Samsung TA session(NPEX)

[LAB1]: MNIST classification with various neural network

[LAB2]: Pyramid Stereo matching network

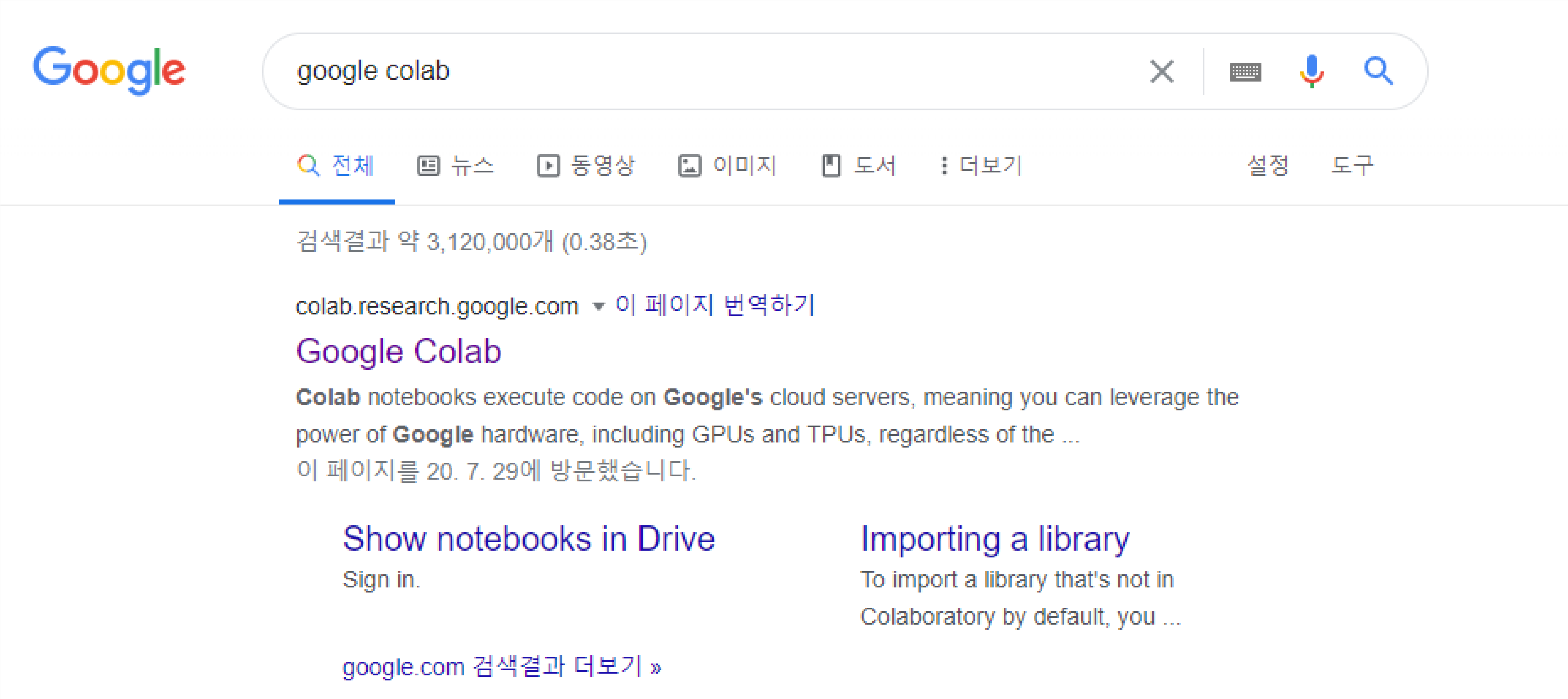
Ph.D student

Taewoo Kim

Visual intelligence lab

Lab1: Prerequisite - Colab.

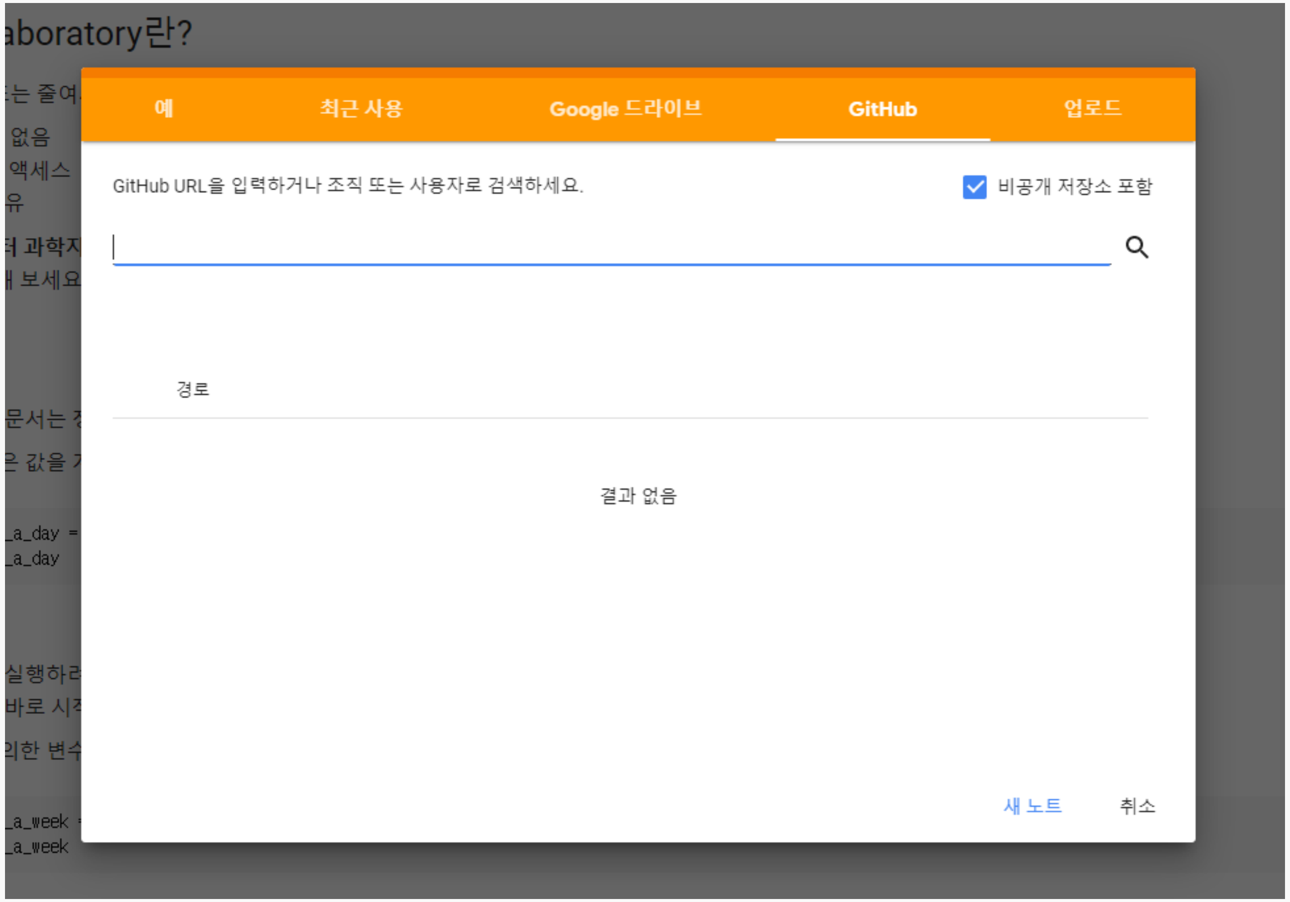
Enter the colab



Lab1: Prerequisite - Colab.

1. Enter the colab

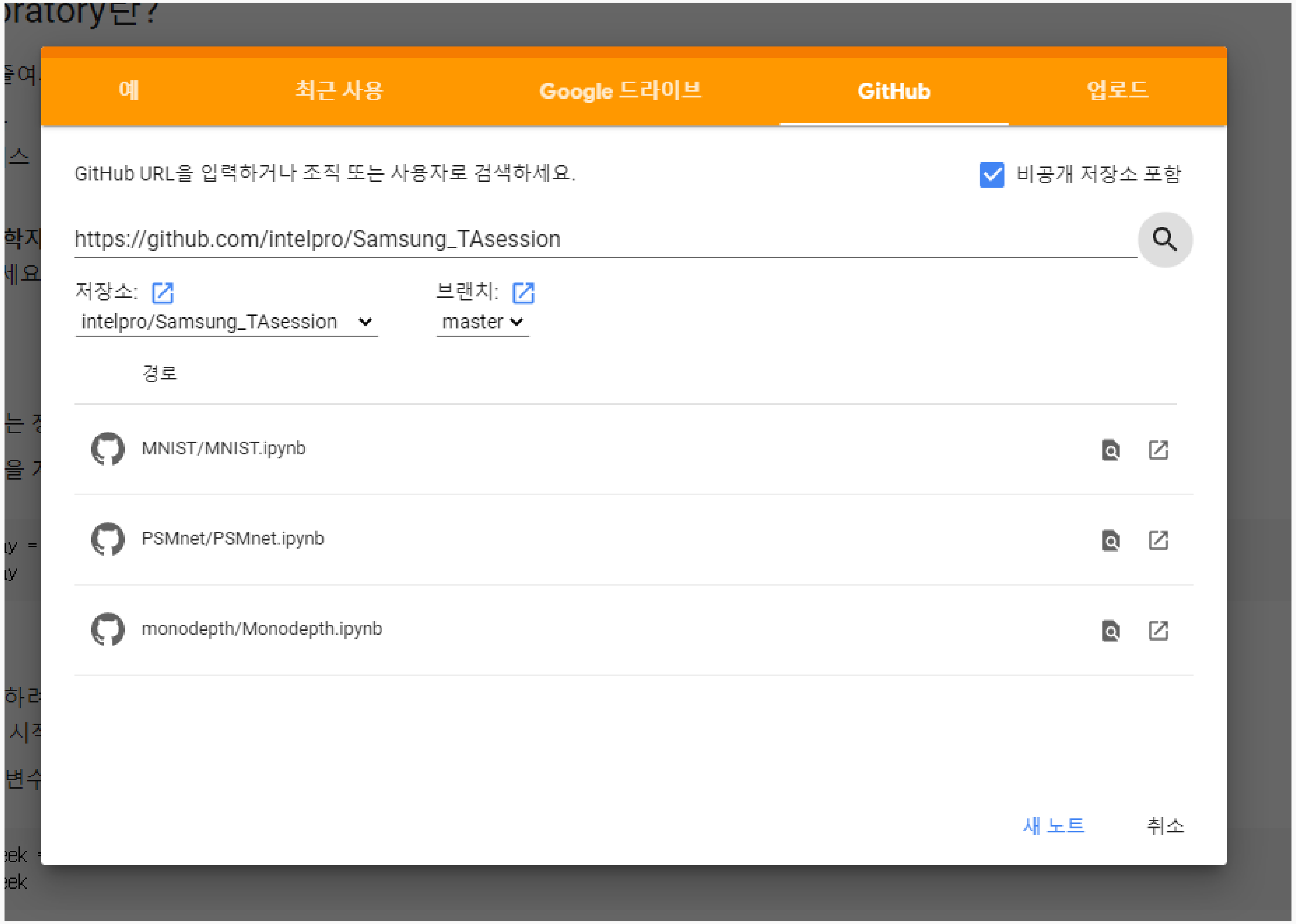
2. Enter the address below



https://github.com/intelpro/Samsung_TAsession

Lab1: Prerequisite - Colab.

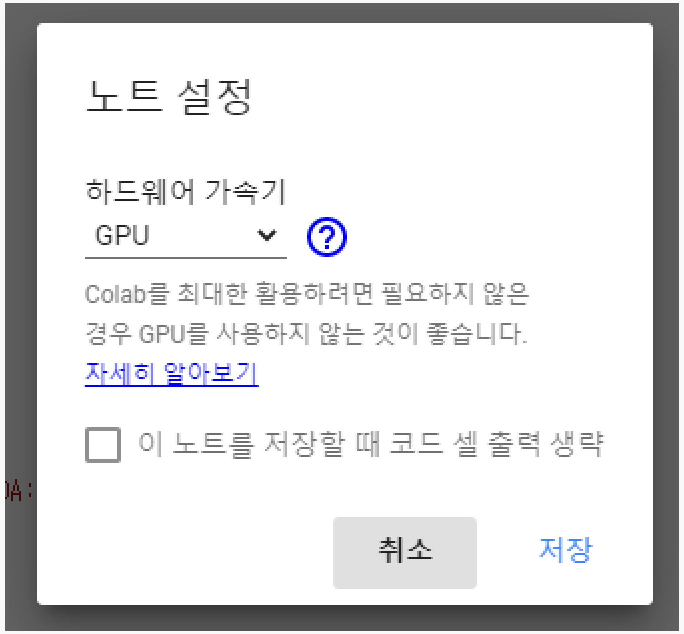
Click MNIST.ipynb



Lab1: Prerequisite - Colab.

(한글)런타임 -> 런타임 유형 변경 -> 하드웨어 가속기(GPU)

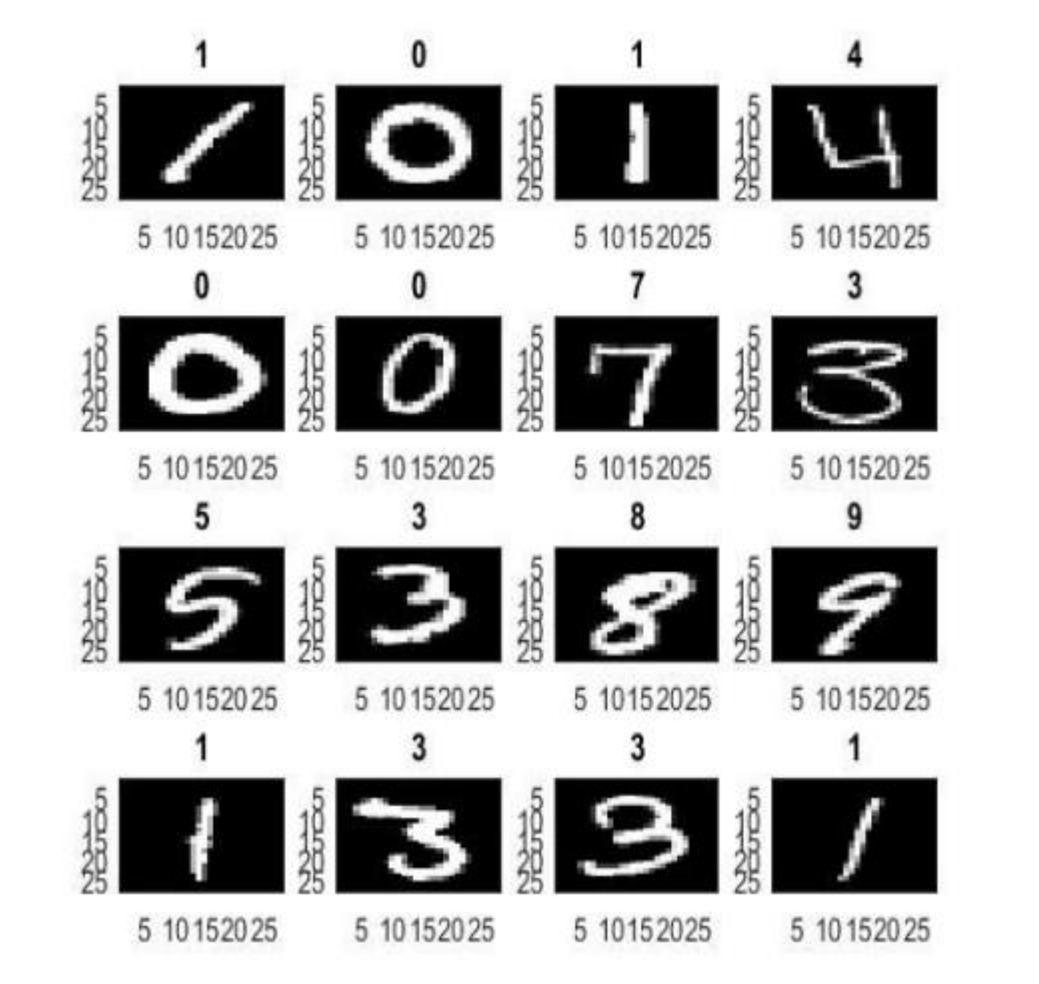
(Engilsh) runtime -> Runtime type change -> Hardware accelerator(GPU)



Lab1: Introduction

MNIST

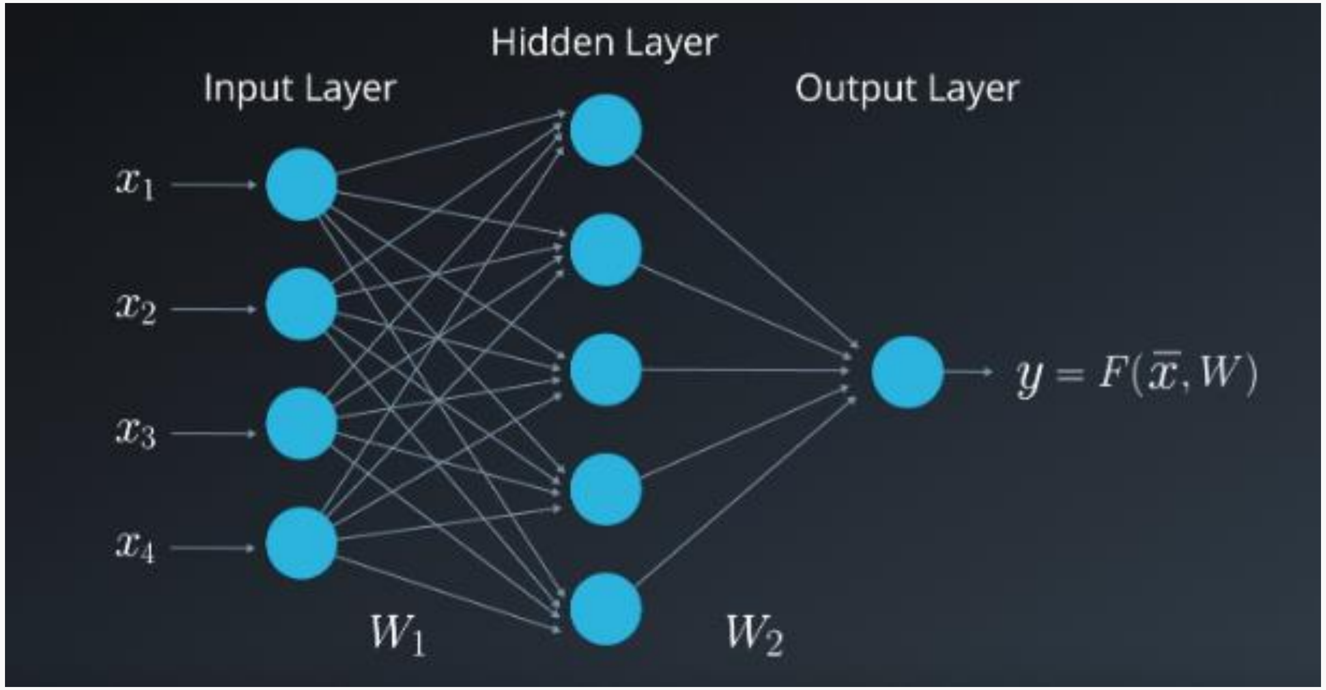
- Large hand written digit classification database
- Format
  - Input: 28 x 28 gray-scale image
  - Output: 10 labels(0-9)
  - Centered on center of the mass



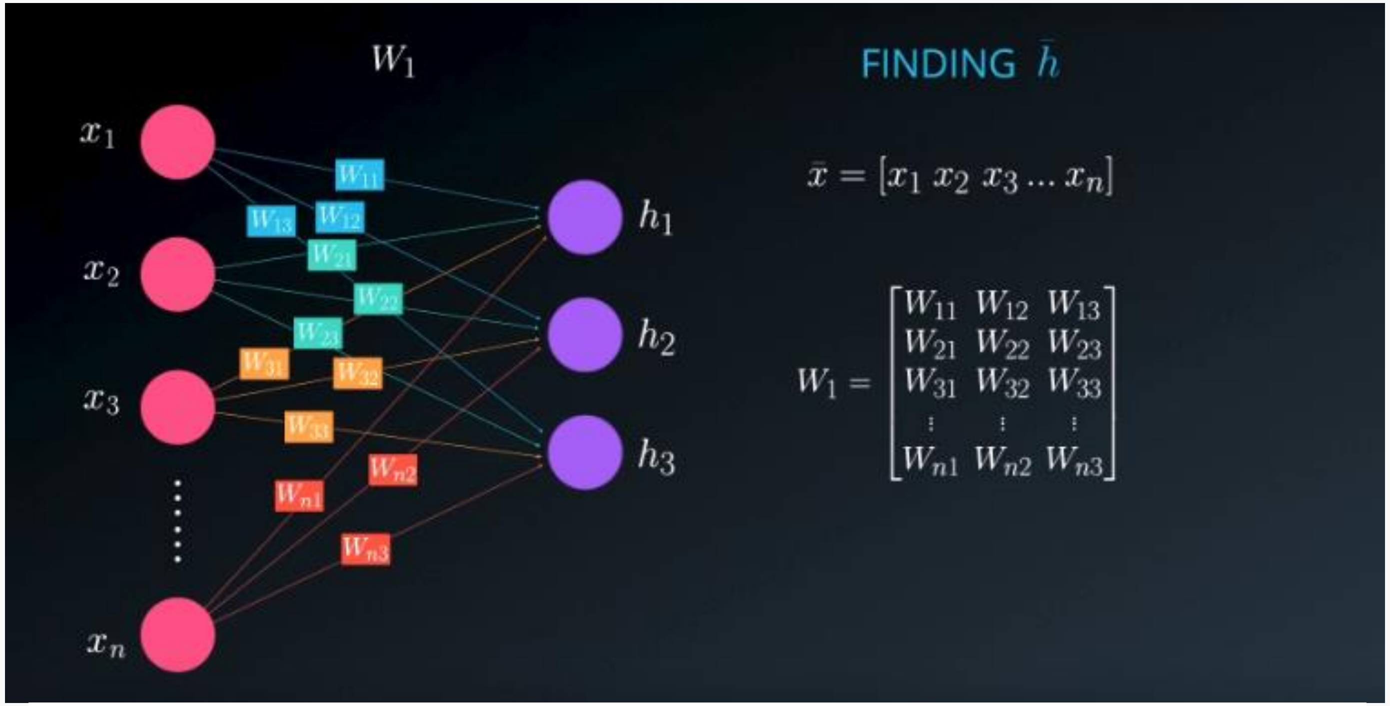
Lab1: Introduction

Multi-layer perceptron

- Layer 1: Input layer – (28x28x1) dimension
- Layer 2: Hidden layer – Multi layer perceptron
- Layer 3: Output Layer – 10 labels(0-9)



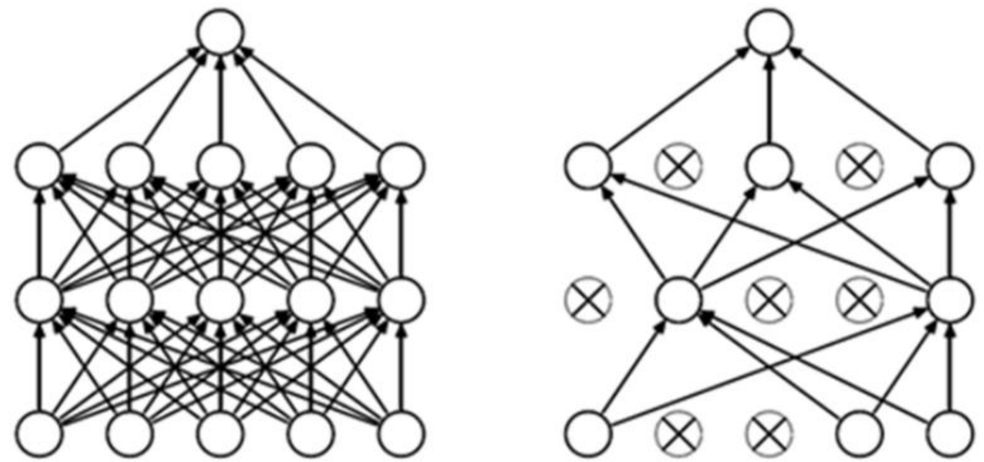
Multi-layer perceptron(hidden layer)



$$\begin{bmatrix} h'_1 & h'_2 & h'_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & \ldots\ldots & x_n \end{bmatrix} \cdot \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ \vdots & & \\ W_{n1} & W_{n2} & W_{n3} \end{bmatrix}$$

Multi-layer perceptron – dropout

- One of the biggest problem of deep learning is over fitting to training data
- The dropout is a learning method using only a part of deep network
- We can partially solve the overfitting problem through dropout



(a) Standard Neural Net (b) After applying dropout.

Lab1: Introduction

Pytorch MLP basic - nn.Linear

CLASS `torch.nn.Linear(in_features: int, out_features: int, bias: bool = True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shape:

- Input: $(N, *, H_{in})$ where $*$ means any number of additional dimensions and $H_{in} = \text{in\_features}$
- Output: $(N, *, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} = \text{out\_features}$.

Variables

- **~Linear.weight** – the learnable weights of the module of shape $(\text{out\_features}, \text{in\_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in\_features}}$
- **~Linear.bias** – the learnable bias of the module of shape (out_features). If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in\_features}}$

https://pytorch.org/docs/master/generated/torch.nn.Linear.html

Lab1: Introduction

Pytorch MLP basic - nn.Dropout

CLASS `torch.nn.Dropout(p: float = 0.5, inplace: bool = False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Shape:

- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

https://pytorch.org/docs/master/generated/torch.nn.Dropout.html

Lab1: Introduction

MNIST tutorial - Dataset preparation

- 데이터 셋을 로딩하여 준비하는 과정
- Training 및 Validation 데이터를 로딩하는 과정에서 Normalization, data augmentation 등을 정의
- Shuffle 유무 / batch size 등 data loading에 필요한 내용들도 정의

```
batch_size = 32

kwargs = {'num_workers': 1, 'pin_memory': True} if cuda else {}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True, **kwargs)

validation_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=False, **kwargs)
```

Lab1: Introduction

MNIST tutorial - Network definition class

- MNIST classification을 수행할 DNN 구조를 정의하는 부분
- Pytorch의 경우 Network정의 부분은 torch.nn.Module의 class를 상속받아 사용함.
- DNN의 구성요소를 정의하는 생성자 부분과 forward 함수로 구분이 되어있음.
- Forward 함수의 경우 input data를 어떻게 처리하는 지를 정의함.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 50)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(50, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        return F.log_softmax(self.fc3(x))
```

Network definition

Forward function

Lab1: Introduction

MNIST tutorial - Training

- Training loop는 보통 다음과 같은 부분으로 구성이 됨.
  1) optimizer로 부터 gradient를 계산하는 부분
  2) model로 부터 output을 얻는 부분
  3) 얻어진 outpu으로 부터 loss를 계산하는 부분
  4) Training관련된 정보를 logging하는 부분

```
def train(epoch, log_interval=100):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)                          → Get output
        loss = F.nll_loss(output, target)             → Calculate loss
        loss.backward()                               → Backpropagation using optimizer
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),    → Print training log
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Lab1: Introduction

MNIST tutorial - Validation

- Validation 는 Training을 하면서 training이 잘 되고 있는지, 성능이 어떻게 변화하고 있는지 등을 관찰하기 위함

- 이 부분은 training loop와 다르게 optimizer가 동작을 하면 안되고, gradient를 계산하지 않아야 함.

- Validation loop는 보통 1) accuracy 계산 2) loss 계산 3) validation log 출력 등으로 구성 됨.

```python
def validate(loss_vector, accuracy_vector):
    model.eval()
    val_loss, correct = 0, 0
    for data, target in validation_loader:
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        val_loss += F.nll_loss(output, target).data[0]
        pred = output.data.max(1)[1] # get the index of the max log-probability
        correct += pred.eq(target.data).cpu().sum()

    val_loss /= len(validation_loader)
    loss_vector.append(val_loss)

    accuracy = 100. * correct / len(validation_loader.dataset)
    accuracy_vector.append(accuracy)

    print('\nValidation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        val_loss, correct, len(validation_loader.dataset), accuracy))
```

Calculate validation loss

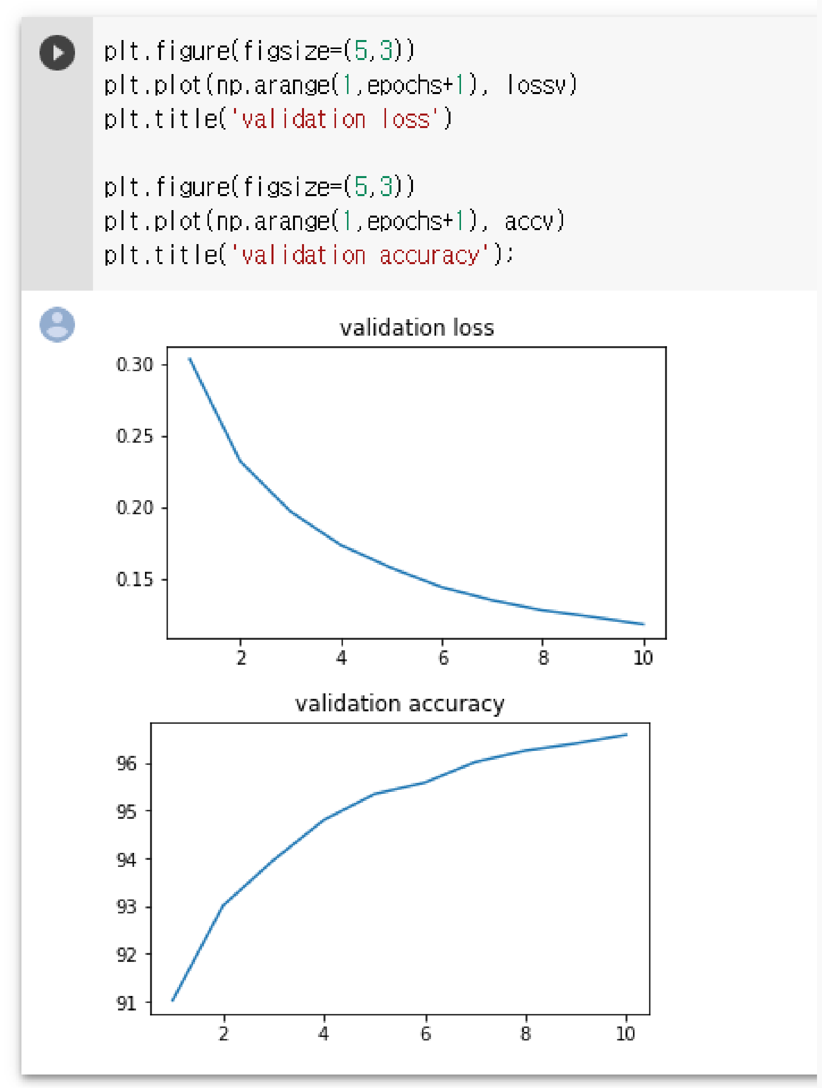
Calculate accuracy of validation set

Print log of accuracy of validation set

Lab1-0 – MNIST classification

MNIST tutorial - Running MNIST classification

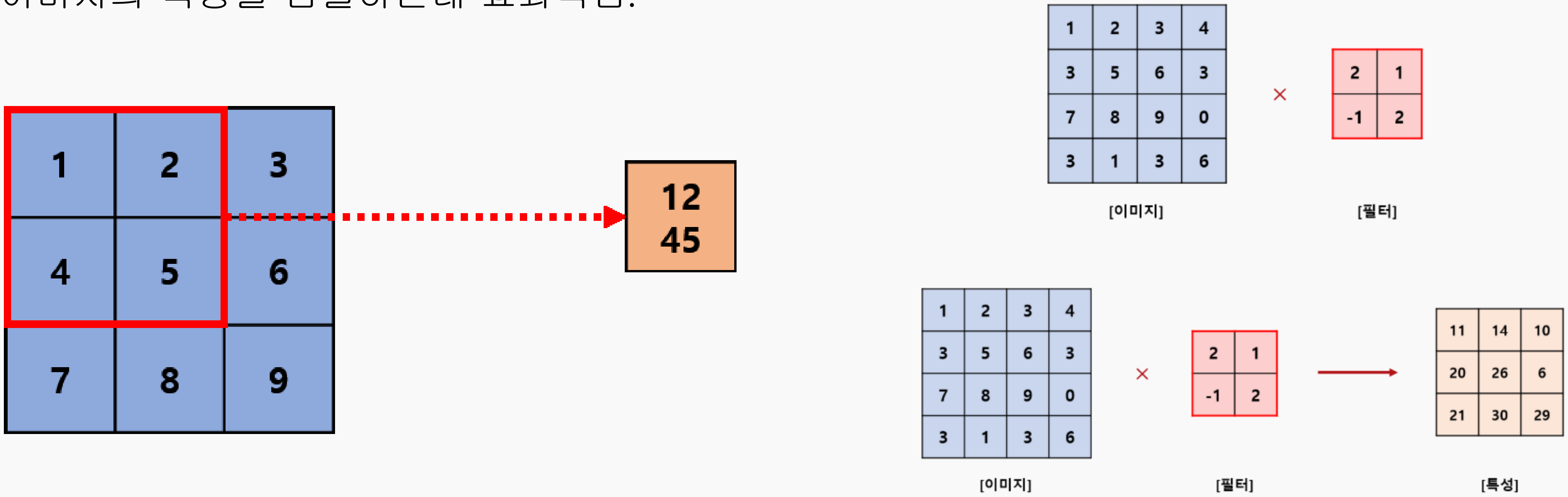
1. Dataloder
2. Print data
3. Network definition
4. Training and validation
5. Plot validation loss and validation accuracy



Final result

MNIST with Simple CNN

- Convolution
  - 필터 혹은 커널을 통해 가중치와 입력 값을 곱하여 얻는 연산 과정
  - 이미지의 특징을 검출하는데 효과적임.

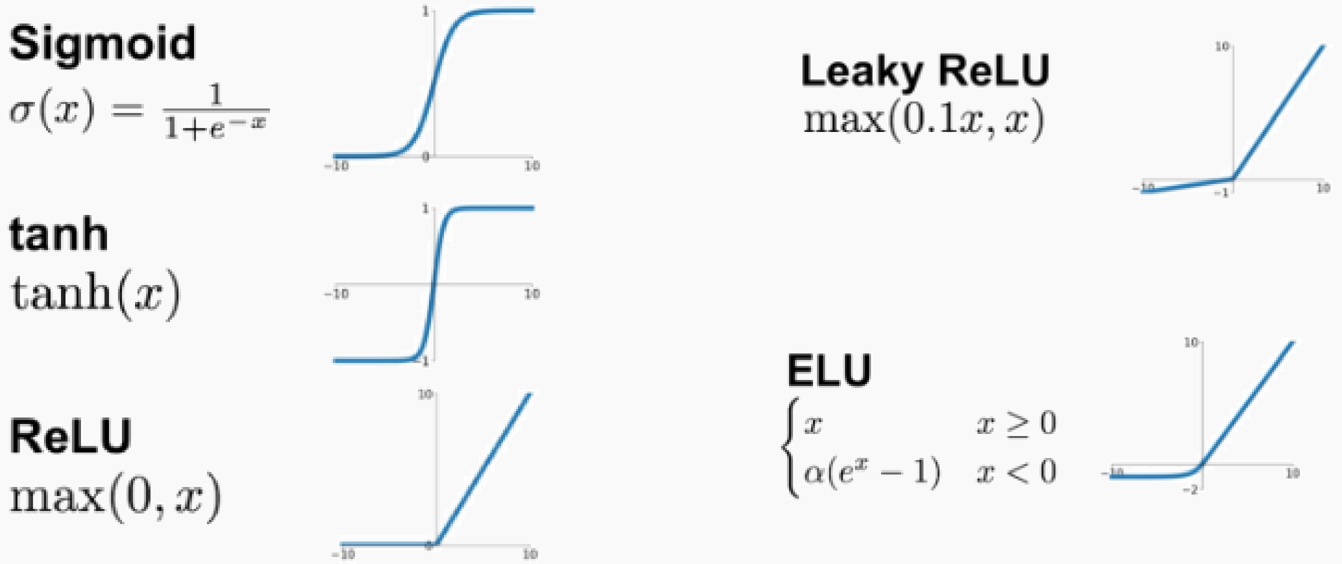


Convolution 연산 과정

Lab 1-1: Simple CNN

MNIST with Simple CNN

- 활성화 함수(Activation function)
  - Convolution filter를 통해서 특징 맵이 추출 되면, 이 특징 맵에 활성화 함수를 적용하여야 함
  - 활성화 함수를 사용하게 되면, 입력 값에 대한 출력이 linear하게 나오지 않게 됨으로, 네트워크의 입력 및 출력 관계의 비선형성을 조금 더 잘 모델링 할 수 있다.

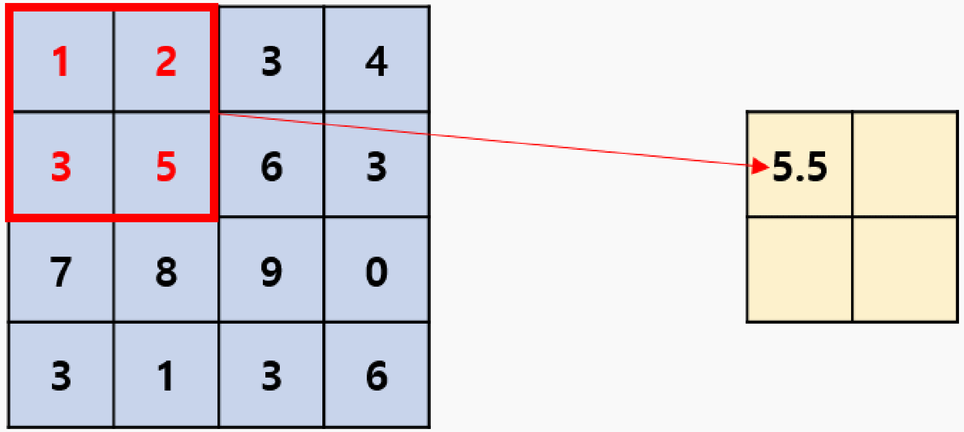


활성화 함수 예시

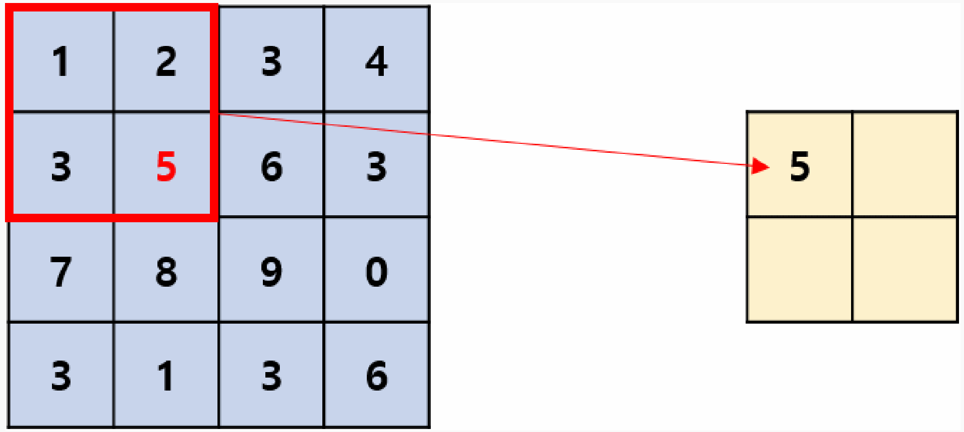
Lab 1-1: Simple CNN

MNIST with Simple CNN

- Pooling
  - 보통 DNN은 입력을 encoding하는 과정이 필요하며, spatial dimension을 줄여줌으로 주요 특징들을 압축하여 encoding할 수 있게 도와줌.
  - Max Pooling 과 Average Pooling으로 분류를 할 수 있으며, Max pooling은 kernel의 최대값을 뽑아 pooling을 수행함.
  - Average Pooling은 kernel의 평균값을 output으로 가지도록 함.



Average Pooling



Max Pooling

Lab 1-1: Simple CNN

MNIST with Simple CNN

- Convolution 연산과 output tensor size
  - O: Size of output image
  - I: Size of input image
  - K: convolution kernel size
  - P: padding size
  - S: Stride of convolution operation
  - $\boldsymbol{O} = 1 + \frac{\boldsymbol{I} - \boldsymbol{K} + 2\boldsymbol{P}}{\boldsymbol{S}}$

Lab 1-1: Simple CNN

MNIST with Simple CNN(Conv2D)

CONV2D

CLASS `torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')` [SOURCE]

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

Lab 1-1: Simple CNN

MNIST with Simple CNN(Conv2D)

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Examples

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
```

Lab 1-1: Simple CNN

MNIST with Simple CNN(MaxPool2D)

MAXPOOL2D

CLASS `torch.nn.MaxPool2d(kernel_size: Union[T, Tuple[T, ...]], stride: Optional[Union[T, Tuple[T, ...]]] = None, padding: Union[T, Tuple[T, ...]] = 0, dilation: Union[T, Tuple[T, ...]] = 1, return_indices: bool = False, ceil_mode: bool = False)` [SOURCE]

Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded on both sides for `padding` number of points. `dilation` controls the spacing between the kernel points. It is harder to describe, but this link has a nice visualization of what `dilation` does.

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

Lab 1-1: Simple CNN

MNIST with Simple CNN(MaxPool2D)

Shape:

- Input: (N, C, H_{in}, W_{in})
- Output: (N, C, H_{out}, W_{out}), where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

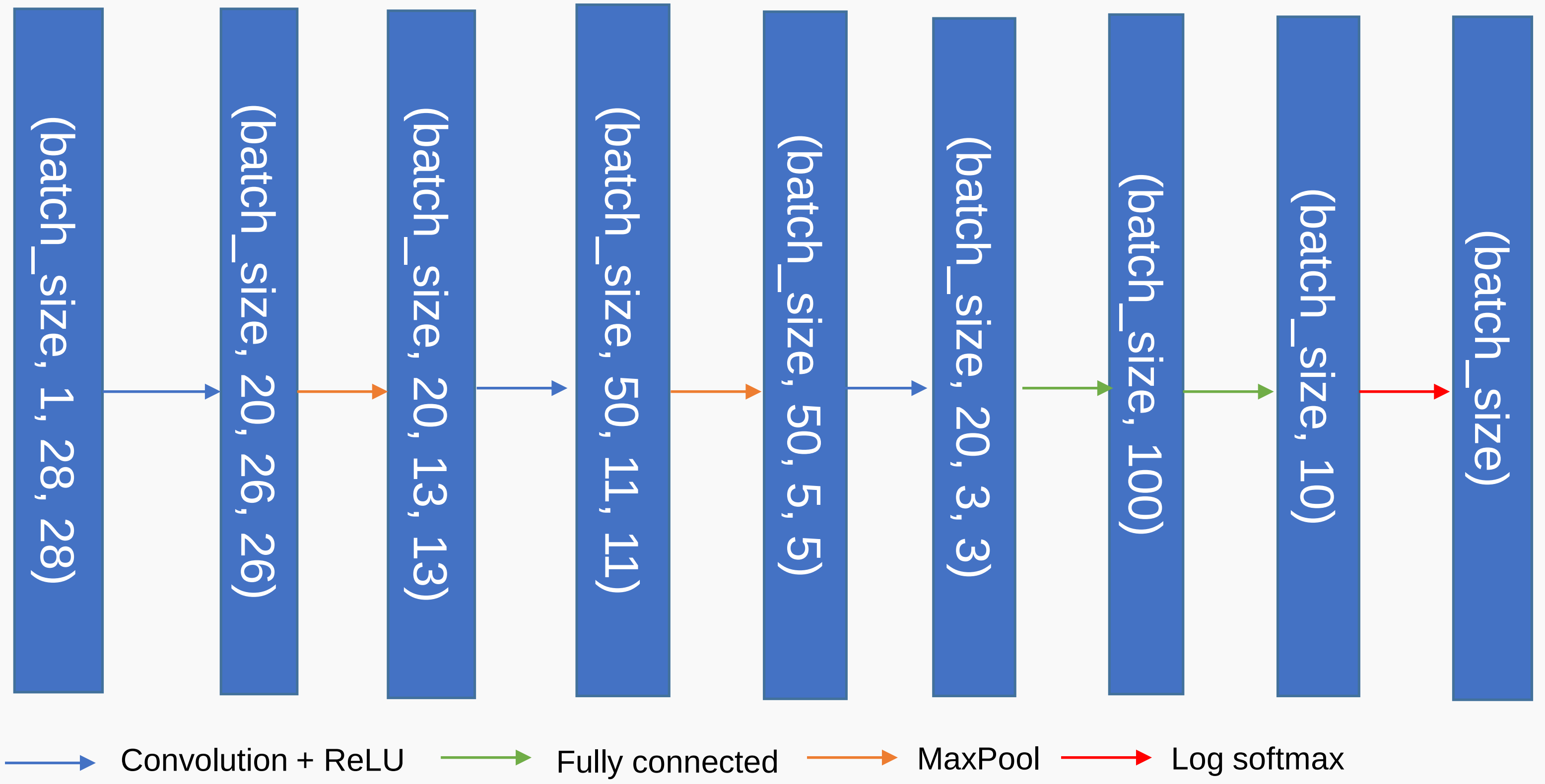
$$W_{out} = \left\lfloor \frac{W_{in} + 2 * \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Examples:

```
>>> # pool of square window of size=3, stride=2
>>> m = nn.MaxPool2d(3, stride=2)
>>> # pool of non-square window
>>> m = nn.MaxPool2d((3, 2), stride=(2, 1))
>>> input = torch.randn(20, 16, 50, 32)
>>> output = m(input)
```

Lab 1-1: Simple CNN

MNIST with Simple CNN




```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = NotImplemented
        self.maxpool1 = NotImplemented
        self.conv2 = NotImplemented
        self.maxpool2 = NotImplemented
        self.conv3 = NotImplemented
        self.fc1 = NotImplemented
        self.fc2 = NotImplemented

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.maxpool1(x)
        x = F.relu(self.conv2(x))
        x = self.maxpool2(x)
        x = F.relu(self.conv3(x))
        x = x.view(-1, 3*3*20)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

** Implementation **

Note:

Fill the blank Learning with simple CNN of Notimplemented

Caution: when you use construct CNN, do not touch stride or padding size, Just use proper kernel size to match dimension

Lab 1-2: VGG-like CNN

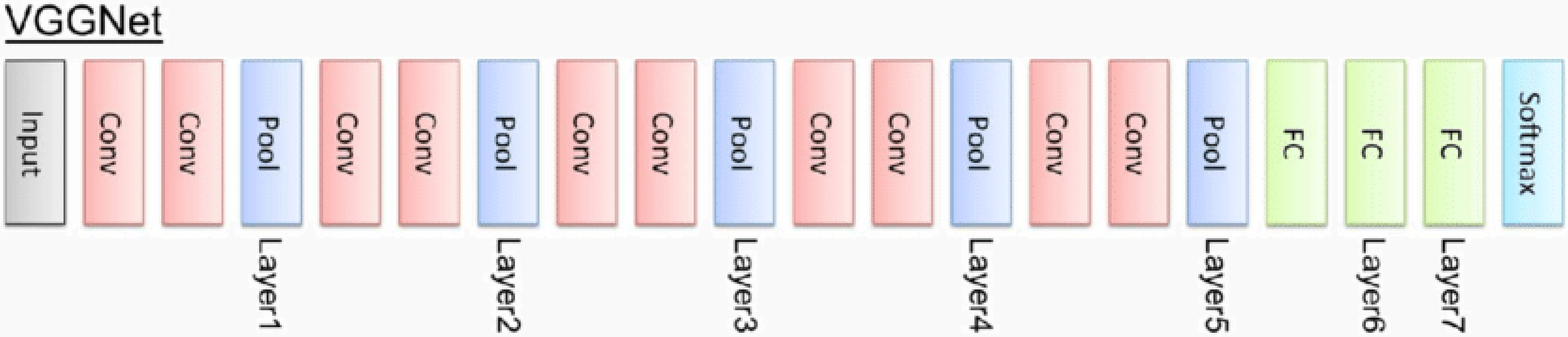
MNIST with VGG-like CNN

- VGGNet
  - VGGNet은 모든 Network 계층에서 3x3 filter를 사용하여 네트워크의 깊이를 높임.
  - 결과적으로 ILSVRC classification and localization에서 높은 정확도를 달성할 뿐만 아니라, 단순한 파이프라인으로 우수한 성능을 얻으며, 두 가지(16layer, 19layer) 최고의 성과를 낸 모델을 출시하고 다른 이미지 인식 데이터 셋에도 적용이 가능함.
  - VGGNet이전의 GoogleNet과 같은 다른 구조 보다 간단한 구조로 구성되어 있어 수정이 용이함.

Lab 1-2: VGG-like CNN

MNIST with VGG-like CNN

- VGGNet
  - Convolution layer: 3x3 filter를 사용함. (13개)
  - Pooling layer: 5개의 max pooling을 사용함. (stride는 2를 사용)
  - Dense layer: 3 layer of MLP(multi-layer perceptron)



네트워크 architecture 모식도

Lab 1-2: VGG-like CNN

MNIST with VGG-like CNN

Layer	Output dimensions	Layer	Output dimensions
0	(batch_size, 1, 28, 28)	12 – Linear1	(batch_size, 100)
1 – Conv + ReLU	(batch_size, 64, 28, 28)	13 – Dropout	(batch_size, 100)
2 – MaxPooling2D	(batch_size, 64, 15, 15)	14 – Linear2	(batch_size, 100)
3 – Conv + ReLu	(batch_size, 128, 17, 17)	15 – Dropout	(batch_size, 10)
4 – MaxPooling2D	(batch_size, 128, 8, 8)	16 – Linear3	(batch_size)
5 – Conv + Conv + ReLU	(batch_size, 128, 10, 10)		
6- maxPooling2D	(batch_size, 128, 5, 5)		
7 – Conv + Conv + ReLU	(batch_size, 512, 5, 5)		
8 – MaxPooling2D	(batch_size, 512, 2, 2)		
9 – Conv + Conv + ReLU	(batch_size, 512, 2, 2)		
10 – MaxPooling2D	(batch_size, 512, 1,)		
11 - Stretch tensor	(batch_size, 512)		

Note:
Please fill in Notimplemented of Learning with VGG like CNN. The park marked in orange is the part that needs to be implemented

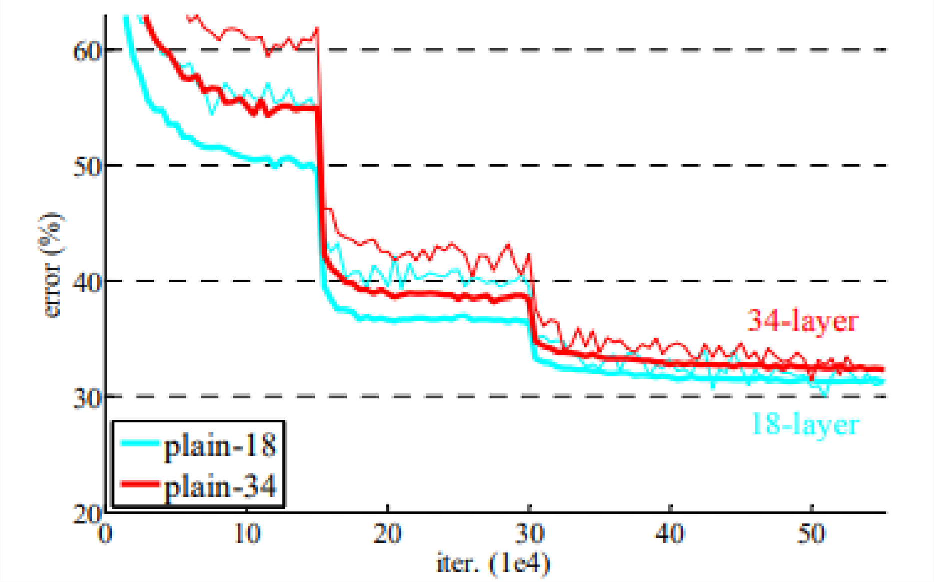
```
class SimpleVGG(nn.Module):
    def __init__(self):
        super(SimpleVGG, self).__init__()
        # Your implementation here
        self.conv1 = NotImplemented
        self.conv2 = NotImplemented
        self.conv3 = NotImplemented
        self.conv4 = NotImplemented
        self.conv5 = NotImplemented
        self.conv6 = NotImplemented
        self.conv7 = NotImplemented
        self.conv8 = NotImplemented
        self.Linear1 = NotImplemented
        self.drop = NotImplemented
        self.Linear2 = NotImplemented
        self.last_linear = NotImplemented
        # end of your implementation
        self.act = nn.ReLU()
        self.maxpool2d = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.act(self.conv1(x))
        x = self.maxpool2d(x)
        x = self.act(self.conv2(x))
        x = self.maxpool2d(x)
        x = self.act(self.conv4(self.conv3(x)))
        x = self.maxpool2d(x)
        x = self.act(self.conv6(self.conv5(x)))
        x = self.maxpool2d(x)
        x = self.act(self.conv8(self.conv7(x)))
        x = self.maxpool2d(x)
        x = x.view(-1, 512)
        # Your implementation here
        return  F.log_softmax(x, dim=1)
```

** Implementation **

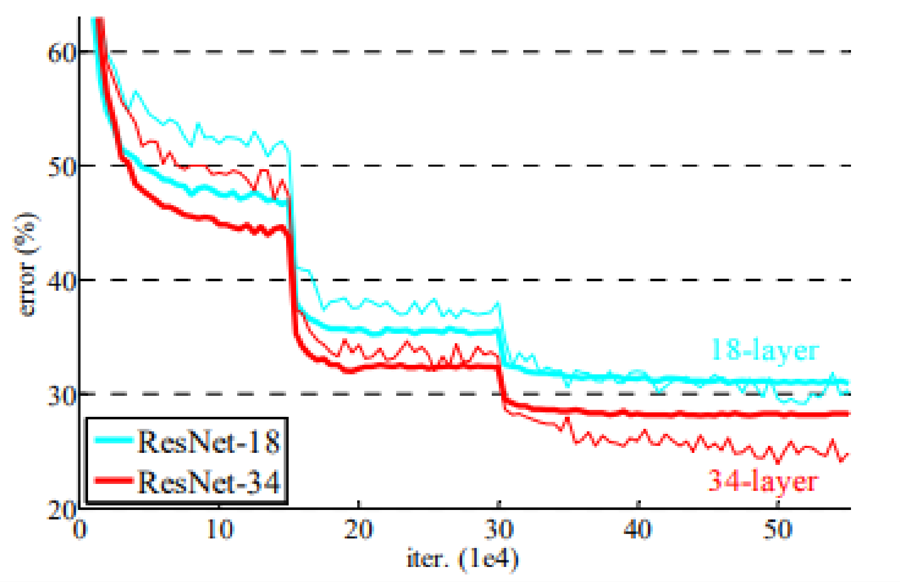
Lab 1-3: ResNet

Deep Residual Learning for image recognition(CVPR 2016)

< ImageNet top-1 training error >



일반적인 CNN

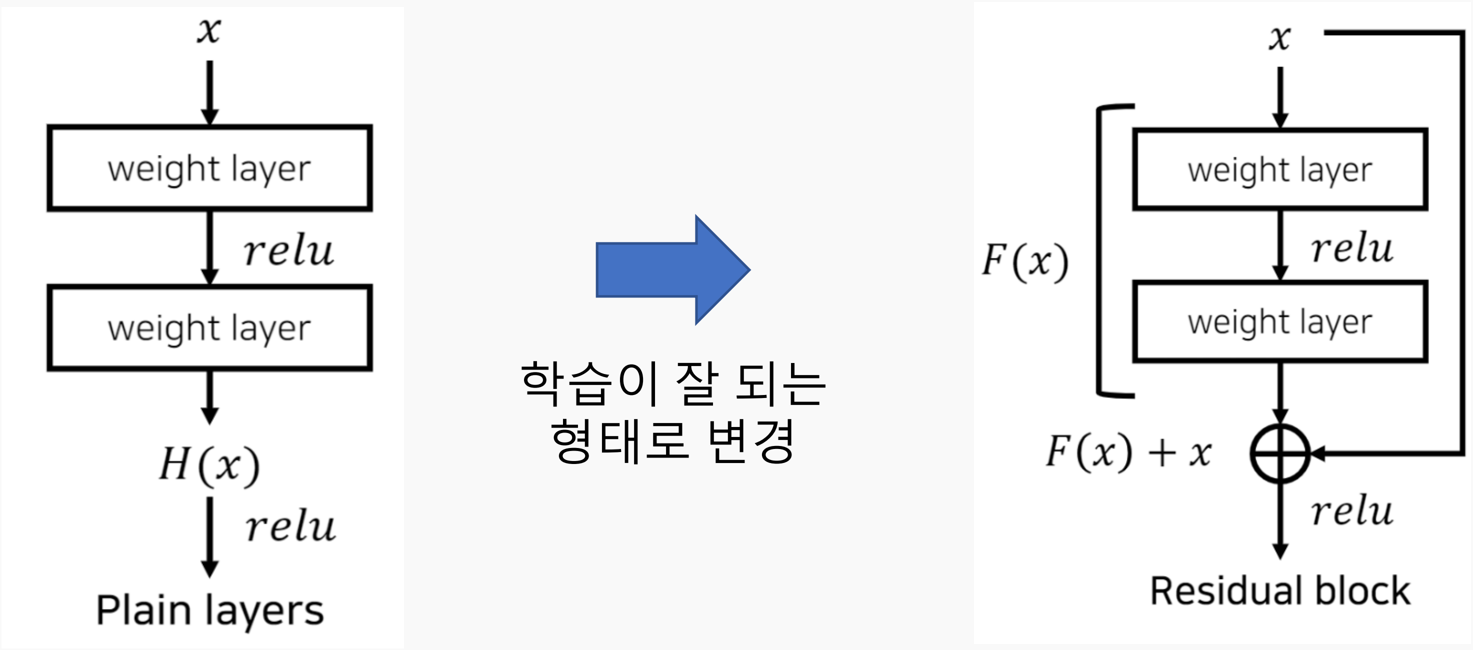


Residual learning 을
사용한 CNN

Lab 1-3: ResNet

Deep Residual Learning for image recognition(CVPR 2016)

- 잔여 블록을 이용하여 네트워크의 최적화(optimization) 난이도를 낮춥니다.
- 바로, H(x)를 구하는 것 보다 residual기반의 learning방법으로 F(x) = H(x) – x를 대신 학습합니다.



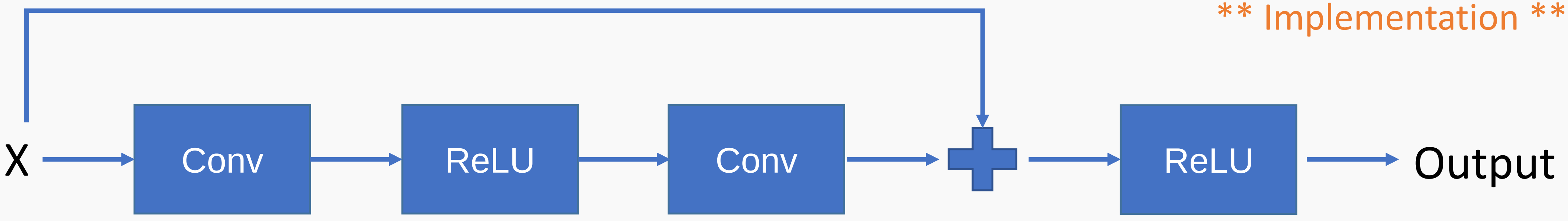
Lab 1-3: ResNet

Deep Residual Learning for image recognition(CVPR 2016) - Residual block

- ResNet을 구성하기 위한 기본 단위 block
- Batch normalization, activation function에 따라 block의 형태가 달라질 수 있음.
- 아래 표시된 residual block은 가장 기본적인 형태의 residual block임.
  (batch norm 사용하지 않음, activation function으로 ReLU를 사용)

```python
class ResBlock(nn.Module):
    def __init(self):
        super(ResBlock, self).__init__()

    def forward(self, x):
        return output

```

** Implementation **



가장 기본적인 형태의
Residual Block

Lab 1-3: ResNet

LAB 1-3 procedure

- MNIST.ipynb의 순서에 따라 구현을 하시면 됩니다.

  1) Resblock
  2) SimpleResNet
  3) Training / validation

```python
class ResBlock(nn.Module):
    def __init(self):
        super(ResBlock, self).__init__()

    def forward(self, x):
        return output

```

** Implementation **(1)

SimpleResNet model 만들기

```python
class SimpleResNet(nn.Module):
    def __init__(self):
        super(SimpleResNet, self).__init__()
        # Your implementation here

    def forward(self, x):
        # Your implementation here
        return  F.log_softmax(x, dim=1)
```

** Implementation **(2)

```python
%%time
epochs = 10
lossv, accv = [], []
for epoch in range(1, epochs + 1):
    train_ResNet(epoch)
    validate_ResNet(lossv, accv)
```

** Training / Validation **(3)

Lab 2: Prerequisite - Colab.

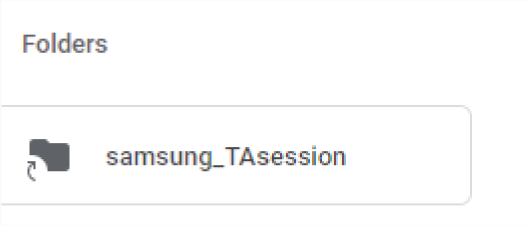
Linking with google drive for colab

1. Login google account

2. Click this link
https://drive.google.com/drive/folders/1ZB4kTn8fXbsshdX1u-Nzz6Xm5-4cb0rq?usp=sharing

3. Samsung_TAsession -> Add shortcut to drive -> My drive

4. Checking shared folder in my drive

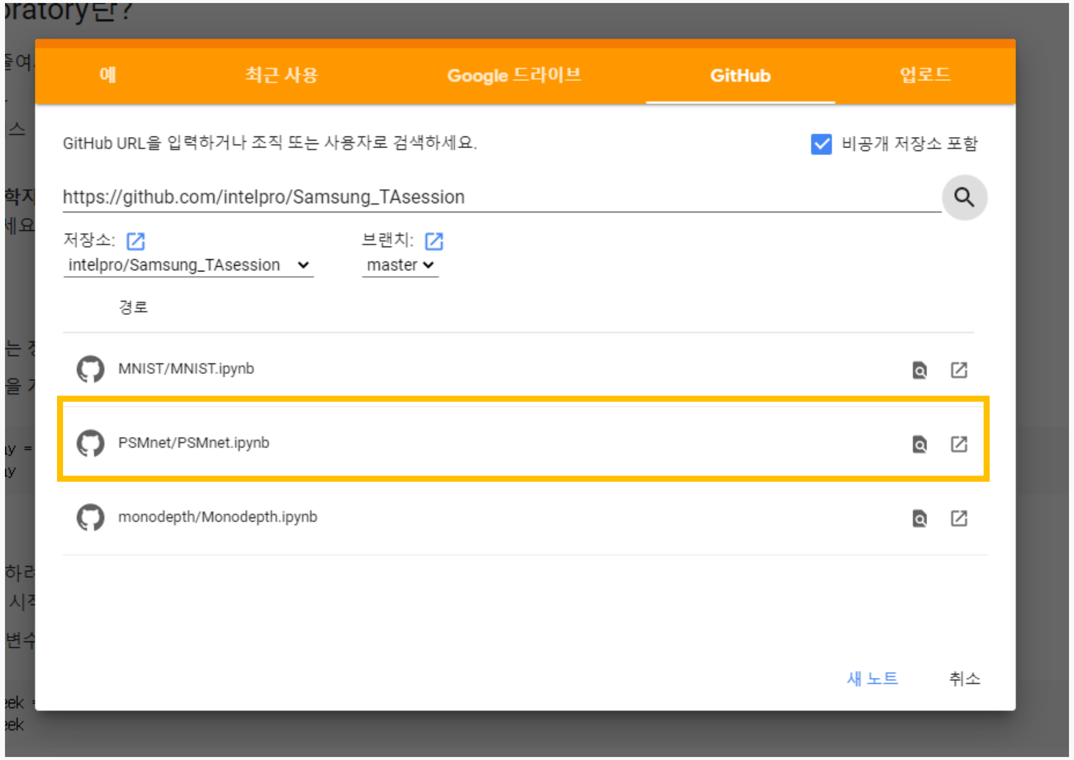


Lab 2: Prerequisite - Colab.

Open Github repository in Colab environment

5. Enter the Colab

6. File -> Open note -> Click on the Github tab -> Copy and paste link below -> open _PSMnet.ipynb
   https://github.com/intelpro/Samsung_TAsession

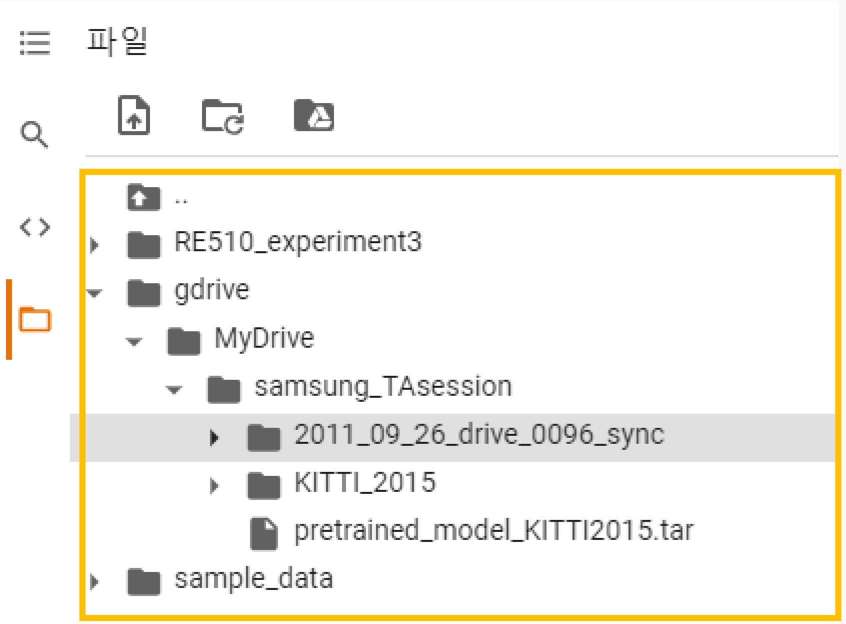


Lab 2: Prerequisite - Colab.

Check the google drive mount

- Confirm that google drive is normally mounted in colab environment

Command line 화면

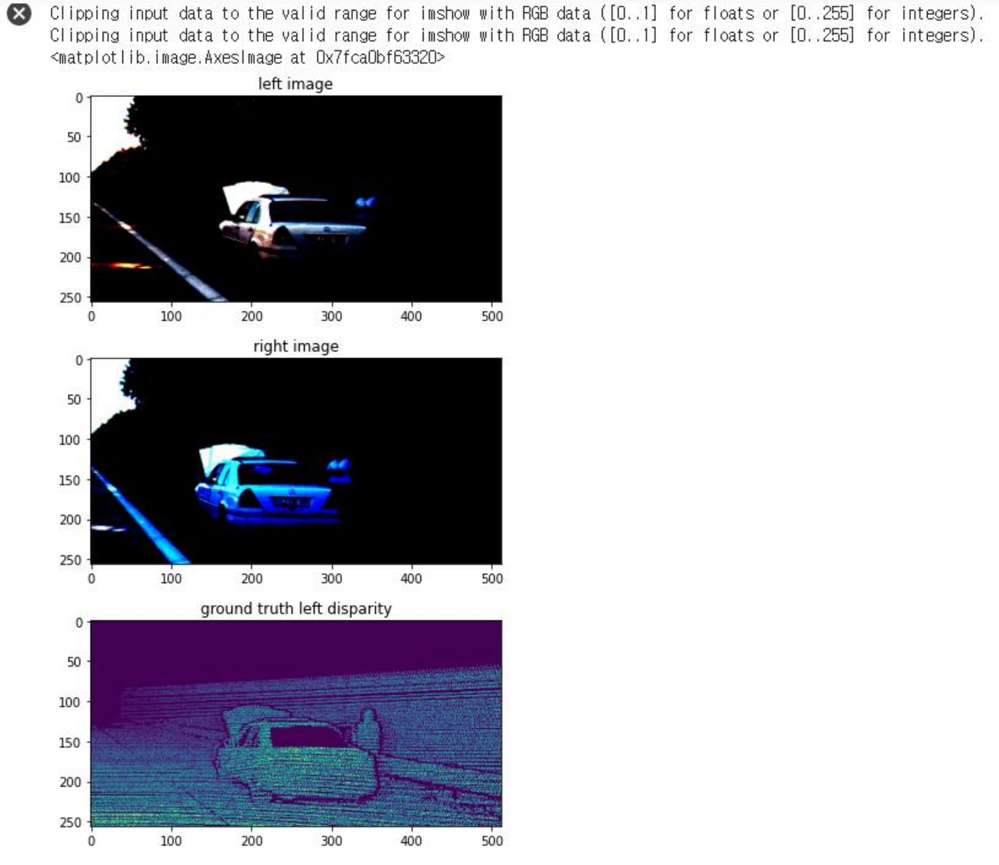


Colab google drive docking 화면

Lab 2: Prerequisite - Colab.

Check data loader

- Import module -> Get dataset string -> Define dataloader -> Check KITTI dataset

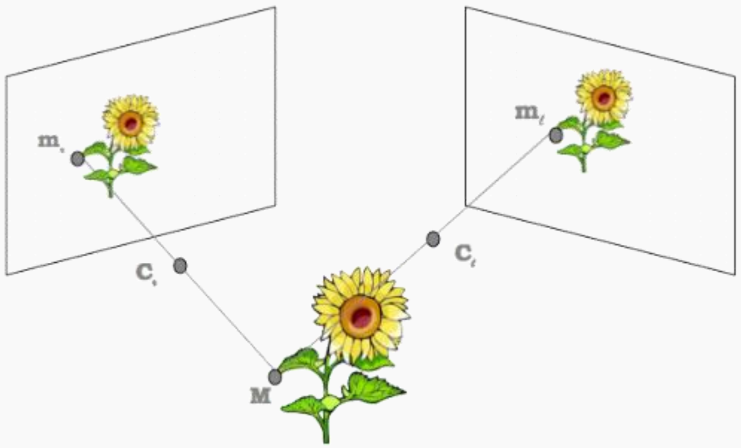


결과 화면

Lab 2: Introduction

What is stereo matching ?

- Stereo matching is the process of finding the pixel in the multi-scopic views that correspond to the same 3D point in the scene.
- The pixel difference between left image and right image along the x-axis is the disparity
- The disparity image can be converted to a depth map by using baseline and focal length.



Correspondence search
between left and right image

Lab 2: Introduction

The major problem with current CNN-based stereo matching method

- It is still hard to find accurate correspondences in ill-posed regions.
  (occlusion areas, repeated patterns, texture-less regions, and reflective surfaces, etc.)

- Solely applying the intensity-consistency constraint b/w different viewpoints is insufficient for accurate correspondence estimation in such ill-posed regions.

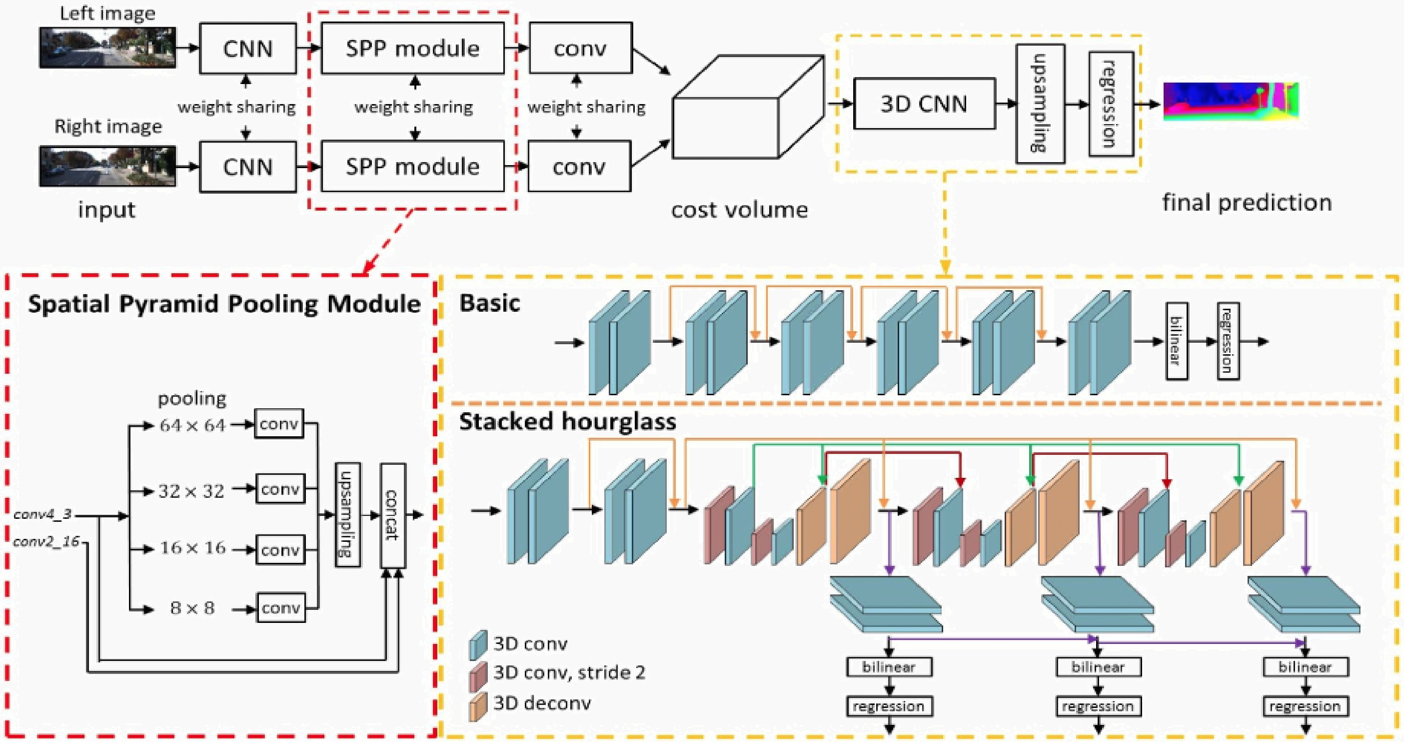
- Global context information must be incorporated into stereo matching(ex, DispNet, CRL, GC-Net)

Then, how to effectively exploit context information?

Lab 2: Introduction

Pyramid stereo matching network

- Spatial pyramid pooling module
- 3D CNN(stacked hour glass)



전체적인 네트워크 아키 텍쳐

Lab 2: Introduction

Pyramid stereo matching network(Network layers)

Basic residual blocks

Name	Layer setting	Output dimension
input		$H \times W \times 3$
CNN		
conv0_1	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_2	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_3	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv1_x	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 16$	$\frac{1}{4}H \times \frac{1}{4}W \times 64$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila = 2	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
conv4_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila= 4	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
SPP module		
branch_1	64×64 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_2	32×32 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_3	16×16 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_4	8×8 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
concat[conv2_16, conv4_3, branch_1, branch_2, branch_3, branch_4]		$\frac{1}{4}H \times \frac{1}{4}W \times 320$
fusion	$3 \times 3, 128$ $1 \times 1, 32$	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
Cost volume		
Concat left and shifted right		$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 64$

3D CNN (stacked hourglass)		
3Dconv0	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 32$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dconv1	$\begin{bmatrix} 3 \times 3 \times 3, 32 \\ 3 \times 3 \times 3, 32 \end{bmatrix}$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack1_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack1_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack1_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack1_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack2_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$ add **3Dstack1_3**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack2_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack2_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack2_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack3_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$ add **3Dstack2_3**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack3_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack3_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack3_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
output_1	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$
output_2	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$ add **output_1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$
output_3	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$ add **output_2**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$

Lab 2: Introduction

Pyramid stereo matching network(Network layers)

Name	Layer setting	Output dimension
input		$H \times W \times 3$
CNN		
conv0_1	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_2	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_3	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv1_x	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 16$	$\frac{1}{4}H \times \frac{1}{4}W \times 64$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila = 2	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
conv4_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila= 4	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
SPP module		
branch_1	64×64 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_2	32×32 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_3	16×16 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_4	8×8 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
concat[conv2_16, conv4_3, branch_1, branch_2, branch_3, branch_4]		$\frac{1}{4}H \times \frac{1}{4}W \times 320$
fusion	$3 \times 3, 128$ $1 \times 1, 32$	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
Cost volume		
Concat left and shifted right		$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 64$

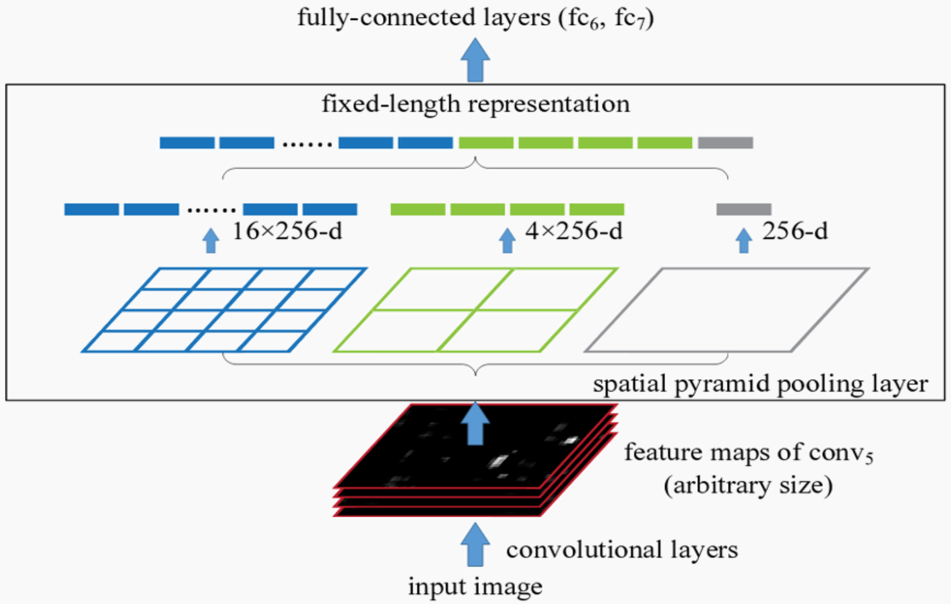
Spatial Pyramid Pooling

3D CNN (stacked hourglass)		
3Dconv0	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 32$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dconv1	$\begin{bmatrix} 3 \times 3 \times 3, 32 \\ 3 \times 3 \times 3, 32 \end{bmatrix}$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack1_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack1_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack1_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack1_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack2_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$ add **3Dstack1_3**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack2_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack2_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack2_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack3_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$ add **3Dstack2_3**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack3_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack3_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack3_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
output_1	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$
output_2	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$ add **output_1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$
output_3	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$ add **output_2**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$

Lab 2: Introduction

Spatial pyramid pooling(SPP)

- To enlarge receptive field
- Enable PSMNet to extend pixel-level features to region-level features with different scales of receptive fields
- Global and local feature clues are used to form the cost volume for reliable disparity estimation

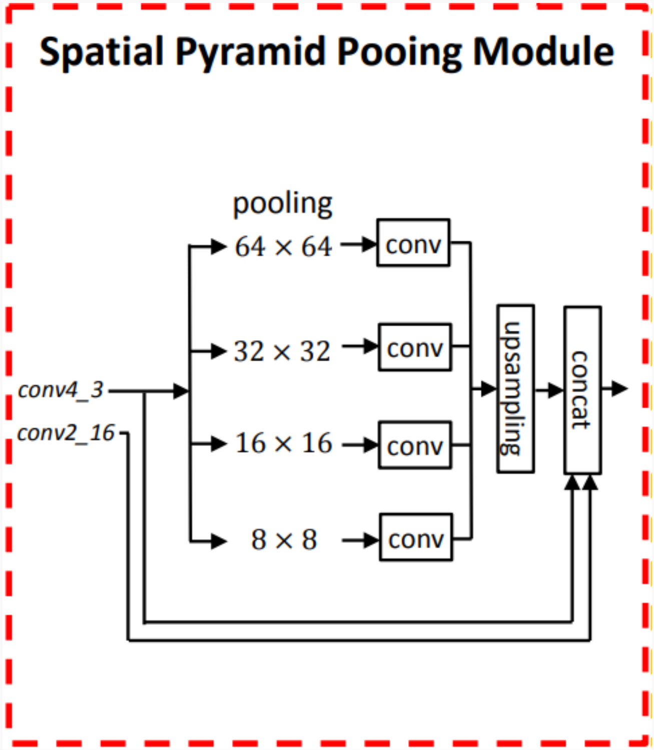


SPP Module

Lab 2: Introduction

Network layers - Spatial pyramid pooling(SPP)

- The relationship between and object and its sub-regions is learned by the SPP module to incorporate hierarchical context information



SPP module		
branch_1	64 × 64 avg. pool 3 × 3, 32 bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_2	32 × 32 avg. pool 3 × 3, 32 bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_3	16 × 16 avg. pool 3 × 3, 32 bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_4	8 × 8 avg. pool 3 × 3, 32 bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
concat[conv2_16, conv4_3, branch_1, branch_2, branch_3, branch_4]		$\frac{1}{4}H \times \frac{1}{4}W \times 320$
fusion	3 × 3, 128 1 × 1, 32	$\frac{1}{4}H \times \frac{1}{4}W \times 32$

Lab 2: Introduction

Network layers - 3D CNN

Name	Layer setting	Output dimension
input		$H \times W \times 3$
CNN		
conv0_1	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_2	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_3	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv1_x	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 16$	$\frac{1}{4}H \times \frac{1}{4}W \times 64$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila = 2	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
conv4_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila= 4	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
SPP module		
branch_1	64×64 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_2	32×32 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_3	16×16 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_4	8×8 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
concat[conv2_16, conv4_3, branch_1, branch_2, branch_3, branch_4]		$\frac{1}{4}H \times \frac{1}{4}W \times 320$
fusion	$3 \times 3, 128$ $1 \times 1, 32$	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
Cost volume		
Concat left and shifted right		$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 64$

3D CNN (stacked hourglass)		
3Dconv0	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 32$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dconv1	$\begin{bmatrix} 3 \times 3 \times 3, 32 \\ 3 \times 3 \times 3, 32 \end{bmatrix}$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack1_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack1_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack1_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack1_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack2_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$ add **3Dstack1_3**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack2_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack2_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack2_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
3Dstack3_1	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$ add **3Dstack2_3**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack3_2	$3 \times 3 \times 3, 64$ $3 \times 3 \times 3, 64$	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 64$
3Dstack3_3	deconv $3 \times 3 \times 3, 64$ add **3Dstack1_1**	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 64$
3Dstack3_4	deconv $3 \times 3 \times 3, 32$ add **3Dconv1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 32$
output_1	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$
output_2	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$ add **output_1**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$
output_3	$3 \times 3 \times 3, 32$ $3 \times 3 \times 3, 1$ add **output_2**	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 1$

Lab 2: Introduction

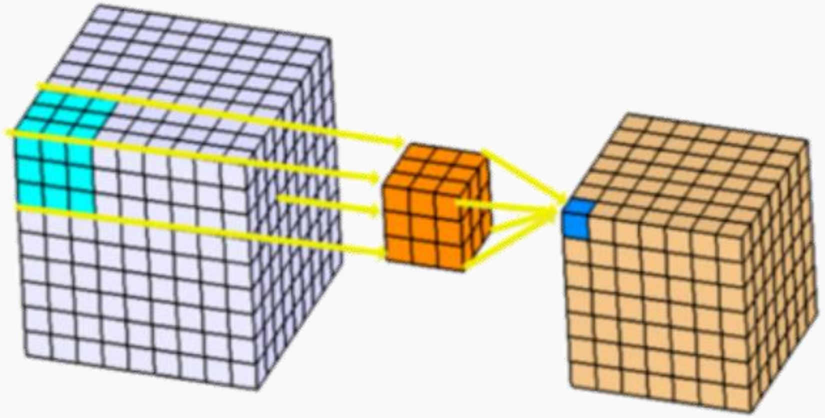
Network layers - 3D CNN

- Cost volume is 5D dimension
  (batch size x disparity range x feature map dimension x height x width)

- To process 5D dimension input in convolution neural network, we should use 3d convolution instead of 2D convolution.

- 3D convolutions applies a 3 dimensional filter to the dataset and the filter moves 3-direction (x, y, z) to calcuate the low level feature representations. Their output shape is a 3 dimensional volume space such as cube or cuboid.

- 3D convolution are mainly used in video domain to deal with spatio-temporal information

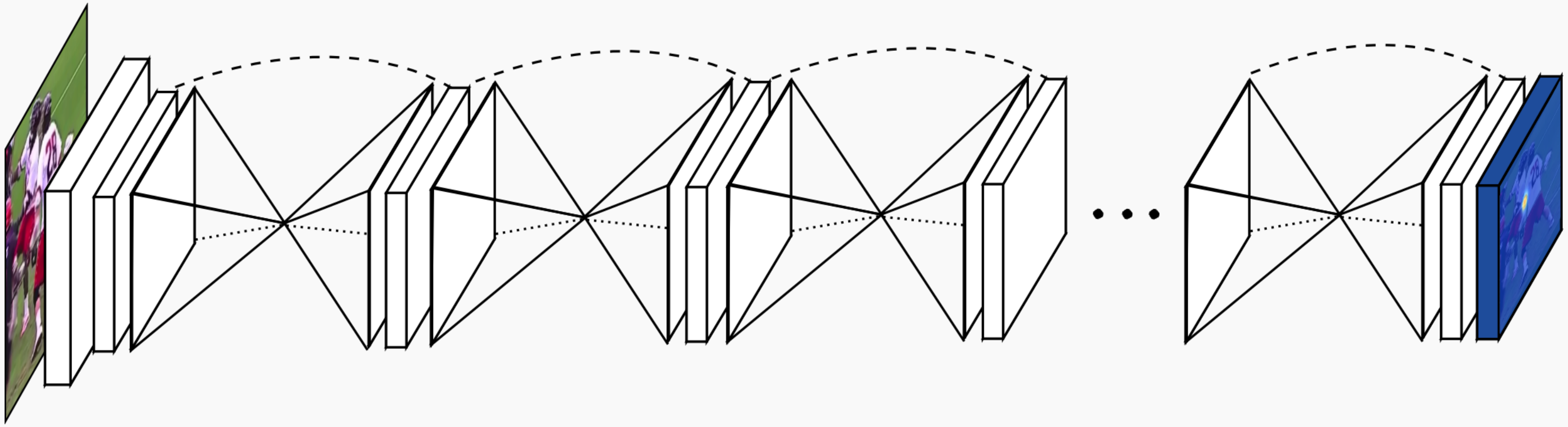


3D convolution example

Lab 2: Introduction

Network layers - 3D CNN(stacked hour glass)

- To regularize cost volume, they have used stacked hourglass module.
- It is necessary to capture local information well and simultaneously capture global context information well to get a good performance of DNN.
- Repeatedly process the cost volume in a top-down/bottom-up manner(encoding-decoding process) to further improve the utilization of global information.

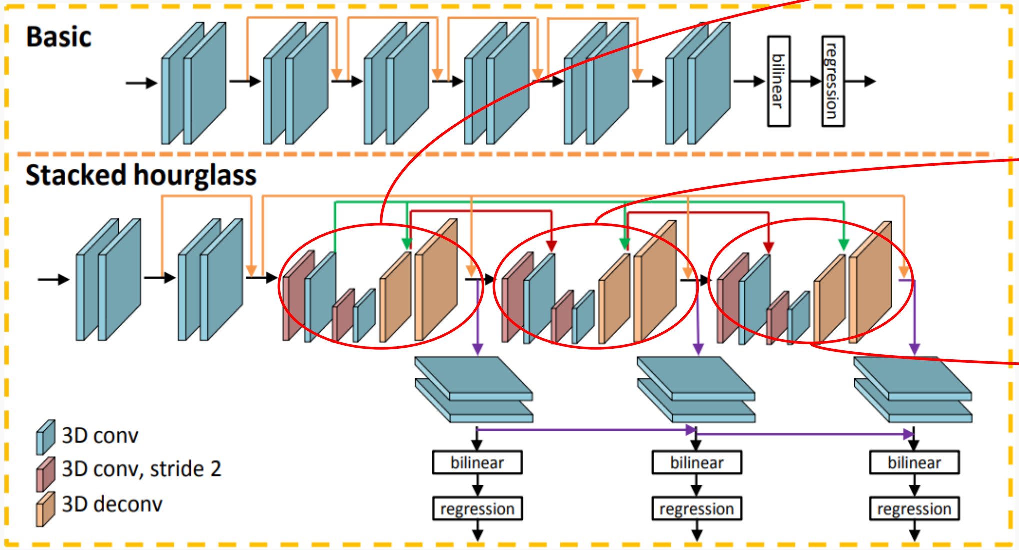


Stacked hour glass in Human Pose estimation

Lab 2: Introduction

Network layers - 3D CNN(stacked hour glass)

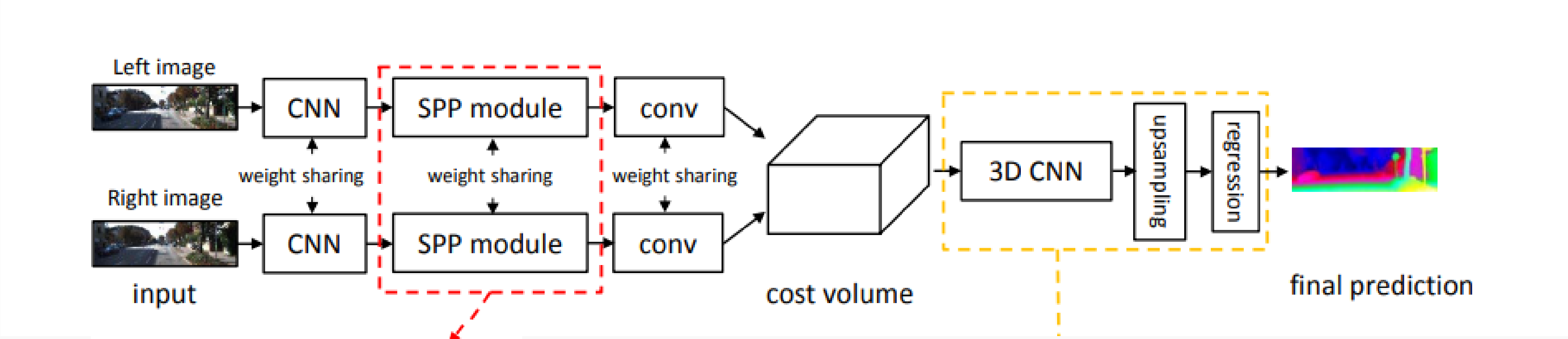
- Global context information by regularizing the cost volume

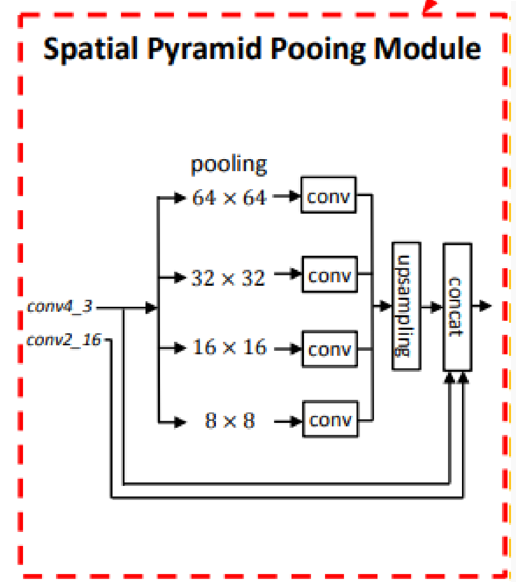


3D CNN (stacked hourglass)		
3Dconv0	$3\times3\times3, 32$ $3\times3\times3, 32$	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times32$
3Dconv1	$\begin{bmatrix}3\times3\times3, 32\\3\times3\times3, 32\end{bmatrix}$	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times32$
3Dstack1_1	$3\times3\times3, 64$ $3\times3\times3, 64$	$\frac{1}{8}D\times\frac{1}{8}H\times\frac{1}{8}W\times64$
3Dstack1_2	$3\times3\times3, 64$ $3\times3\times3, 64$	$\frac{1}{16}D\times\frac{1}{16}H\times\frac{1}{16}W\times64$
3Dstack1_3	deconv $3\times3\times3, 64$ add **3Dstack1_1**	$\frac{1}{8}D\times\frac{1}{8}H\times\frac{1}{8}W\times64$
3Dstack1_4	deconv $3\times3\times3, 32$ add **3Dconv1**	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times32$
3Dstack2_1	$3\times3\times3, 64$ $3\times3\times3, 64$ add **3Dstack1_3**	$\frac{1}{8}D\times\frac{1}{8}H\times\frac{1}{8}W\times64$
3Dstack2_2	$3\times3\times3, 64$ $3\times3\times3, 64$	$\frac{1}{16}D\times\frac{1}{16}H\times\frac{1}{16}W\times64$
3Dstack2_3	deconv $3\times3\times3, 64$ add **3Dstack1_1**	$\frac{1}{8}D\times\frac{1}{8}H\times\frac{1}{8}W\times64$
3Dstack2_4	deconv $3\times3\times3, 32$ add **3Dconv1**	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times32$
3Dstack3_1	$3\times3\times3, 64$ $3\times3\times3, 64$ add **3Dstack2_3**	$\frac{1}{8}D\times\frac{1}{8}H\times\frac{1}{8}W\times64$
3Dstack3_2	$3\times3\times3, 64$ $3\times3\times3, 64$	$\frac{1}{16}D\times\frac{1}{16}H\times\frac{1}{16}W\times64$
3Dstack3_3	deconv $3\times3\times3, 64$ add **3Dstack1_1**	$\frac{1}{8}D\times\frac{1}{8}H\times\frac{1}{8}W\times64$
3Dstack3_4	deconv $3\times3\times3, 32$ add **3Dconv1**	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times32$
output_1	$3\times3\times3, 32$ $3\times3\times3, 1$	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times1$
output_2	$3\times3\times3, 32$ $3\times3\times3, 1$ add **output_1**	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times1$
output_3	$3\times3\times3, 32$ $3\times3\times3, 1$ add **output_2**	$\frac{1}{4}D\times\frac{1}{4}H\times\frac{1}{4}W\times1$

Lab 2: Assignment

Spatial Pyramid pooling module





Note

- A method to increase the receptive field of CNN while average pooling window size to multiple levels of 64, 32, 16, 8
- SPP module is widely used in various fields such as object detection, semantic segmentation, image classification, as well as stereo matching

Lab 2: Assignment

Spatial Pyramid pooling module - Feature extraction

Name	Layer setting	Output dimension
input		$H \times W \times 3$
CNN		
conv0_1	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_2	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv0_3	$3 \times 3, 32$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv1_x	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$	$\frac{1}{2}H \times \frac{1}{2}W \times 32$
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 16$	$\frac{1}{4}H \times \frac{1}{4}W \times 64$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila = 2	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
conv4_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 3$, dila= 4	$\frac{1}{4}H \times \frac{1}{4}W \times 128$
SPP module		
branch_1	64×64 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_2	32×32 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_3	16×16 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
branch_4	8×8 avg. pool $3 \times 3, 32$ bilinear interpolation	$\frac{1}{4}H \times \frac{1}{4}W \times 32$
concat[conv2_16, conv4_3, branch_1, branch_2, branch_3, branch_4]		$\frac{1}{4}H \times \frac{1}{4}W \times 320$
fusion	$3 \times 3, 128$ $1 \times 1, 32$	$\frac{1}{4}H \times \frac{1}{4}W \times 32$

```python
## Your implementation Here
self.branch1 = nn.Sequential(nn.AvgPool2d((None, None), stride=(None,None)),
                             convbn(None, None, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.branch2 = nn.Sequential(nn.AvgPool2d((None, None), stride=(None,None)),
                             convbn(None, None, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.branch3 = nn.Sequential(nn.AvgPool2d((None, None), stride=(None,None)),
                             convbn(None, None, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.branch4 = nn.Sequential(nn.AvgPool2d((None, None), stride=(None,None)),
                             convbn(None, None, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))
## Your implementation end
```

Assignment details

- Fill in the part with None Pooling level is 64, 32, 16, 8
- The stride number should be filled to fit the pooling window size.
- Note that the channel changes from 128 to 32 channels after the feature map extracted from conv4_x passes each branch
- If you want to know the definition of convbn, refer to the section called def convbn(…).

Lab 2: Assignment

Hourglass module - Cost volume regularization

Layer	Stride	Kernel size	Padding	Feature dims
Conv1(Conv3D + BN3D + ReLU)	2	3	1	2배
Conv2(Conv3D + BN)	1	3	1	1배
Conv3(Conv3D + BN3D + ReLU)	2	3	1	1배
Conv4(Conv3D + BN3D + ReLU)	1	3	1	1배
Conv5(Transpose Conv 3D + BN3D)	2	3	1	1배
Conv5(Transpose Conv 3D + BN3D)	2	3	1	0.5배

```python
class hourglass(nn.Module):
    def __init__(self, inplanes):
        super(hourglass, self).__init__()
        ## Your implementation Here - Hourglass module
        self.conv1 = NotImplemented
        self.conv2 = NotImplemented
        self.conv3 = NotImplemented
        self.conv4 = NotImplemented
        self.conv5 = NotImplemented
        self.conv6 = NotImplemented
        ## Your implementation end - Hourglass module

    def forward(self, x, presqu, postsqu):

        out  = self.conv1(x) #in:1/4 out:1/8
        pre  = self.conv2(out) #in:1/8 out:1/8
        if postsqu is not None:
           pre = F.relu(pre + postsqu, inplace=True)
        else:
           pre = F.relu(pre, inplace=True)

        out  = self.conv3(pre) #in:1/8 out:1/16
        out  = self.conv4(out) #in:1/16 out:1/16

        if presqu is not None:
           post = F.relu(self.conv5(out)+presqu, inplace=True) #in:1/16 out:1/8
        else:
           post = F.relu(self.conv5(out)+pre, inplace=True)

        out  = self.conv6(post)  #in:1/8 out:1/4
        return out, pre, post
```

** Implementation **

Lab 2: Assignment

If you have properly implemented the spatial pooling module and hourglass module..

When running training code with KITTI dataset

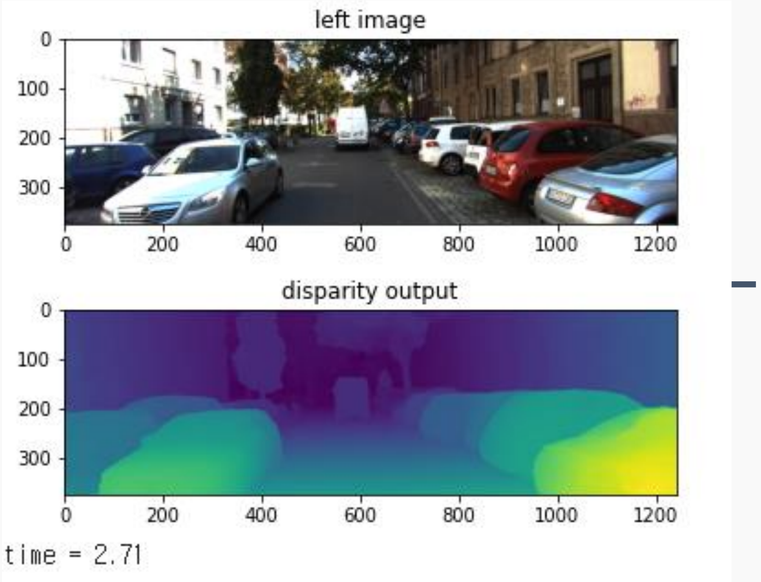
```
... This is 0-th epoch
    /usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2796:
      warnings.warn("nn.functional.upsample is deprecated. Use nn.functi
    /usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973:
      "See the documentation of nn.Upsample for details.".format(mode))
    /usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973:
      "See the documentation of nn.Upsample for details.".format(mode))
    /usr/local/lib/python3.6/dist-packages/torch/nn/_reduction.py:43: Us
      warnings.warn(warning.format(ret))
    Iter 0 training loss = 76.164 , time = 2.01
    Iter 3 training loss = 9.776 , time = 1.76
    Iter 6 training loss = 7.997 , time = 1.77
    Iter 9 training loss = 13.371 , time = 1.78
    Iter 12 training loss = 11.346 , time = 1.77
    Iter 15 training loss = 10.733 , time = 1.75
    Iter 18 training loss = 7.748 , time = 1.75
    Iter 21 training loss = 7.496 , time = 1.74
    Iter 24 training loss = 6.336 , time = 1.73
    Iter 27 training loss = 6.175 , time = 1.74
    Iter 30 training loss = 7.019 , time = 1.74
    Iter 33 training loss = 5.445 , time = 1.74
    Iter 36 training loss = 7.691 , time = 1.74
    Iter 39 training loss = 3.301 , time = 1.73
    Iter 42 training loss = 9.453 , time = 1.72
    Iter 45 training loss = 6.158 , time = 1.72
    Iter 48 training loss = 5.845 , time = 1.74
    Iter 51 training loss = 6.859 , time = 1.74
```

You can check the loss reduction..

Lab 2: Assignment

If you have properly implemented the spatial pooling module..

When running test with KITTI test sample



Confirm disparity output

Lab 2: Assignment

For Your Information…

nn.AvgPool2d

https://pytorch.org/docs/master/generated/torch.nn.AvgPool2d.html

AVGPOOL2D

CLASS torch.nn.AvgPool2d(*kernel_size: Union[T, Tuple[T, T]], stride: Optional[Union[T, Tuple[T, T]]] = None, padding: Union[T, Tuple[T, T]] = 0, ceil_mode: bool = False, count_include_pad: bool = True, divisor_override: bool = None*) [SOURCE]

Parameters

- **kernel_size** – the size of the window
- **stride** – the stride of the window. Default value is `kernel_size`
- **padding** – implicit zero padding to be added on both sides
- **ceil_mode** – when True, will use *ceil* instead of *floor* to compute the output shape
- **count_include_pad** – when True, will include the zero-padding in the averaging calculation
- **divisor_override** – if specified, it will be used as divisor, otherwise `kernel_size` will be used

Appendix: Answer(Lab1)

SimpleCNN

```
class SimpleCNN(nn.Module):
    """ConvNet -> Max_Pool -> RELU -> ConvNet -> Max_Pool -> RELU -> FC -> RELU -> FC -> SOFTMAX"""
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 3, 1)
        self.maxpool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(20, 50, 3, 1)
        self.maxpool2 =nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(50, 20, 3, 1)
        self.fc1 = nn.Linear(3*3*20, 100)
        self.fc2 = nn.Linear(100, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.maxpool1(x)
        x = F.relu(self.conv2(x))
        x = self.maxpool2(x)
        x = F.relu(self.conv3(x))
        x = x.view(-1, 3*3*20)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Appendix: Answer(Lab1)

SimpleVGG

```python
class SimpleVGG(nn.Module):
    def __init__(self):
        super(SimpleVGG, self).__init__()
        self.act = nn.ReLU()
        self.maxpool2d = nn.MaxPool2d(2, 2)
        self.conv1 = nn.Conv2d(1, 64, 3, 1, 2)
        self.conv2 = nn.Conv2d(64, 128, 3, 1, 2)
        self.conv3 = nn.Conv2d(128, 256, 3, 1, 2)
        self.conv4 = nn.Conv2d(256, 256, 3, 1, 1)
        self.conv5 = nn.Conv2d(256, 512, 3, 1, 1)
        self.conv6 = nn.Conv2d(512, 512, 3, 1, 1)
        self.conv7 = nn.Conv2d(512, 512, 3, 1, 1)
        self.conv8 = nn.Conv2d(512, 512, 3, 1, 1)
        # Your implementation here
        self.Linear1 = nn.Linear(512, 100)
        self.drop = nn.Dropout(0.2)
        self.Linear2 = nn.Linear(100, 100)
        self.last_linear = nn.Linear(100, 10)
```

```python
def forward(self, x):
    x = self.act(self.conv1(x))
    x = self.maxpool2d(x)
    x = self.act(self.conv2(x))
    x = self.maxpool2d(x)
    x = self.act(self.conv4(self.conv3(x)))
    x = self.maxpool2d(x)
    x = self.act(self.conv6(self.conv5(x)))
    x = self.maxpool2d(x)
    x = self.act(self.conv8(self.conv7(x)))
    x = self.maxpool2d(x)
    x = x.view(-1, 512)

    x = F.relu(self.Linear1(x))
    x = self.drop(x)
    x = F.relu(self.Linear2(x))
    x = self.drop(x)
    x = F.relu(self.last_linear(x))
    return  F.log_softmax(x, dim=1)
```

Appendix: Answer(Lab1)

ResNet(Residual block)

```
class ResBlock(nn.Module):
    """
    Residual block
    """
    def __init__(self, in_chs, stride,  activation='relu', batch_norm=False):
        super(ResBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_chs, in_chs*stride, 3, 1, 1, bias=False)
        self.bn1 = nn.BatchNorm2d(in_chs*stride)
        self.conv2 = nn.Conv2d(in_chs*stride, in_chs*stride, 3, 1, 1, bias=False)
        self.bn2 = nn.BatchNorm2d(in_chs*stride)
        self.shortcut = nn.Sequential()
        self.relu=nn.ReLU()
    def forward(self, x):
        short = x.clone()
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        short=self.bn1(self.conv1(short))
        out += self.shortcut(short)
        out = self.relu(out)
        return out
```

Appendix: Answer(Lab1)

ResNet

```
class SimpleResNet(nn.Module):
    def __init__(self):
        super(SimpleResNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, 7, 2, 3, bias=False)
        self.relu=nn.ReLU()
        self.max_pool = nn.MaxPool2d(3, 2, 1)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = ResBlock(64,1)
        self.layer2 = ResBlock(64,2)
        self.layer3 = ResBlock(128,2)
        self.layer4 = ResBlock(256,2)
        self.avgpool = nn.AdaptiveAvgPool2d((1,1))
        self.linear = nn.Linear(512, 10)



    def forward(self, x):
        out=self.conv1(x)
        out=self.bn1(out)
        out = self.relu(out)
        out = self.max_pool(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.avgpool(out)
        out=torch.flatten(out,1)
        out = self.linear(out)
        return  F.log_softmax(out, dim=1)
```

Appendix: Answer(Lab2)

SPP

```
self.branch1 = nn.Sequential(nn.AvgPool2d((64, 64), stride=(64,64)),
                             convbn(128, 32, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.branch2 = nn.Sequential(nn.AvgPool2d((32, 32), stride=(32,32)),
                             convbn(128, 32, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.branch3 = nn.Sequential(nn.AvgPool2d((16, 16), stride=(16,16)),
                             convbn(128, 32, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.branch4 = nn.Sequential(nn.AvgPool2d((8, 8), stride=(8,8)),
                             convbn(128, 32, 1, 1, 0, 1),
                             nn.ReLU(inplace=True))

self.lastconv = nn.Sequential(convbn(320, 128, 3, 1, 1, 1),
                              nn.ReLU(inplace=True),
                              nn.Conv2d(128, 32, kernel_size=1, padding=0, stride = 1, bias=False))
```

Appendix: Answer(Lab2)

Hourglass

```
class hourglass(nn.Module):
    def __init__(self, inplanes):
        super(hourglass, self).__init__()

        self.conv1 = nn.Sequential(convbn_3d(inplanes, inplanes*2, kernel_size=3, stride=2, pad=1),
                                   nn.ReLU(inplace=True))

        self.conv2 = convbn_3d(inplanes*2, inplanes*2, kernel_size=3, stride=1, pad=1)

        self.conv3 = nn.Sequential(convbn_3d(inplanes*2, inplanes*2, kernel_size=3, stride=2, pad=1),
                                   nn.ReLU(inplace=True))

        self.conv4 = nn.Sequential(convbn_3d(inplanes*2, inplanes*2, kernel_size=3, stride=1, pad=1),
                                   nn.ReLU(inplace=True))

        self.conv5 = nn.Sequential(nn.ConvTranspose3d(inplanes*2, inplanes*2, kernel_size=3, padding=1, output_padding=1, stride=2,bias=False),
                                   nn.BatchNorm3d(inplanes*2)) #+conv2

        self.conv6 = nn.Sequential(nn.ConvTranspose3d(inplanes*2, inplanes, kernel_size=3, padding=1, output_padding=1, stride=2,bias=False),
                                   nn.BatchNorm3d(inplanes)) #+x

    def forward(self, x ,presqu, postsqu):

        out  = self.conv1(x) #in:1/4 out:1/8
        pre  = self.conv2(out) #in:1/8 out:1/8
        if postsqu is not None:
           pre = F.relu(pre + postsqu, inplace=True)
        else:
           pre = F.relu(pre, inplace=True)

        out  = self.conv3(pre) #in:1/8 out:1/16
        out  = self.conv4(out) #in:1/16 out:1/16

        if presqu is not None:
           post = F.relu(self.conv5(out)+presqu, inplace=True) #in:1/16 out:1/8
        else:
           post = F.relu(self.conv5(out)+pre, inplace=True)

        out  = self.conv6(post)  #in:1/8 out:1/4

        return out, pre, post
```