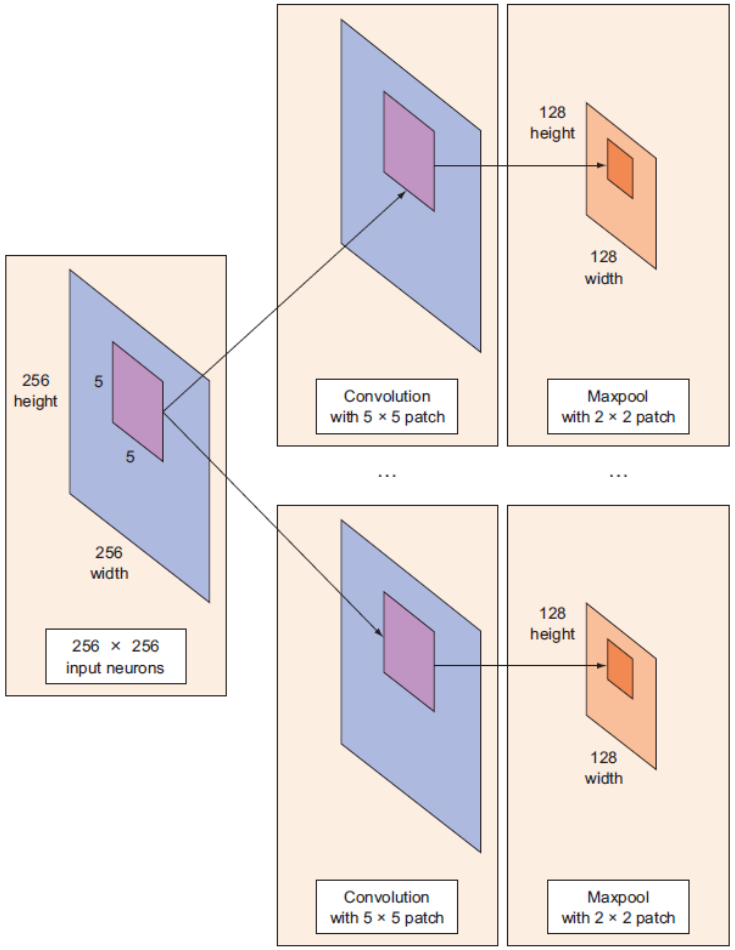
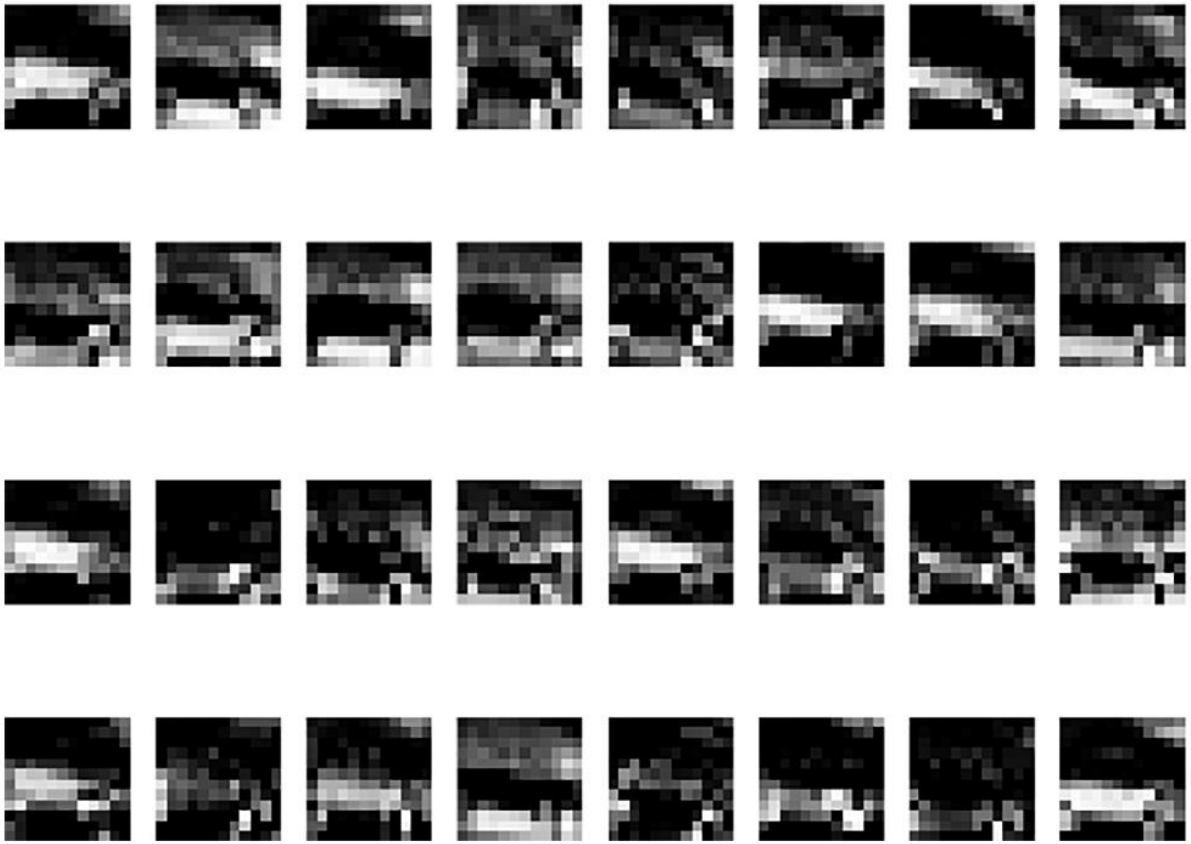


Image from Learning TensorFlow book

Results after Pooling



Model with Pooling

- Example of a model with Pooling:

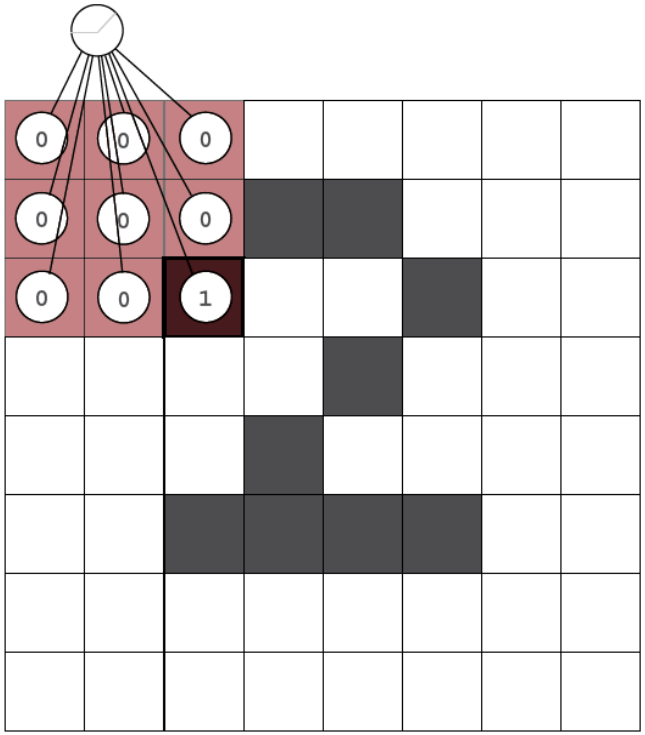
INPUT => CONV => RELU => POOL => FC

To POOL or CONV?

- In 2014 paper, striving for simplicity: The ALL Convolutional Net, Springenberg et al., propose to discard the POOL layer entirely and use CONV layers with a larger stride to handle downsmapling
- Now, it becomes increasingly common trend to not use POOL

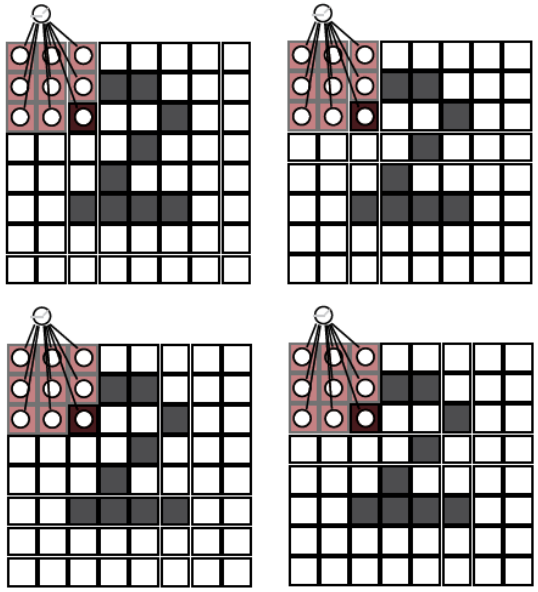
Another way of Pooling

- We can achieve good translation invariant with non-explode weight by using Convoluation Neural Network

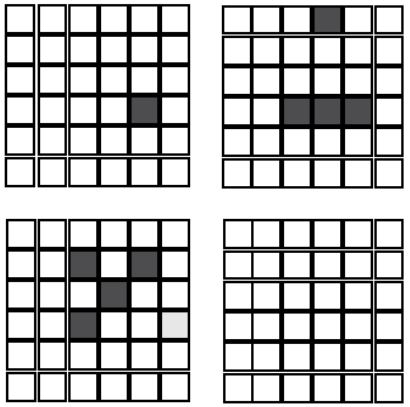


Another way of pooling

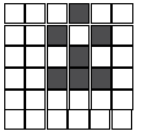
- Convolution is done on many filters automatically
- Pooling can also be used between filters



Four convolutional kernels predicting over the same 2



Outputs from each of the four kernels in each position



The max value of each kernel's output forms a meaningful representation and is passed to the next layer.

Fully Connected Layer

- It is common to use one or two FC layers prior to applying the softmax classifier
- There is also a trend to not use FC layer as it is computing intensive
- Model:

INPUT => CONV => RELU => POOL => FC

Batch Normalization

- It is introduced by Ioffe and Szegedy in 2015 paper, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift to add BN layer
- The idea is to normalize the data where x_i is mini-batch
- The equation is as follow:

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}}$$

when

$$\mu_\beta = \frac{1}{M}\sum_{i=1}^{m} x_i$$

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

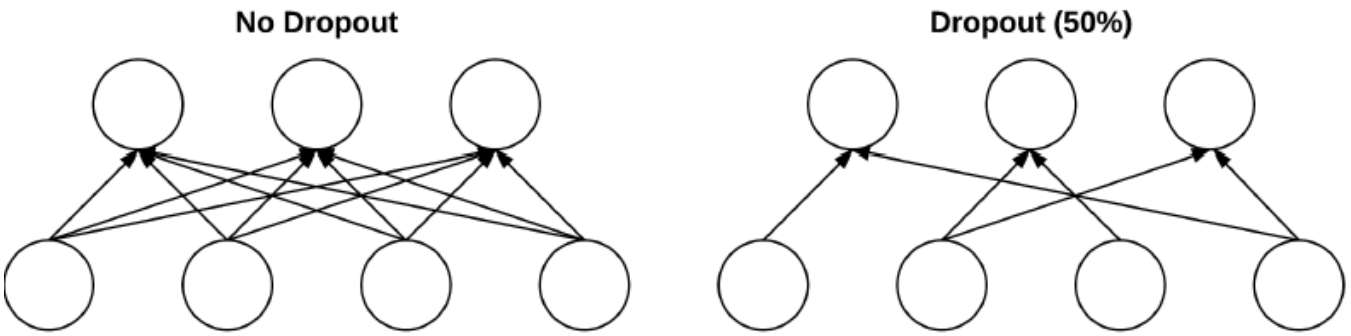
Batch Normalization

- Advantage:
  - Help reduce the number of epochs for training and help for regularization
  - Recommend to put wherever we can
- Drawback:
  - Slow down the system
- Model:

INPUT => CONV => RELU => BN => POOL => FC

Dropout

- The dropout is a regularization technique
- The idea is to keep turn off some neural so that we use many good neural nodes for prediction (not relying only on one)
- It provides multiple redundant nodes



Dropout

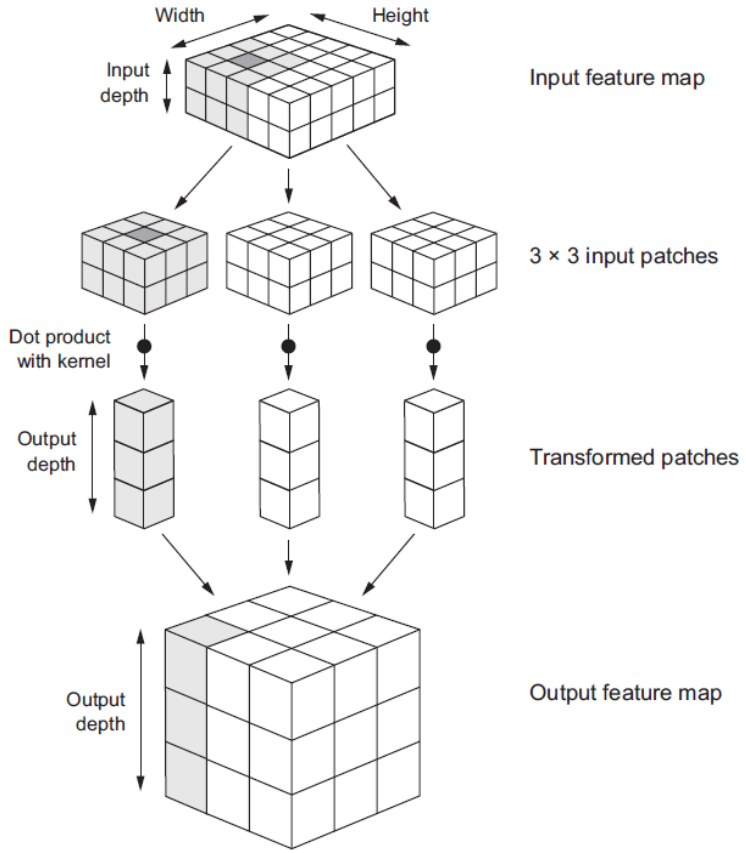
- Note that we randomly add dropout only during the training time, testing time, we activate back all nodes

- Model:

INPUT => CONV => RELU => BN => POOL => FC => DO => FC => DO

Convolution on an image

- How convolution work:



Common mistake is to use kernel that are too large

Image from Deep Learning with Python book

Complete CNN Architecture

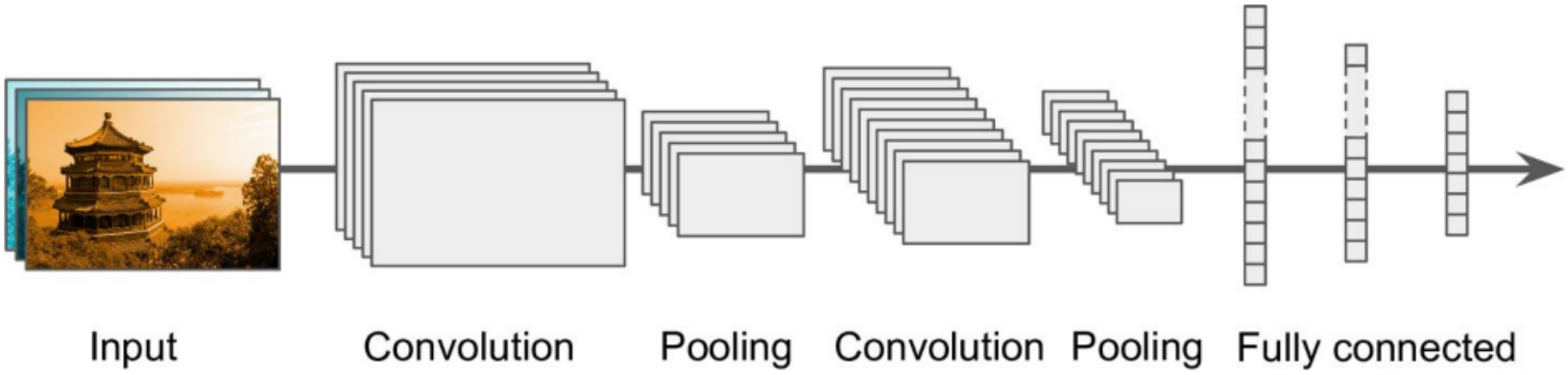


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Four Important Feature for Deep Learning

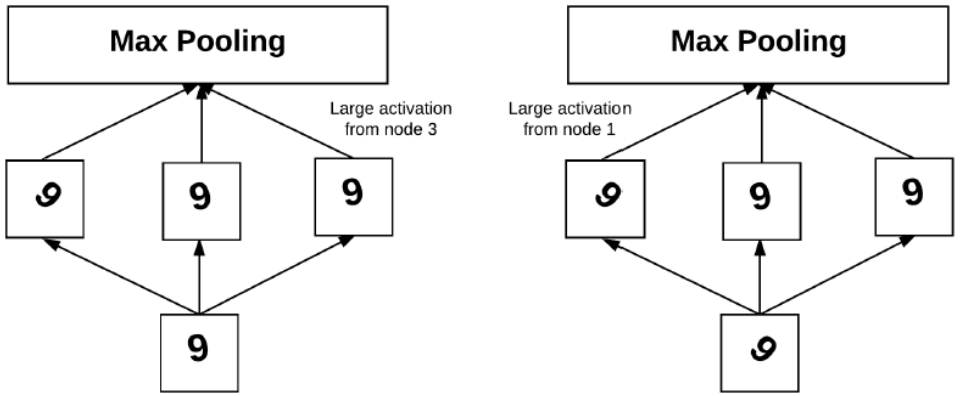
- Dataset
- A Loss Function
- A Neural Network Architecture
- An optimization method

Rules of Thumb

- Common input sizes include 32x32, 64x64, 96x96, 224x224, 227x227 and 229x229
- The input layer should be divisible by two multiple times (to use POOL)
- CONV layers should be small size such as 3x3, 5x5, or 1x1
- Large filter can be used in very first CONV such as 7x7 and 11x11

Is CNN Translation, rotation, and scaling invariant

- CNN is translation invariant with the help of convolutional layer
- It is not rotation and scaling invariant unless you let the network learn a lot of rotation and scaling samples



Training using Keras

```
from keras import layers
from keras import models
from keras.datasets import cifar10
from sklearn.preprocessing import LabelBinarizer

model = models.Sequential()
model.add(layers.Conv2D(64, (5, 5), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
              "dog", "frog", "horse", "ship", "truck"]

model.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

H = model.fit(trainX, trainY, validation_data=(testX, testY),
              batch_size=250, epochs=100, verbose=1)
```

Training on Mnist-cloth

```
import numpy as np
import os

import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn



import matplotlib as mpl
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

```
# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "ann"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

tf.__version__
keras.__version__
```

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_train_full.shape
X_train_full.dtype
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()

y_train
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

```
class_names[y_train[0]]
X_valid.shape
X_test.shape
n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_plot', tight_layout=False)
plt.show()
```

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
keras.backend.clear_session()
np.random.seed(42)
tf.set_random_seed(42)
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
model.layers
model.summary()
from keras.utils import plot_model
```

```
# plot_model(model, "my_fashion_mnist_model.png")
hidden1 = model.layers[1]
hidden1.name
model.get_layer(hidden1.name) is hidden1
weights, biases = hidden1.get_weights()
weights
weights.shape
biases
biases.shape
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
history.params
print(history.epoch)
history.history.keys()
import pandas as pd
```

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
save_fig("keras_learning_curves_plot")
plt.show()
model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
y_pred = model.predict_classes(X_new)
y_pred
np.array(class_names)[y_pred]
y_new = y_test[:3]
y_new
```

```
plt.figure(figsize=(7.2, 2.4))
for index, image in enumerate(X_new):
    plt.subplot(1, 3, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_images_plot', tight_layout=False)
plt.show()
```

Rules of Thumb

- We commonly use a stride of S=1, unless we want to do use CONV instead of POOL
- Zero padding should always be applied
- At a novice, POOL is easier to use. Once, you get enough experience, try to avoid it
- POOL should be used with max pooling with 2x2 size and stride =2
- BN and DO should be applied if possible

Questions?