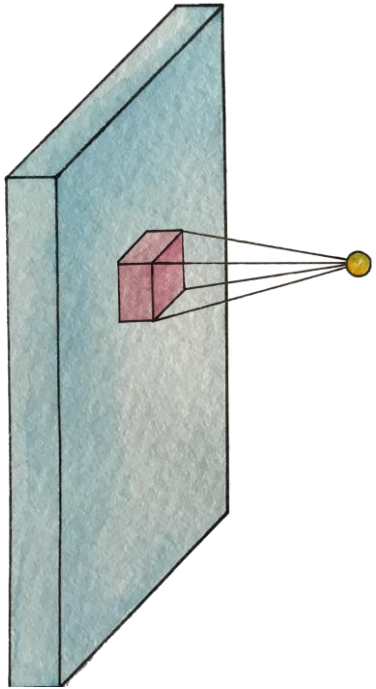
TF2-08

CNNs

Dong Kook Kim

Convolution layer and max pooling



Single depth slice

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

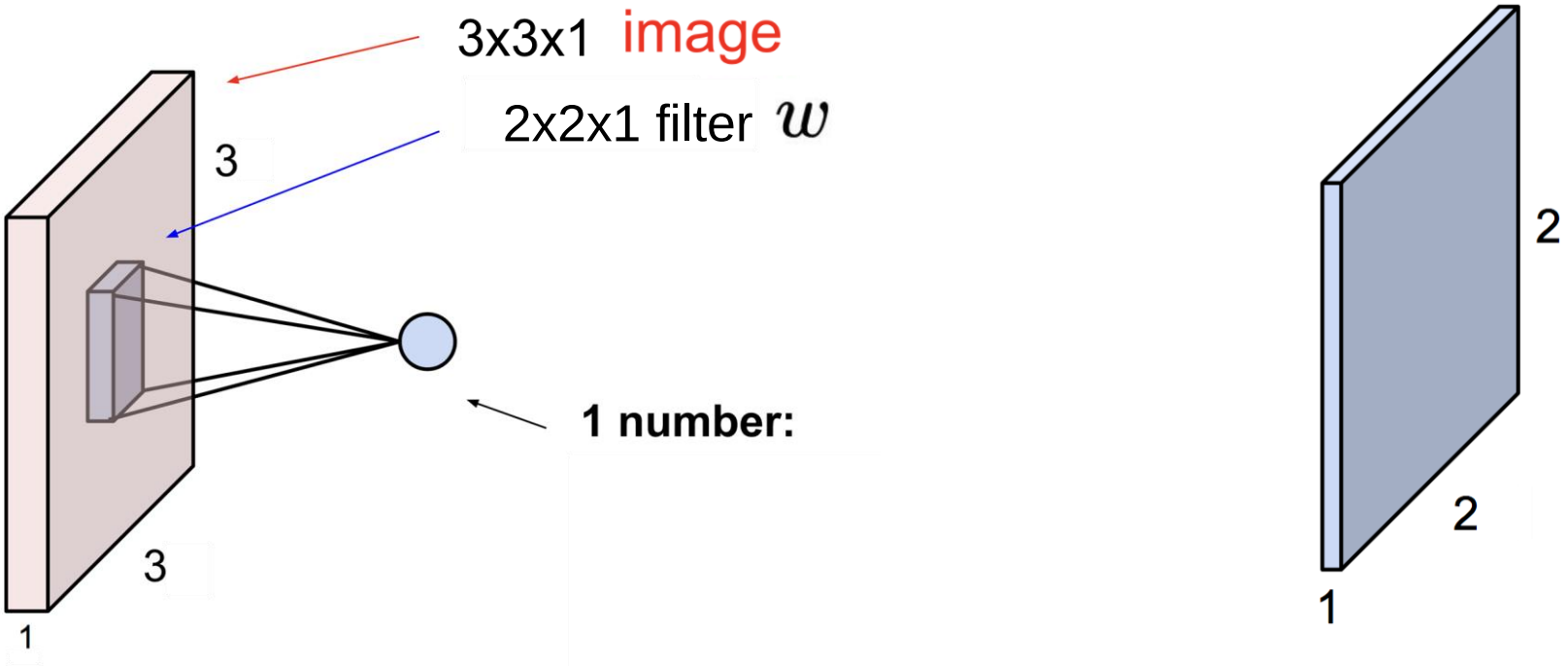
y

max pool with 2x2 filters
and stride 2

6	8
3	4

Simple convolution layer

`Stride: 1x1`



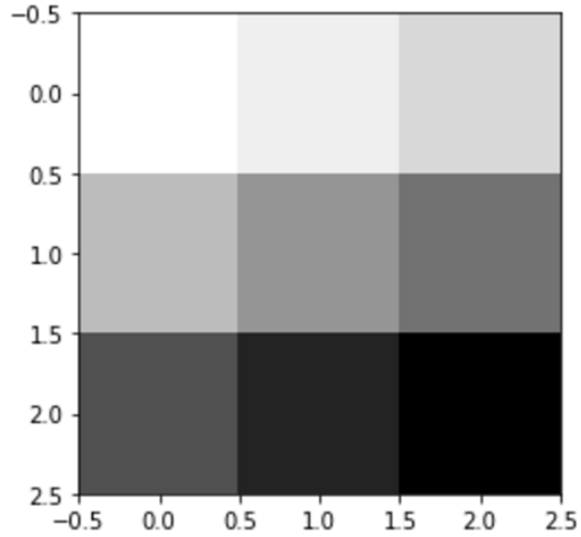
Exercise 08-1.

tf2-08-1-cnn_basics.py

In [2]:
```
sess = tf.InteractiveSession()
image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print(image.shape)
plt.imshow(image.reshape(3,3), cmap='Greys')
```

```
(1, 3, 3, 1)
```

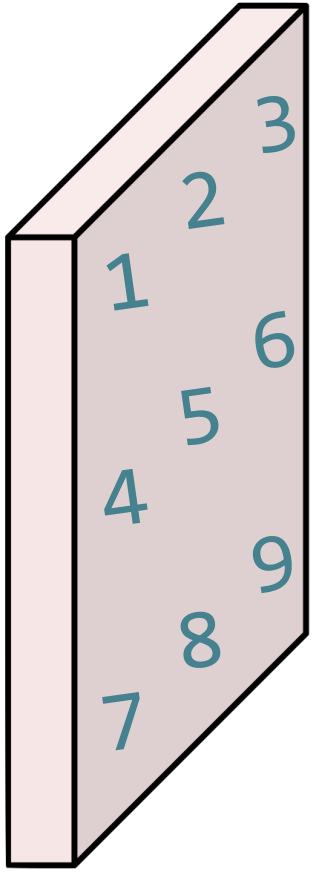
Out[2]:
```
<matplotlib.image.AxesImage at 0x10db67dd8>
```

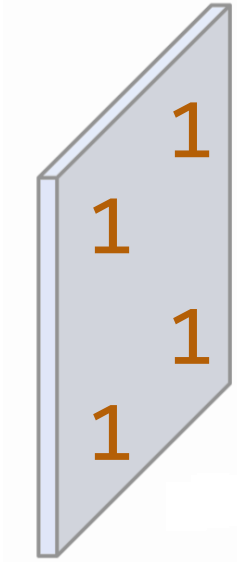


Toy image

Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID






```
[[[[1.]],[[1.]]],
 [[[1.]],[[1.]]]]
shape=(2,2,1,1)
```

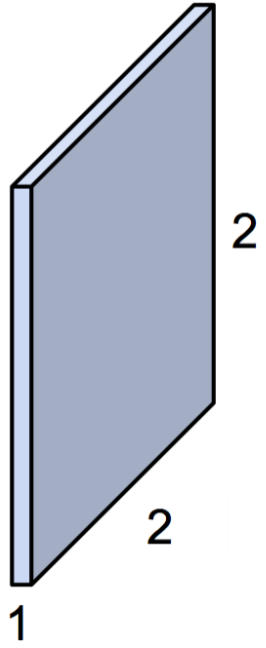
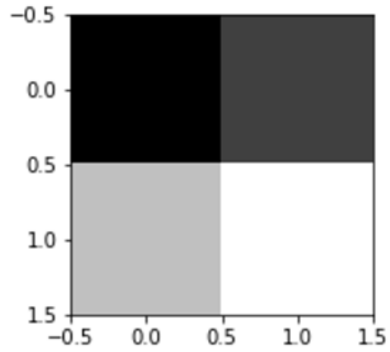
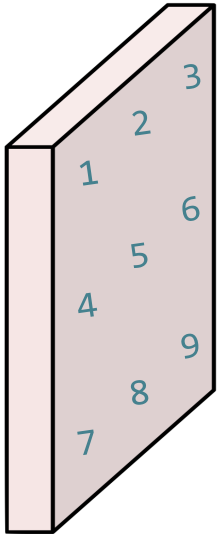


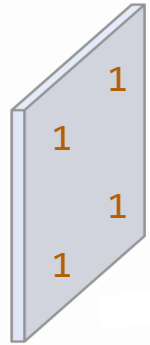
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID

```
# print("imag:\n", image)
print("image.shape", image.shape)
weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 2, 2, 1)
[[ 12.  16.]
 [ 24.  28.]]
```

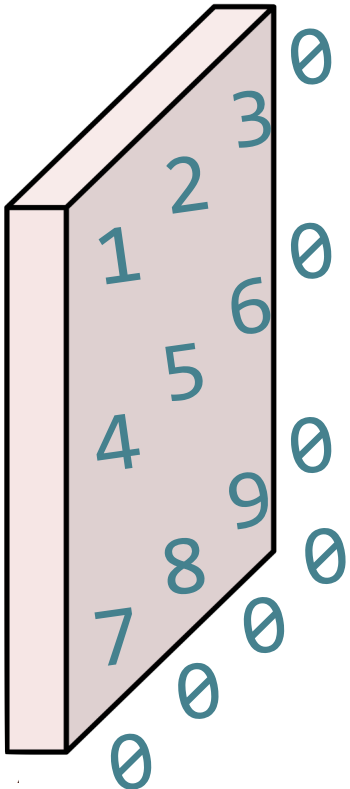


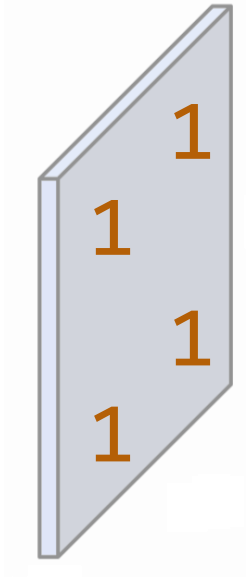




Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: **SAME**





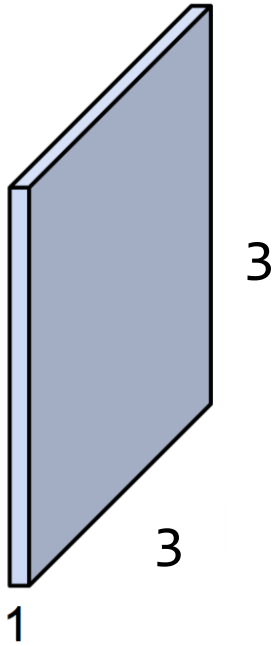
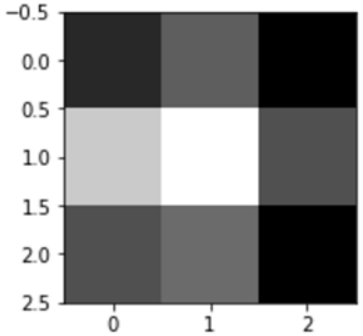


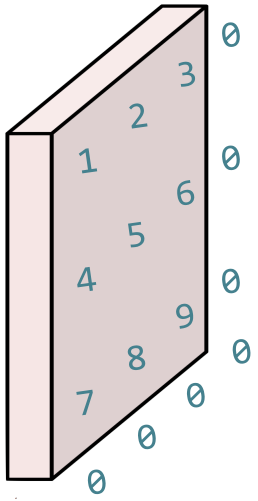
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: SAME

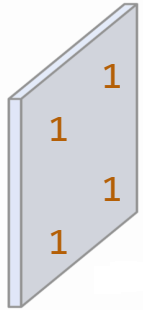
```
# print("imag:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 3, 3, 1)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
```





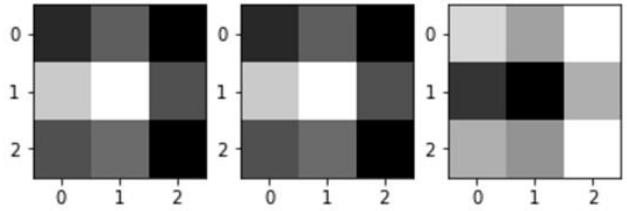


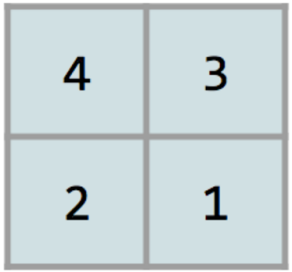
3 filters (2,2,1,3)

```
# print("imag:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.,10.,-1.]],[[1.,10.,-1.]]],
                      [[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
conv2d_img.shape (1, 3, 3, 3)
[[ 12.  16.   9.]
 [ 24.  28.  15.]
 [ 15.  17.   9.]]
[[ 120.  160.   90.]
 [ 240.  280.  150.]
 [ 150.  170.   90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```





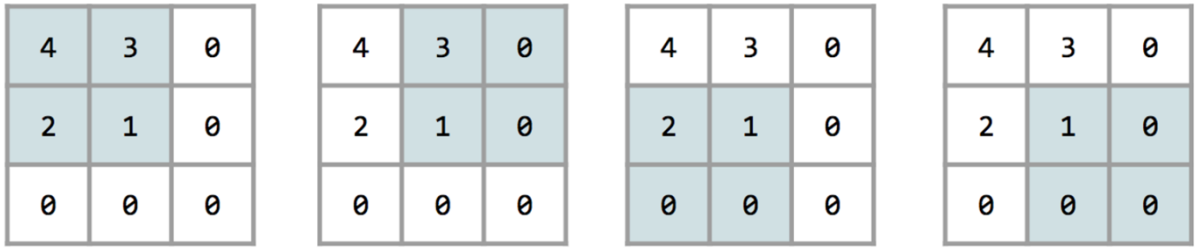
Max Pooling

In [19]:
```
image = np.array([[[[4],[3]],
                   [[2],[1]]]], dtype=np.float32)
pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
                      strides=[1, 1, 1, 1], padding='SAME')
print(pool.shape)
print(pool.eval())
```

```
(1, 2, 2, 1)
[[[[ 4.]
   [ 3.]]

  [[ 2.]
   [ 1.]]]]
```

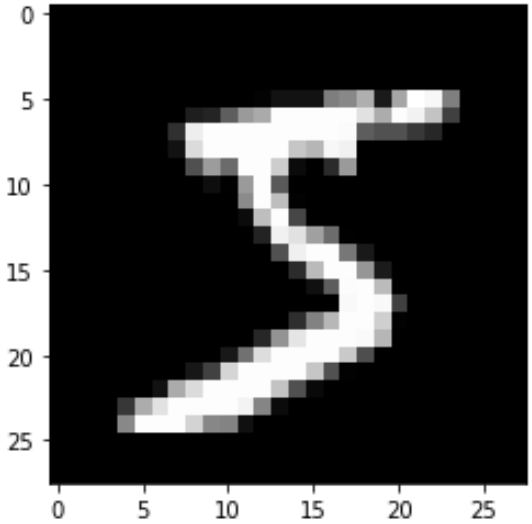
SAME: Zero paddings

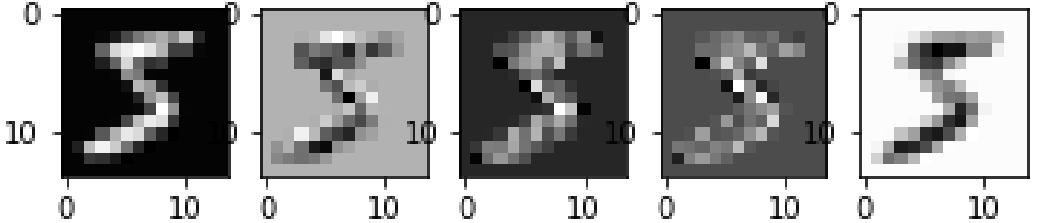


https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-11-0-cnn_basics.ipynb

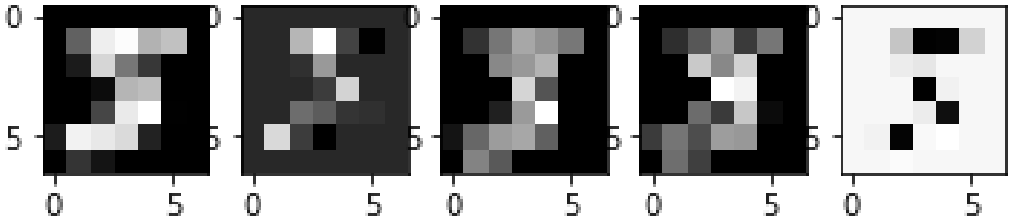
MNIST image

Original image





Convolution layer



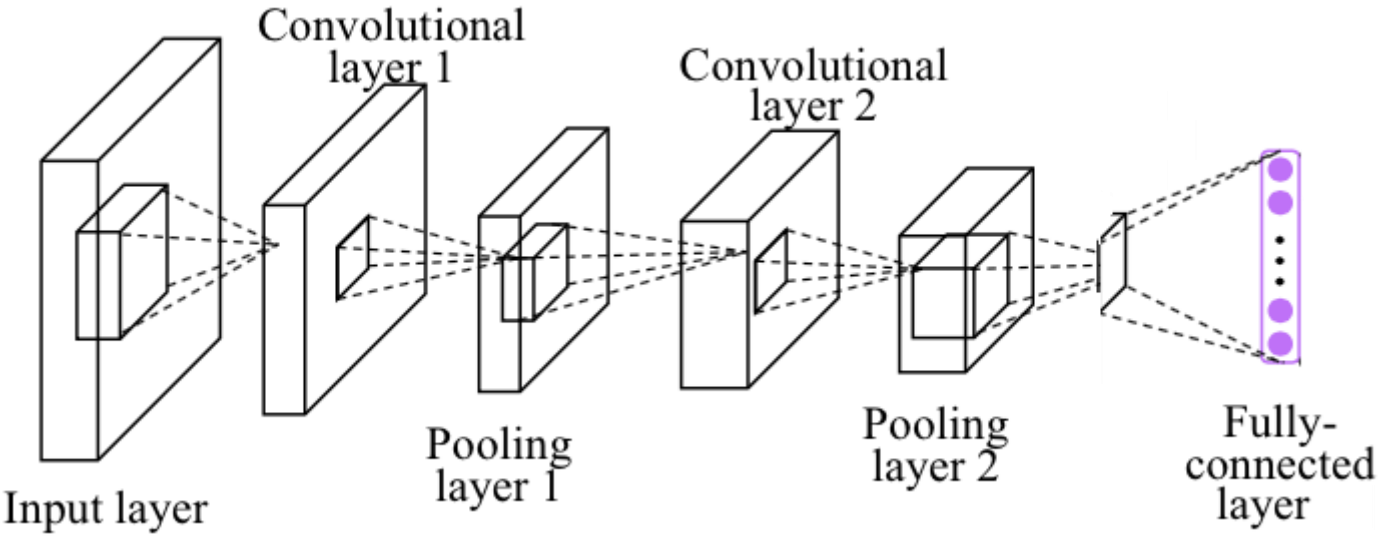
Max pooling

TF2 08-2

CNN MNIST: 99%!

Dong Kook Kim

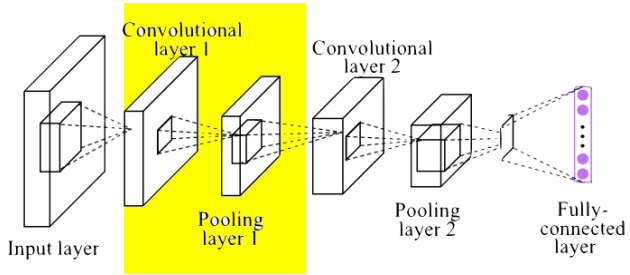
Simple CNN


Convolutional layer 1
Convolutional layer 2
Input layer
Pooling layer 1
Pooling layer 2
Fully-connected layer

Exercise 08-2.

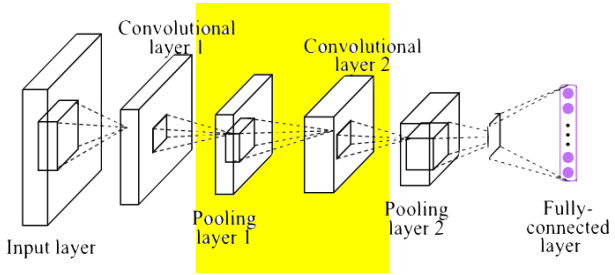
tf2-08-2-mnist_cnn.py

Conv layer 1



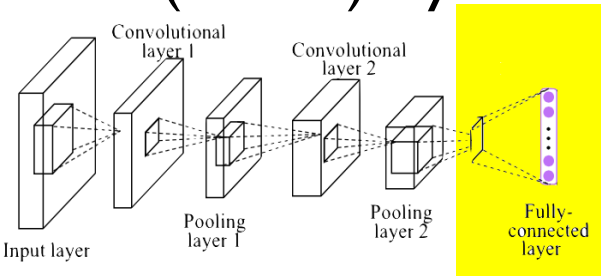

```
model.add(Conv2D(32, kernel_size, padding='same',
 input_shape=(X_train.shape[1],X_train.shape[2],1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
```

Conv layer 1



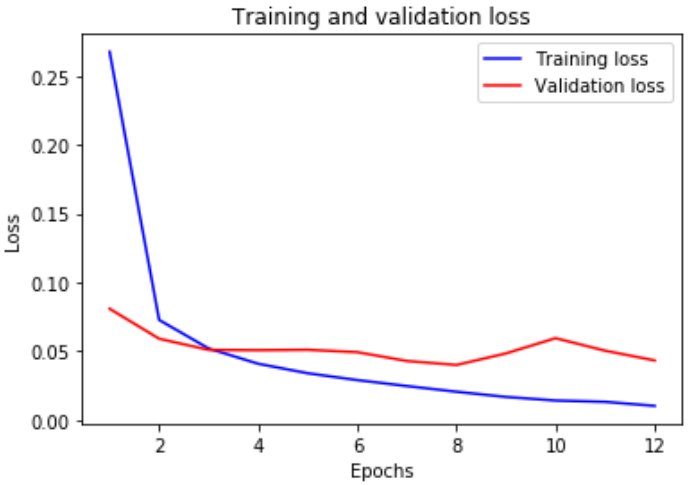

```
model.add(Conv2D(64, kernel_size, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
```

Fully Connected (Dense) layer

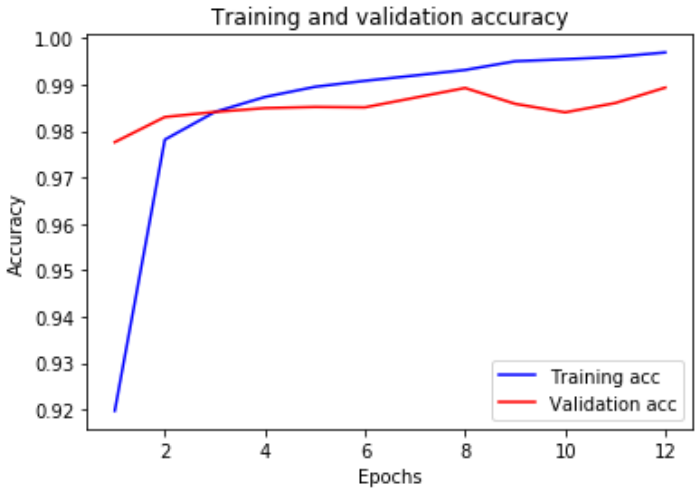



```
model.add(Flatten())
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Training/Validation Loss



Training/Validataion Accuracy



Test Accuracy

99.0 %

Deep CNN

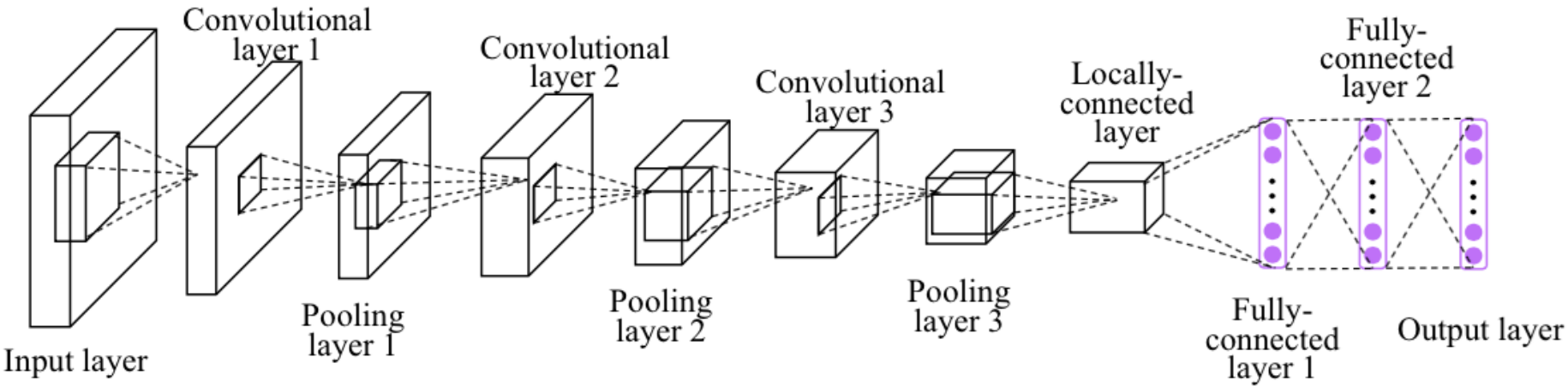
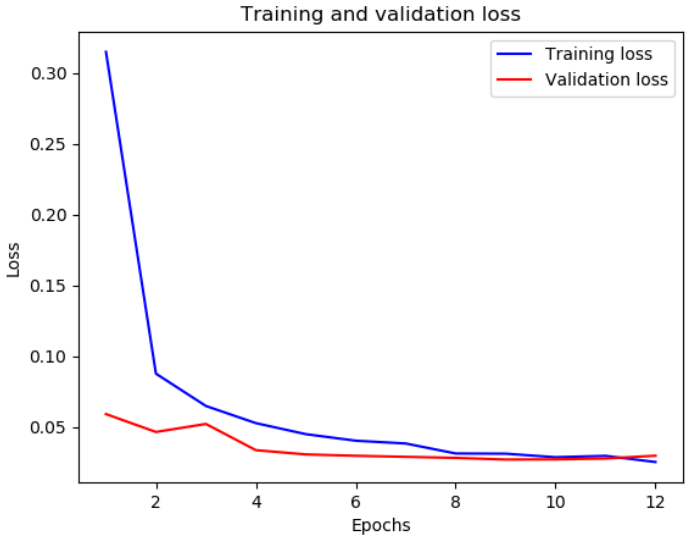


Image credit: http://personal.ie.cuhk.edu.hk/~ccloy/project_target_code/index.html

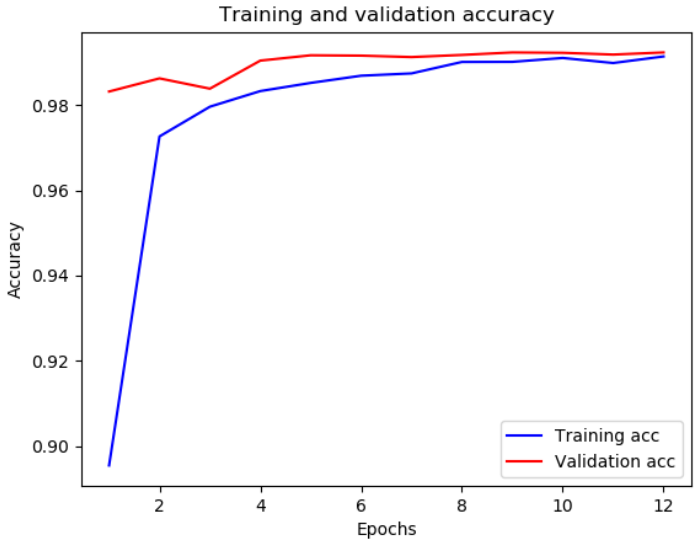
Exercise 08-3.

tf2-08-3-mnist_deep_cnn.py

Training/Validation Loss



Training/Validataion Accuracy

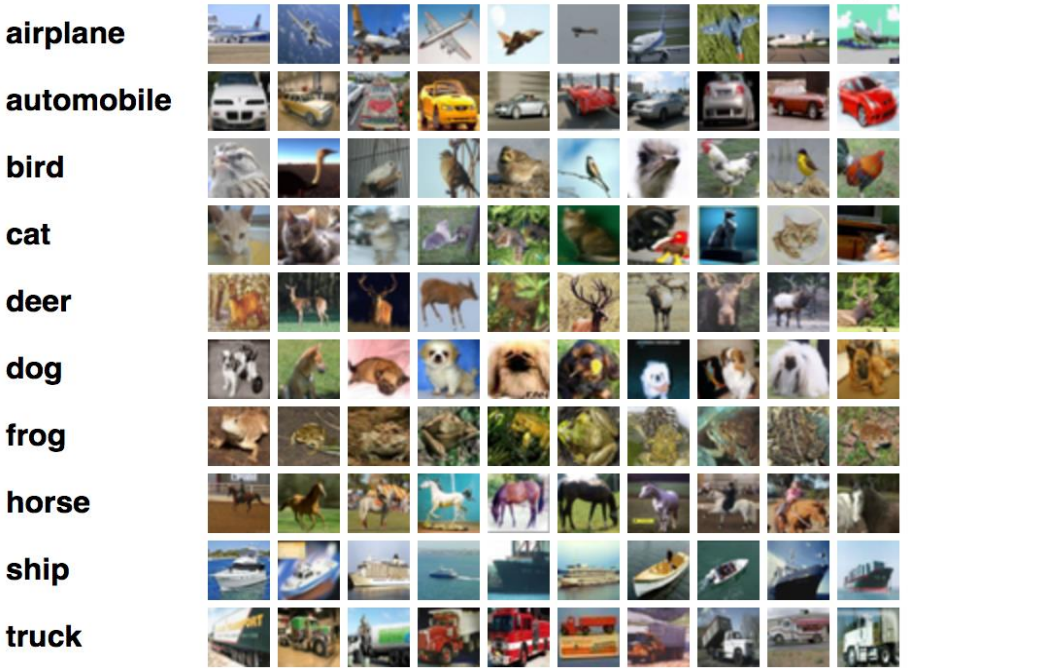


Test Accuracy

99.24 %

CIFAR 10

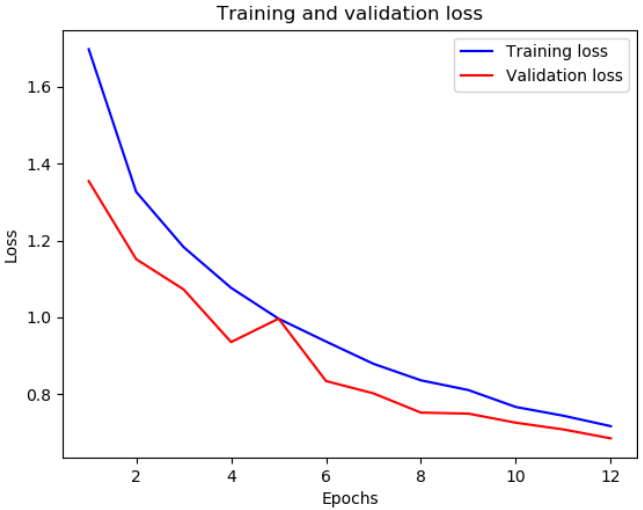
Here are the classes in the dataset, as well as 10 random images from each



Exercise 08-4.

tf2-08-4-cifar10_cnn.py

Training/Validation Loss



Training/Validataion Accuracy

Test Accuracy

75.88 %

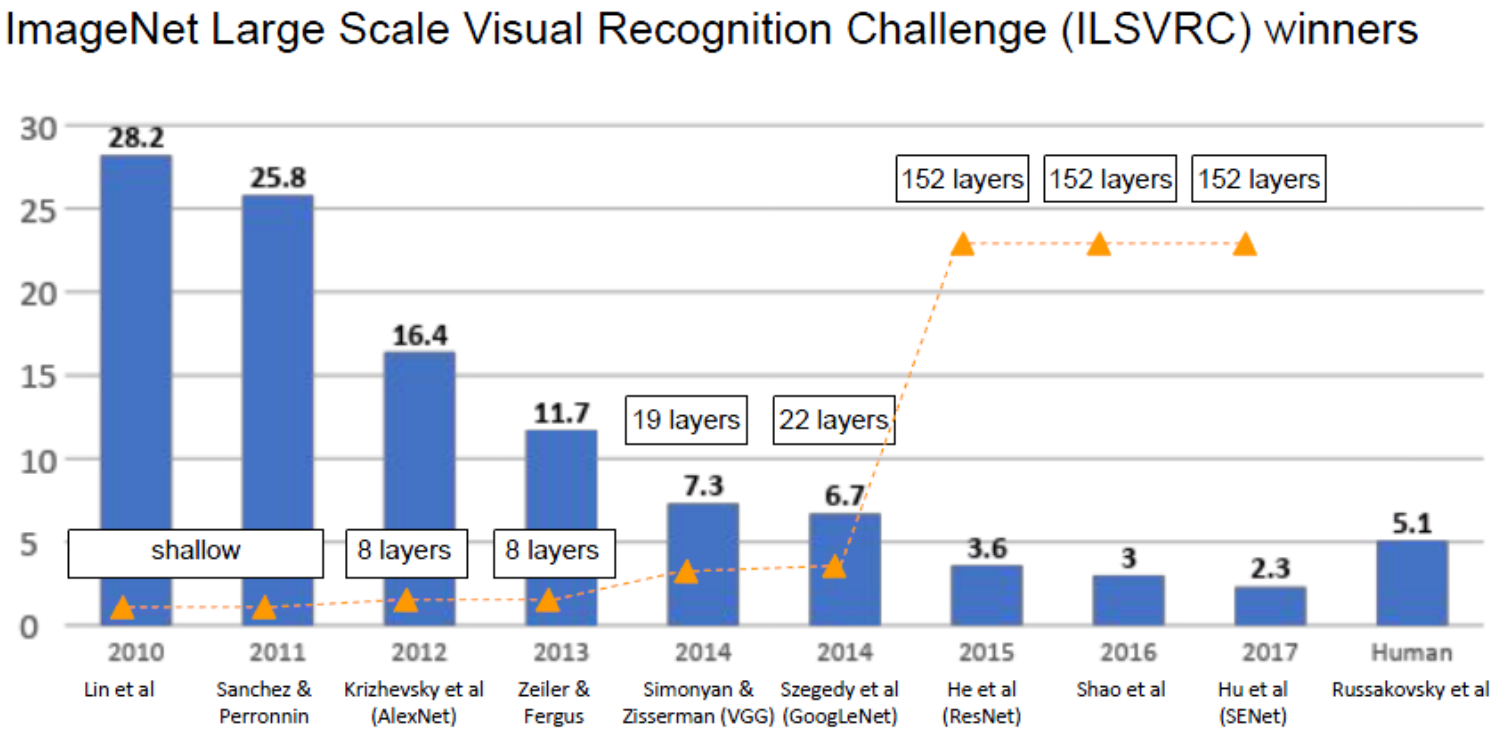
CIFAR-10 : More CNNs

Models	Results	Ref
CIFAR-10 CNN	79%	https://keras.io/examples/cifar10_cnn/
CIFAR-10 CNN-Capsule	83.79%	https://keras.io/examples/cifar10_cnn_capsule/
CIFAR-10 CNN With augmentation	78.70%	https://keras.io/examples/cifar10_cnn_tfaugment2d/
CIFAR-10 ResNet	92.65%	https://keras.io/examples/cifar10_resnet/

Imagenet and CNNs

22K categories, 14M images

Imagenet and CNNs



Keras Models for Imagenet

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

https://keras.io/applications/

Exercise 08-5.

tf2-08-5-cnn_models.py

Transfer Learning and CNN Visualization

see Ch 5.
in 캐라스창시자에게 배우는 딥러닝