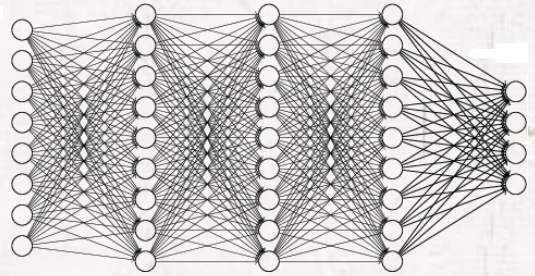
Convolutional Neural Networks

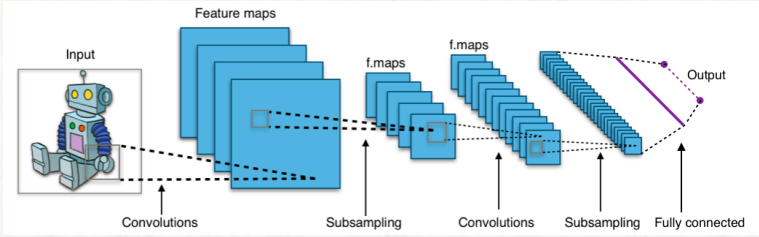
J.H. Lee, Dept. of Software, Sungkyunkwan Univ.

Convolutional Neural Network

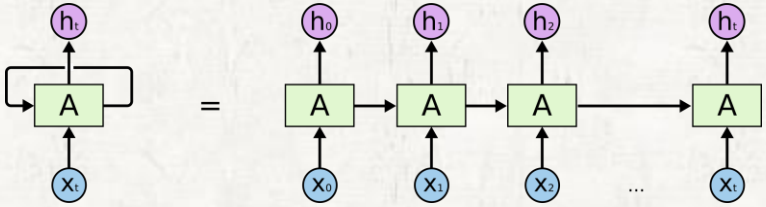
- Type of Deep Neural Network
  - Dense Network (=Fully-connected Neural Network)
  - **Convolutional Neural Network**
  - Recurrent Neural Network
  - …



<Dense Network>



<Convolutional Neural Network>



<Recurrent Neural Network>

Convolutional Neural Network

- What is the Convolutional Neural Network?
  - Consists of Convolution, Pooling layers

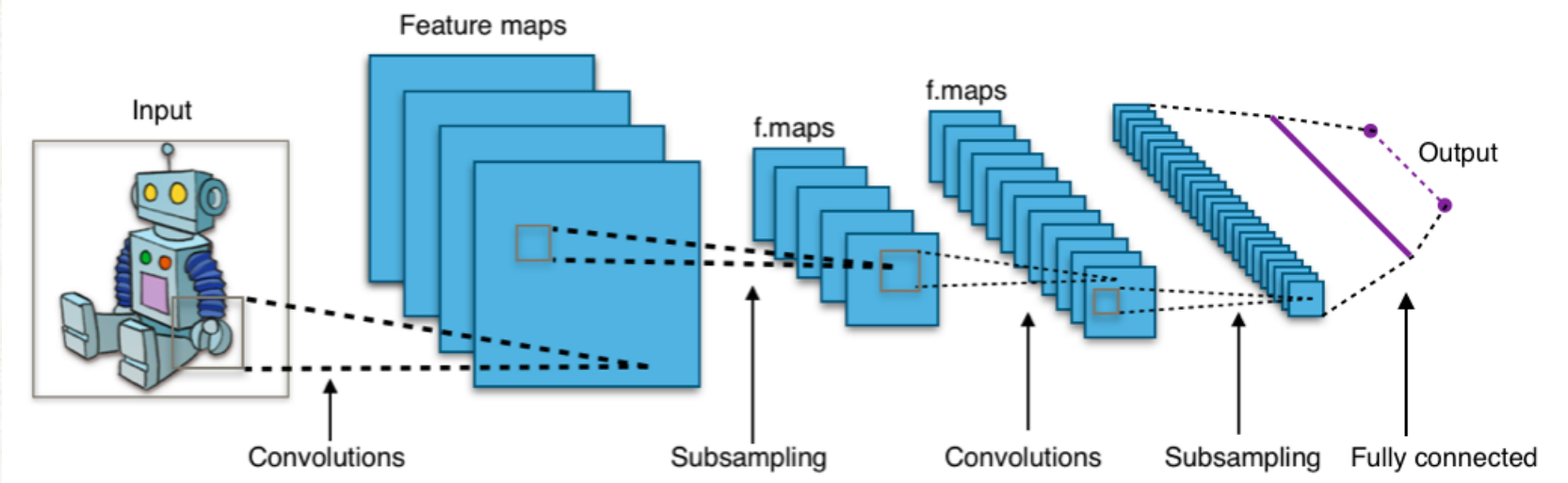
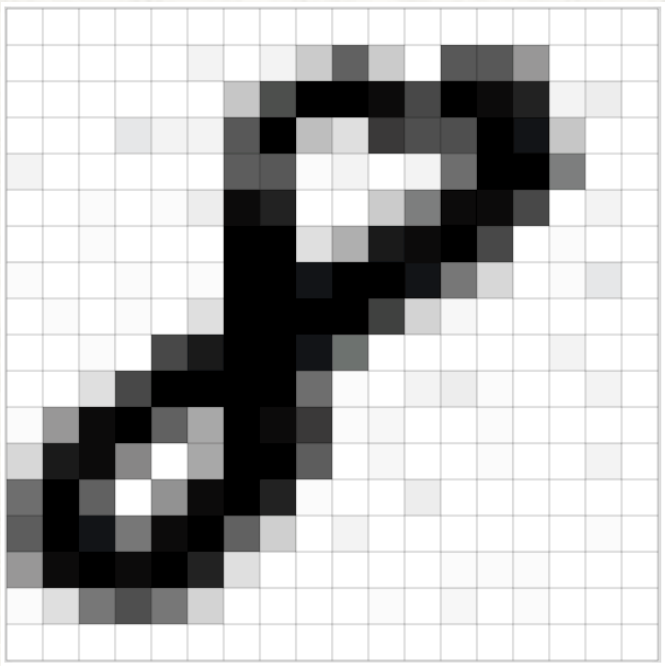


Image data

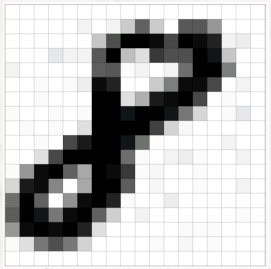
- How to perceive on computer



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Image data

- How to perceive on computer
  - DNN(Dense Neural network)



```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
 0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

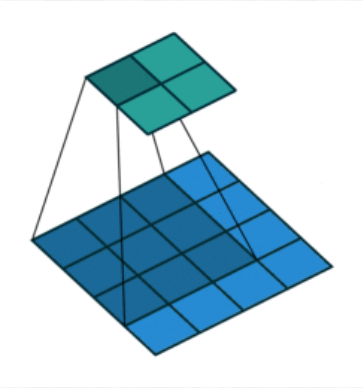
  - CNN(Convolutional Neural network)

```
  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  12   0  11  39 137  37   0 152 147  84   0   0   0   0   0
  0   0   1   0   0   0  41 160 250 255 235 162 255 238 206  11  13   0   0   0
  0   0   0  16   9   9 150 251  45  21 184 159 154 255 233  40   0   0   0   0
 10   0   0   0   0   0 145 146   3  10   0  11 124 253 255 107   0   0   0   0
  0   0   3   0   4  15 236 216   0   0  38 109 247 240 169   0  11   0   0   0
  1   0   2   0   0   0 253 253  23  62 224 241 255 164   0   5   0   0   0   0
  6   0   0   4   0   3 252 250 228 255 255 234 112  28   0   2  17   0   0   0
  0   2   1   4   0  21 255 253 251 255 172  31   8   0   1   0   0   0   0   0
  0   0   4   0 163 225 251 255 229 120   0   0   0   0   0  11   0   0   0   0
  0   0  21 162 255 255 254 255 126   6   0  10  14   6   0   0   9   0   0   0
  3  79 242 255 141  66 255 245 189   7   8   0   0   5   0   0   0   0   0   0
 26 221 237  98   0  67 251 255 144   0   8   0   0   7   0   0  11   0   0   0
125 255 141   0  87 244 255 208   3   0   0  13   0   1   0   1   0   0   0   0
145 248 228 116 235 255 141  34   0  11   0   1   0   0   0   1   3   0   0   0
 85 237 253 246 255 210  21   1   0   1   0   0   6   2   4   0   0   0   0   0
  6  23 112 157 114  32   0   0   0   0   2   0   8   0   7   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

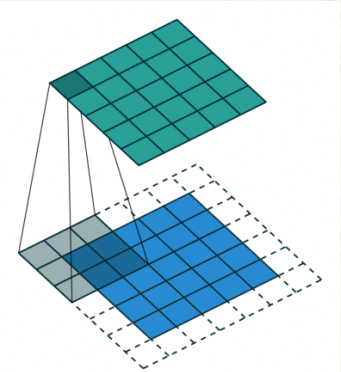
https://github.com/vdumoulin/conv_arithmetic

Convolution

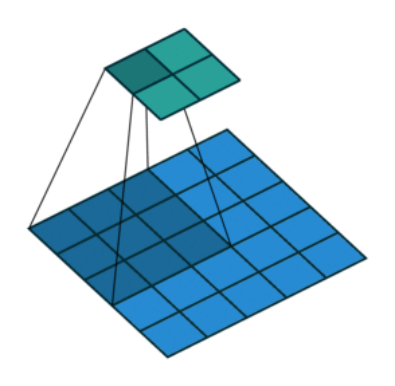
- Summary (1)



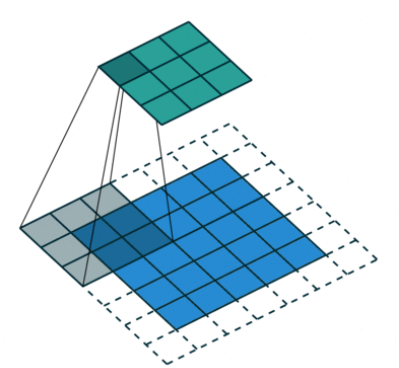
Padding=0, Strides=1



Padding=1 , Strides=1



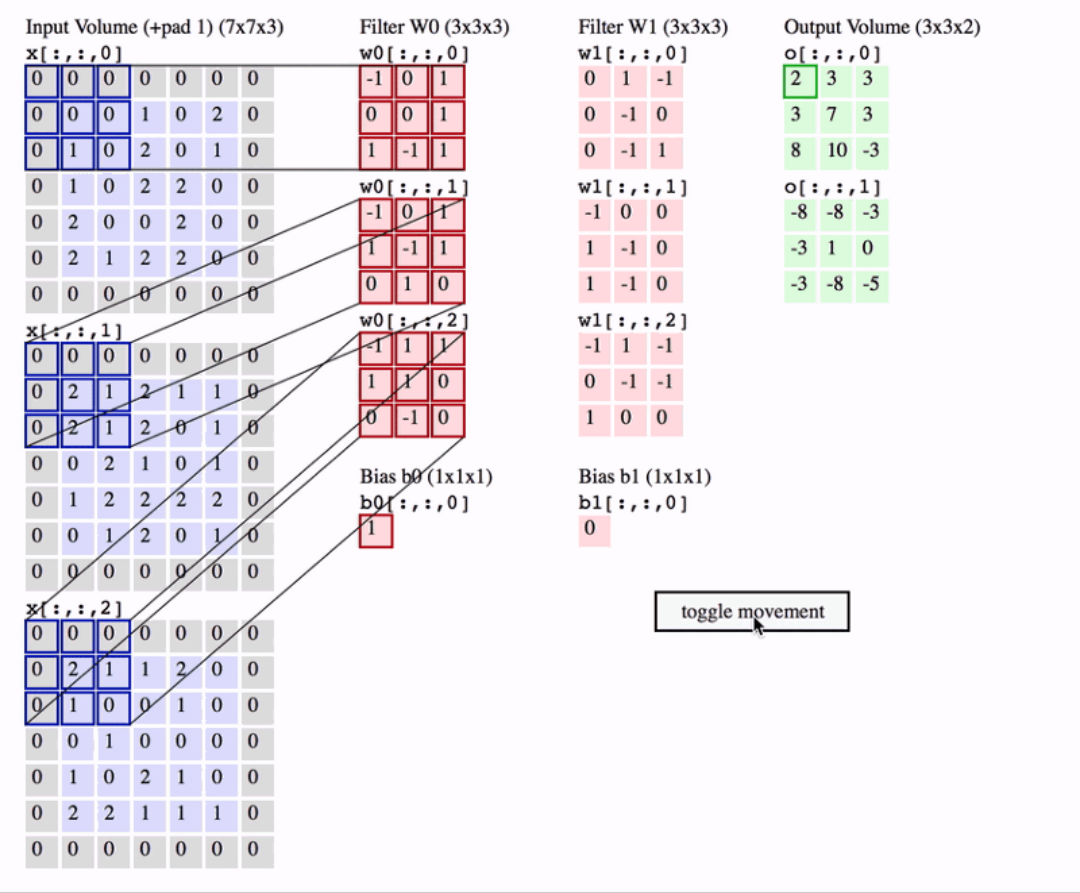
Padding=0, Strides=2



Padding=1, Strides=2

Convolution

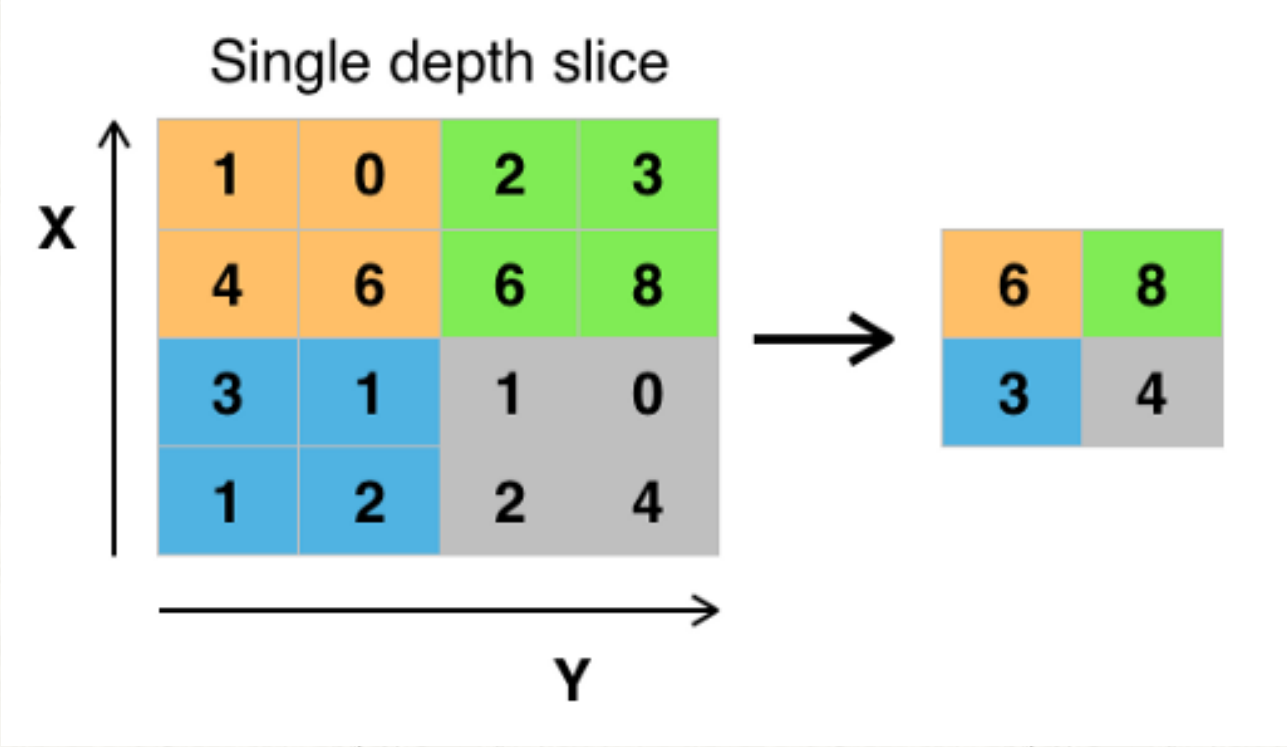
- Summary (2)



https://medium.com/@phidaouss/convolutional-neural-networks-cnn-or-convnets-d7c688b0a207

Pooling

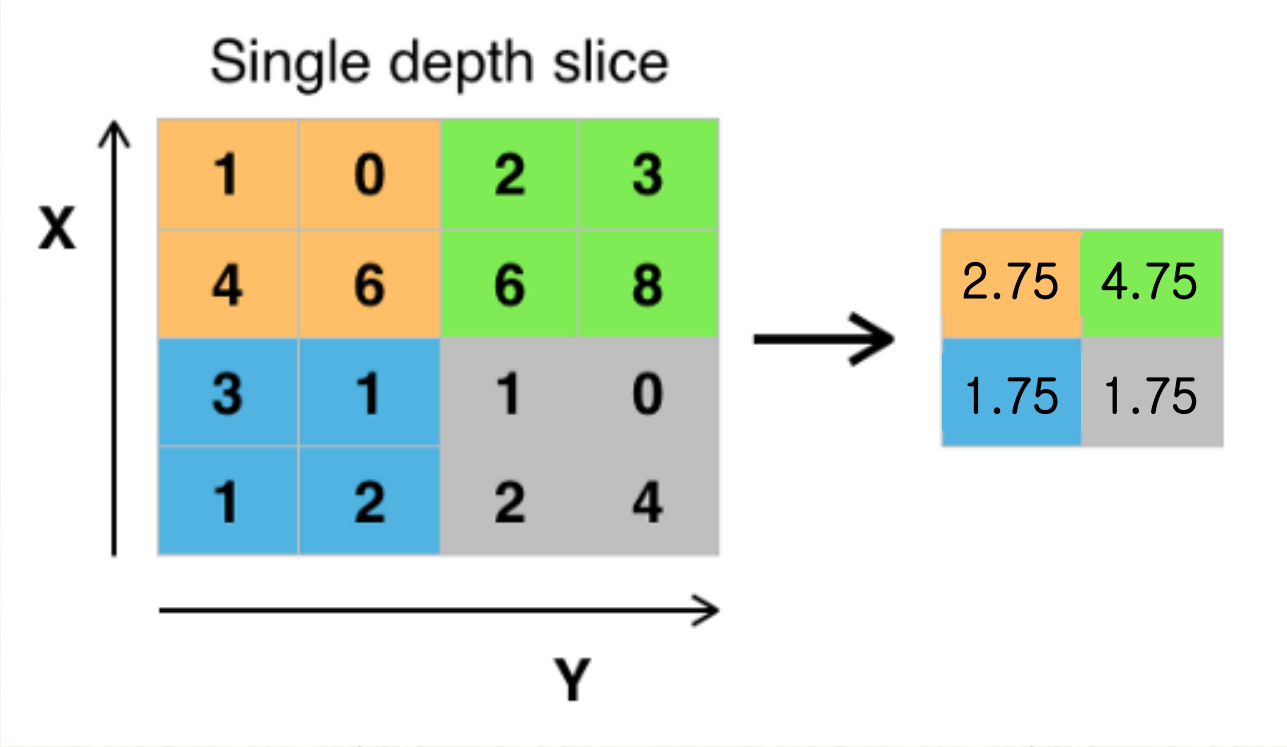
- Summary (1)
  - MAX pooling
    - Sride=2, Kernel_size=2



https://medium.com/@phidaouss/convolutional-neural-networks-cnn-or-convnets-d7c688b0a207

Pooling

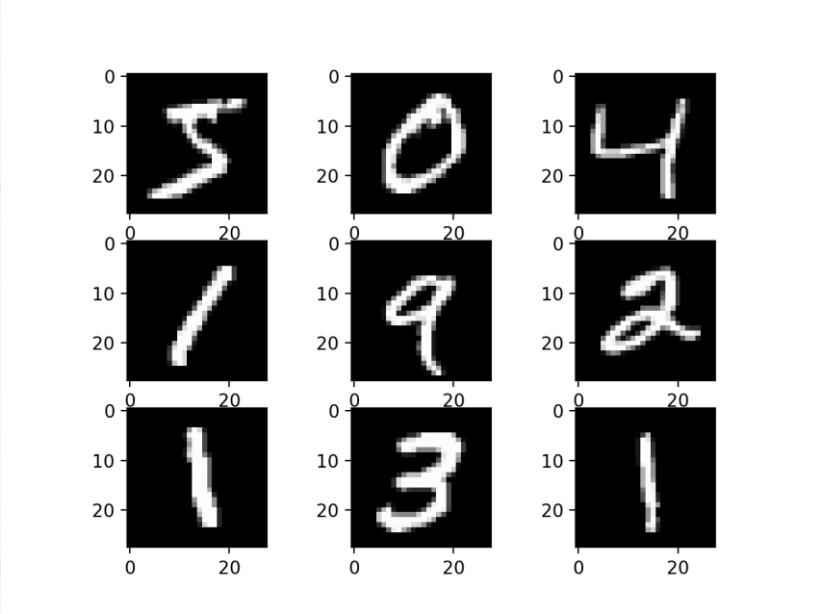
- Summary (2)
  - Average pooling
    - Sride=2, Kernel_size=2



https://medium.com/@phidaouss/convolutional-neural-networks-cnn-or-convnets-d7c688b0a207

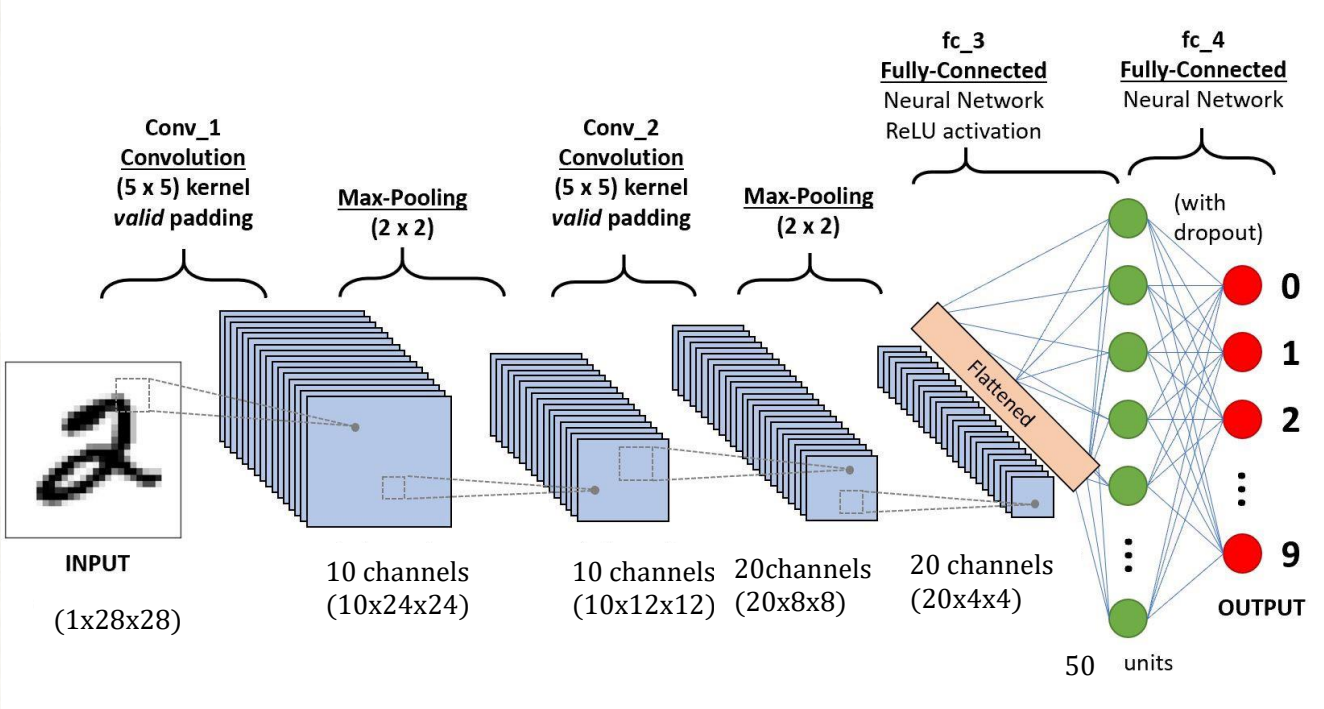
CNN for MNIST

- MNIST Dataset
  - MNIST is handwritten digit images
  - It contains training set of 60,000 examples and a test set of 10,000 examples
  - Each example is a 28x28 grayscale image, associated with a label from 10 classes.



CNN for MNIST

- CNN Architecture



CNN for MNIST

- Preparation

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import transforms, datasets

USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
```

  - `USE_CUDA = torch.cuda.is_available()`
    `DEVICE = torch.device("cuda" if USE_CUDA else "cpu")`
    - Select device to use (GPU or CPU)
    - If GPU can be used, use GPU
    - else, use CPU

CNN for MNIST

- Setup data

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.1,))
])


trainset = datasets.MNIST(
    root='./.data/',
    train=True,
    download=False,
    transform=transform
)
testset = datasets.MNIST(
    root='./.data/',
    train=False,
    download=False,
    transform=transform
)
```

- Transforms : Normalize data

CNN for MNIST

- Setup for mini-batch

```
BATCH_SIZE = 64

train_loader = torch.utils.data.DataLoader(
    dataset=trainset,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
test_loader = torch.utils.data.DataLoader(
    dataset=testset,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
```

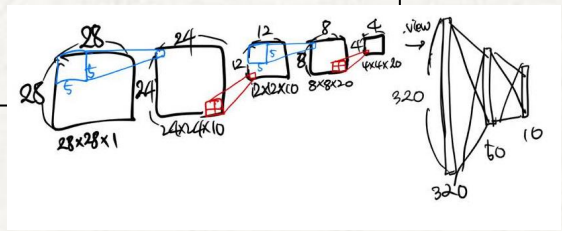
  - Divide by each batch of data
  - Data Shuffling

CNN for MNIST

- Setup Network

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```



- nn.Linear(x, y) = x_dim -> y_dim (node)
- nn.Conv2d(x, y) = x_dim -> y_dim (channel)

CNN for MNIST

- Model and Optimizer

```
model = Net().to(DEVICE)
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

  - Net.to(DEVICE)
    - .to(): function sends the parameters of the model to the specified location.
    - DEVICE = GPU or CPU (defined it before)
  - Stochastic Gradient Descent (SGD)
    - Parameters to learn
    - Learning rate – Hyper Parameter

CNN for MNIST

- Setup for training

```
def train(model, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(DEVICE), target.to(DEVICE)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 200 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss:
{:.6f}'.format(
                epoch, batch_idx * len(data),
len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

CNN for MNIST

- Setup for evaluation

```
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)

            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()

            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

CNN for MNIST

- Run!

```
EPOCHS= 30

for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer, epoch)

    if epoch % 10 == 0:
        test_loss, test_accuracy = evaluate(model, test_loader)

        print('[{}] Test Loss: {:.4f}, Accuracy: {:.2f}%'.format(
            epoch, test_loss, test_accuracy))
```

```
[10] Test Loss: 0.4043, Accuracy: 96.98%
[20] Test Loss: 0.3592, Accuracy: 97.22%
[30] Test Loss: 0.3326, Accuracy: 98.47%
```

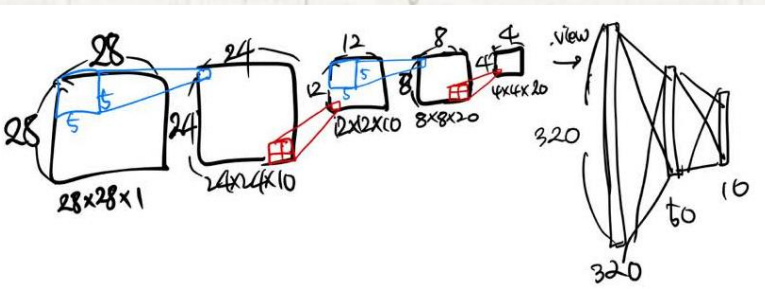
CNN for MNIST

- Convolution Layer

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)
```

  - Conv2d
    - Filter: 5x5x10, 5x5x20
    - Default: Strides: 1, Padding: 0

$$out = \frac{in + 2p - f}{s} + 1$$



Add Convolution and Pooling Layer

- CNN의 channel, kernel size, stride, padding 을 변경해서 자신만의 CNN 구조를 만들어보자!

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5, stride=1, padding=0)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5, stride=1, padding=0)
        self.fc1 = nn.Linear(???, ???)
        self.fc2 = nn.Linear(???, 10)
```

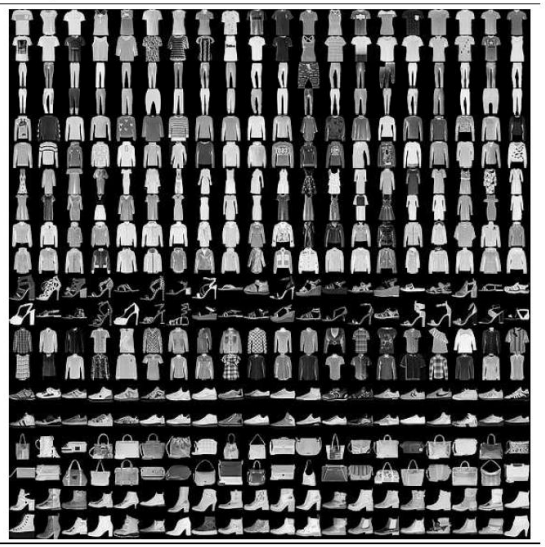
Add Dropout and Batch Normalization

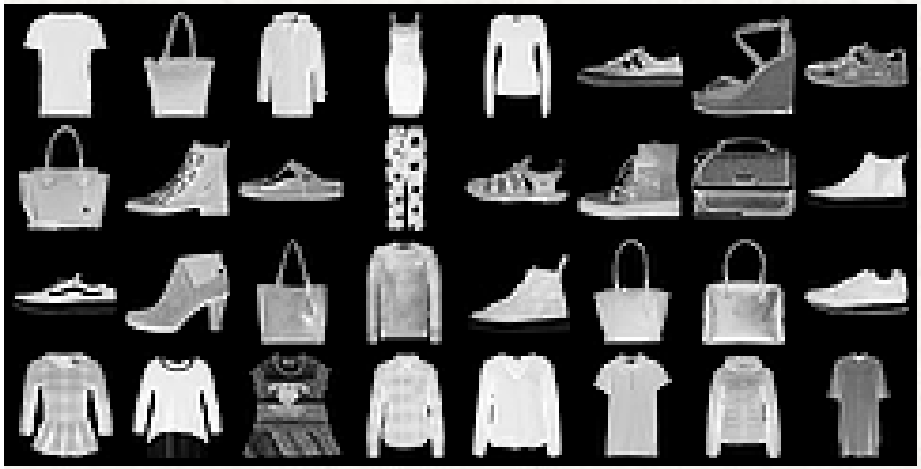
- Convolution Dropout, Batch Normalization 을 추가하여 성능을 비교해보자!

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.bc1 = nn.BatchNorm2d(10)
        self.bc2 = nn.BatchNorm2d(20)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)
        self.conv_drop = nn.Dropout2d(p=0.2)
    def forward(self, x):
        …
        return x
```

Fahsion MNIST

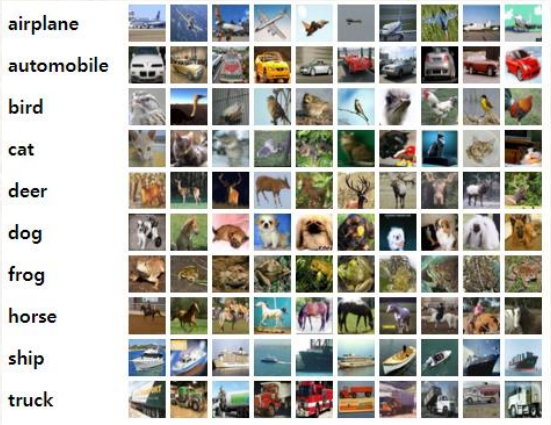
- Fashion MNIST Dataset
  - Fashion-MNIST is a dataset of Zalando's article images
  - It contains training set of 60,000 examples and a test set of 10,000 examples
  - Each example is a 28x28 grayscale image, associated with a label from 10 classes.

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	



Cifar-10

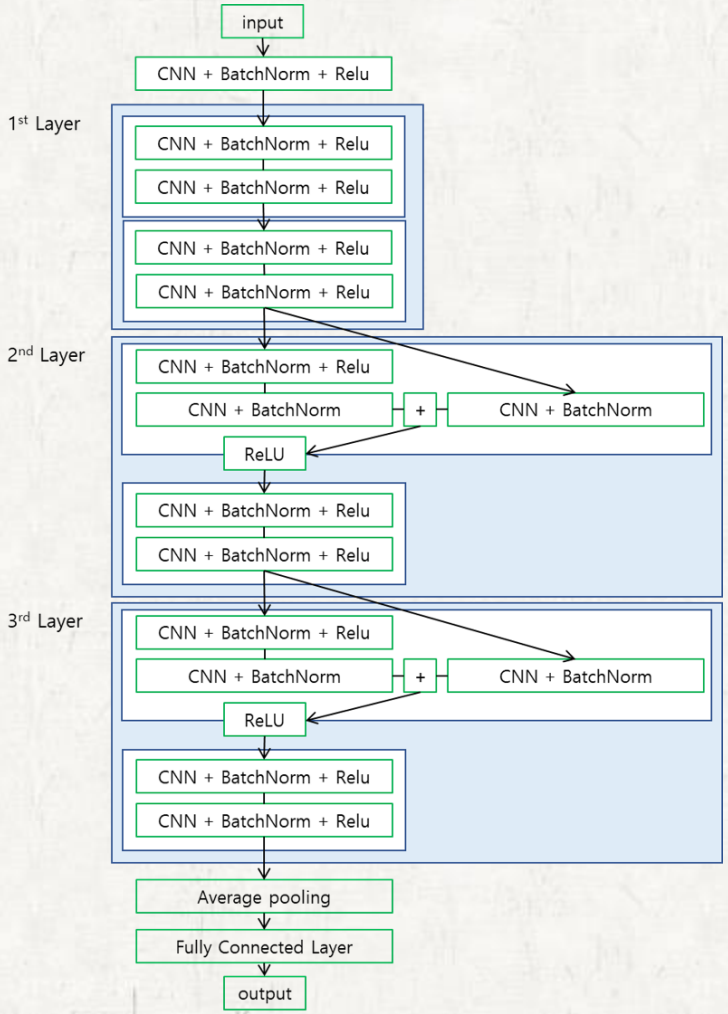
- Cifar-10 Dataset
  - Subsets of the 80 million tiny images dataset that is commonly used for training various image processing systems and machine learning
  - It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.



<Sample images from Cifar-10 dataset>

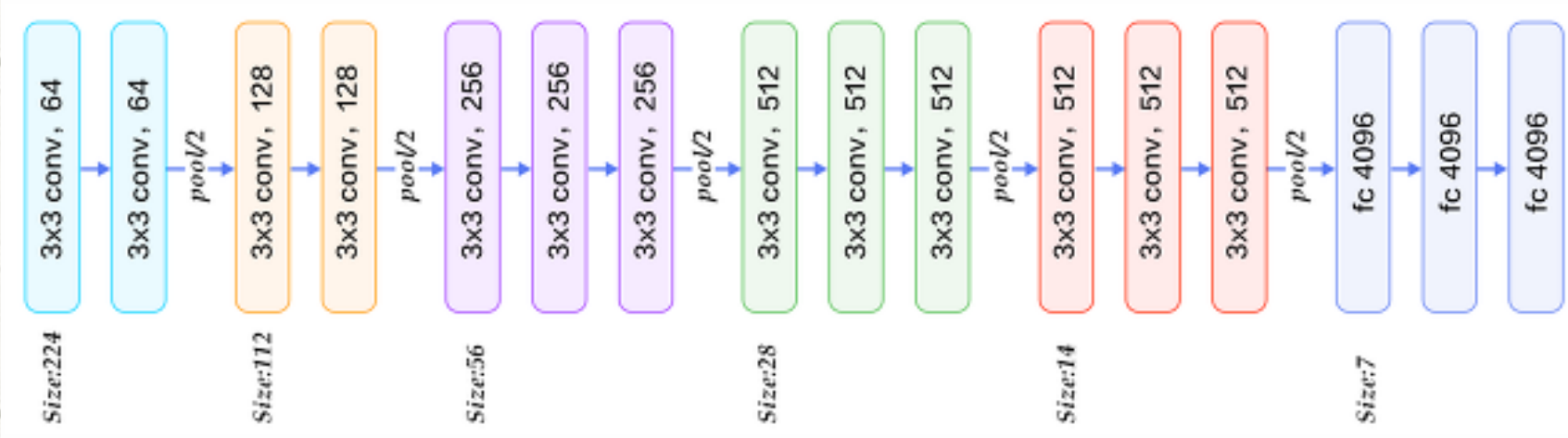
Try to make ResNet

- Res Network



Try to make VGG Net

- VGG Network



Pytorch Net

- nn.Sequential()

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 10, kernel_size=5),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.ReLU(),
            nn.Dropout2d(0.5),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(10, 20, kernel_size=5),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.ReLU(),
            nn.Dropout2d(0.5),
        )
        self.fc = nn.Sequential(
            nn.Linear(320, 50),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(50, 10),
        )

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(-1, 320)
        x = self.fc(x)
        return F.log_softmax(x, dim=1)
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = make_Sequential(
                        1, 10, 5, 2, 2, 0.5)
        self.conv2 = make_Sequential(
                        10, 20, 5, 2, 2, 0.5)
        self.fc = nn.Sequential(
            nn.Linear(320, 50),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(50, 10),
        )

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(-1, 320)
        x = self.fc(x)
        return F.log_softmax(x, dim=1)


def make_Sequential(in_ch, out_ch,
                    conv_k, pool_k, pool_s, dropout_p):
    return nn.Sequential(
        nn.Conv2d(in_ch, out_ch,
                            kernel_size=conv_k),
        nn.MaxPool2d(kernel_size=pool_k,
                                stride=pool_s),
        nn.ReLU(),
        nn.Dropout2d(dropout_p),
    )
```