Assignment #2: CNN Implementation

Paul Hongsuck Seo

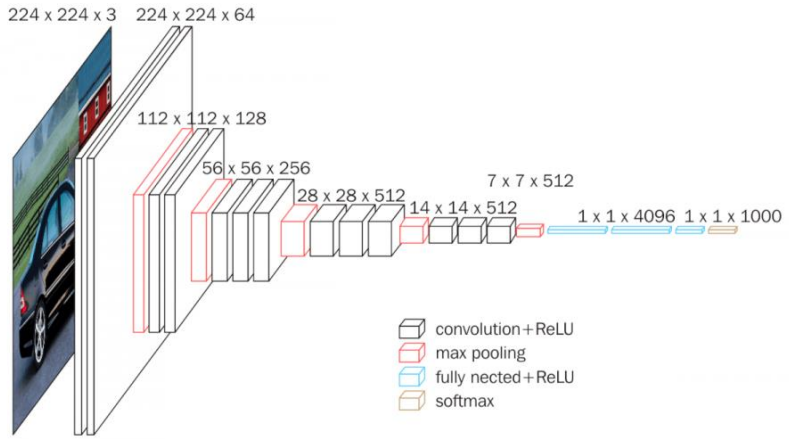
Korea University

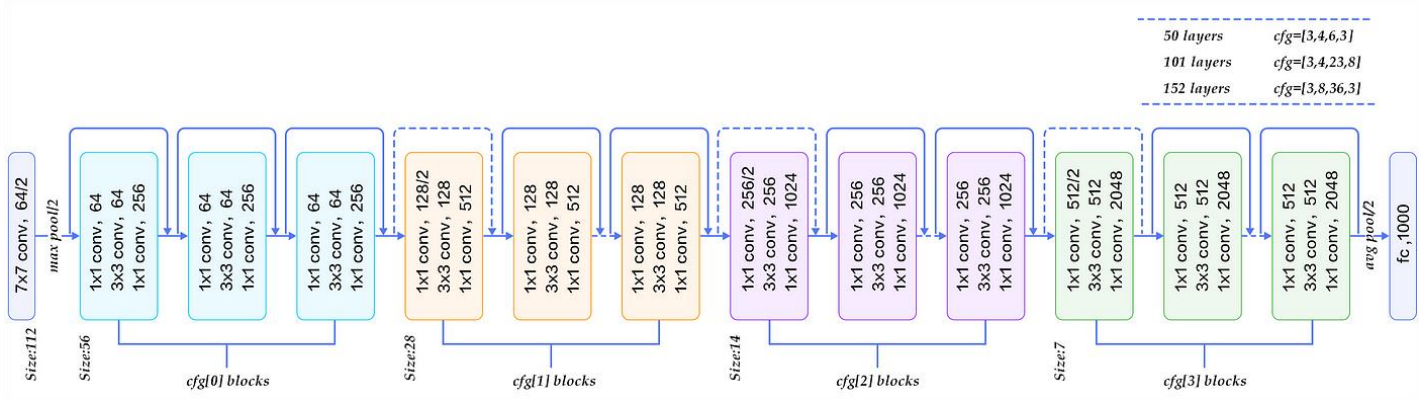
KOREA UNIVERSITY

MIIL Multimodal Interactive Intelligence Laboratory

CNN Implementation

Implement CNN Classifier



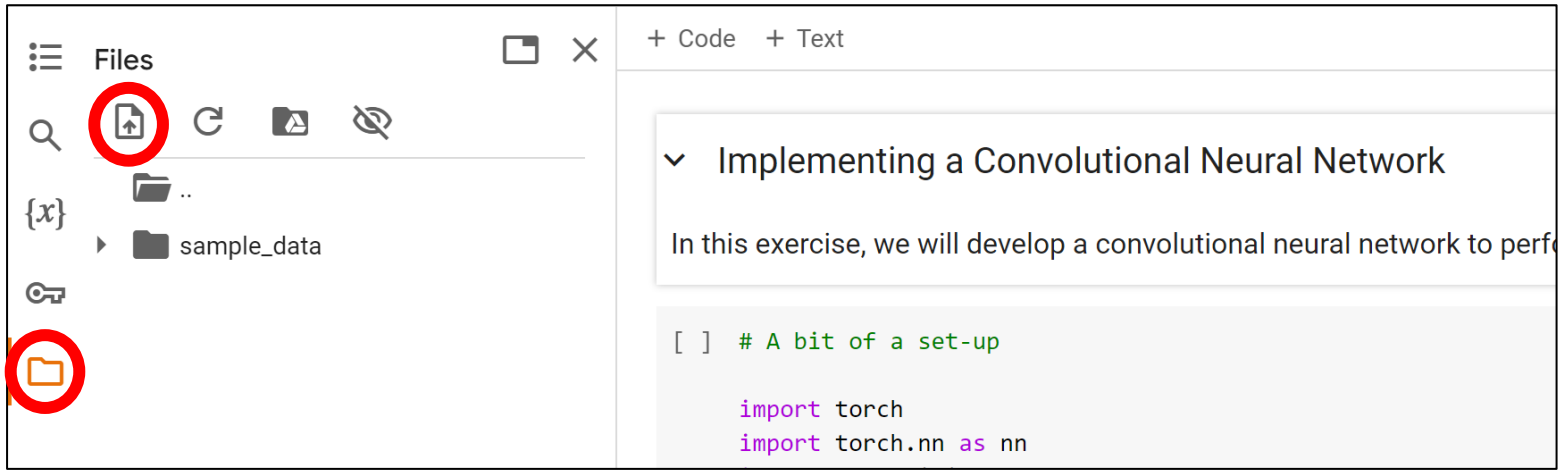


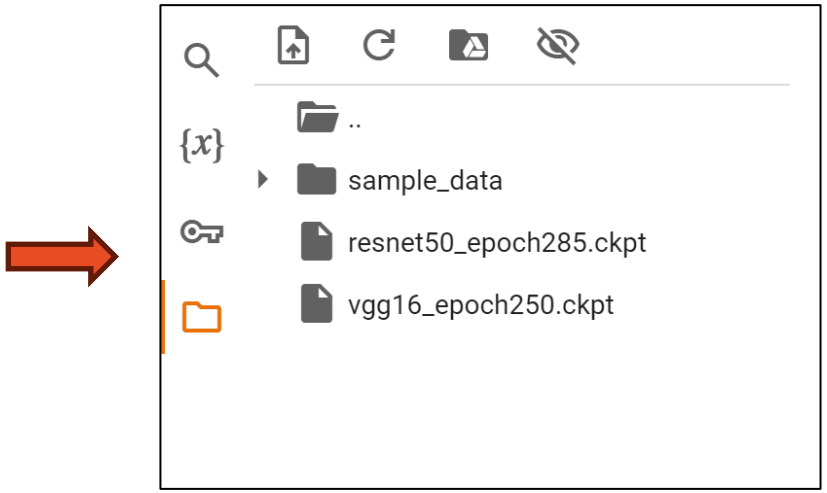
- Perform the image classification using “CIFAR-10” dataset with CNN (VGG and ResNet).
- The Ipython Notebook “CNN_Implementation.ipynb” will walk you through the implementation of CNN classifier.

CNN Implementation

Requirements

- Download the attached zip file
- Open the **CNN_Implementation.ipynb** with colab notebook
- Upload the other .ckpt files to session storage





CNN Implementation

Instructions

- Follow the instructions in the **CNN_Implementation.ipynb** notebook to complete the assignment.
  - Load the **“CIFAR-10”** data **(No need for any modifications)**
  - Practice training using the provided VGG code **(No need for any modifications)**
  - Complete the ResNet code and train the ResNet model
- **CNN_Implementation.ipynb** is based on PyTorch.

CNN Implementation

Instructions

- After practicing with the provided VGG code, complete the **resnet50_skeleton.py** and the corresponding cell in **CNN_Implementation.ipynb**
- Training ResNet-50 Model using the other cells in the **CNN_Implementation.ipynb**

```python
########################################################################
# Question 1 : Implement the "bottle neck building block" part.
# Hint : Think about difference between downsample True and False. How we make the difference by code?
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, middle_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        self.downsample = downsample

        if self.downsample:
            self.layer = nn.Sequential(
                ########################################
                ############## fill in here (20 points)
                # Hint : use these functions (conv1x1, conv3x3)
                ########################################
            )
            self.downsize = conv1x1(in_channels, out_channels, 2, 0)

        else:
            self.layer = nn.Sequential(
                ########################################
                ############ fill in here (20 points)
                ########################################
            )
            self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)

```


```python
########################################################################
# Question 1 : Implement the "bottle neck building block" part.
# Hint : Think about difference between downsample True and False. How we make the difference by code?
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, middle_channels, out_channels, downsample=False):
        super(ResidualBlock, self).__init__()
        self.downsample = downsample

        if self.downsample:
            self.layer = nn.Sequential(
                ########################################
                ############## fill in here (20 points)
                # Hint : use these functions (conv1x1, conv3x3)
                ########################################
            )
            self.downsize = conv1x1(in_channels, out_channels, 2, 0)

        else:
            self.layer = nn.Sequential(
                ########################################
                ############ fill in here (20 points)
                ########################################
            )
            self.make_equal_channel = conv1x1(in_channels, out_channels, 1, 0)
```

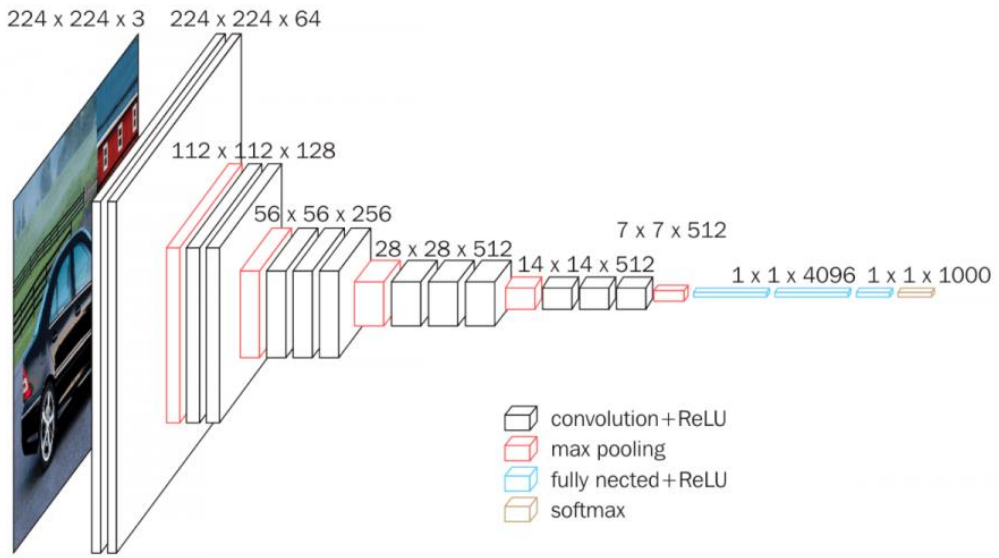
CIFAR-10 Dataset

- What is CIFAR-10?
  - CIFAR-10 stands for the Canadian Institute For Advanced Research.
  - The dataset consists of 60,000 32x32 color images in 10 different classes.
  - https://www.cs.toronto.edu/~kriz/cifar.html
- Dataset Composition
  - Training Set: 50,000 images.
  - Test Set: 10,000 images.
  - Each image is a 32x32 color pixel image (32x32 pixels x 3 color channels).

VGG Networks – *Practice*

VGG Networks

- VGG is a deep convolutional neural network architecture developed by researchers at the University of Oxford
- VGG use small 3x3 convolutional filters consistently across the entire network.
- https://arxiv.org/abs/1409.1556



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG Networks – *Practice*

VGG Networks

- Design Characteristics
  - The VGG family consists of several models, named from A to E.
  - VGG networks range from 11 to 19 layers, with increasing depth.
  - Starting with 64 channels, the number doubles after each max-pooling layer, reaching 512 channels.
  - All configurations follow a similar design but differ in depth and the number of convolutional layers.

VGG Networks – *Practice*

Train “VGG-16”

- Code Flow

```
model = vgg16().to(device)
```

Initialize

```
def vgg16():
    # cfg shows 'kernel size'
    # 'M' means 'max pooling'
    cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']
    return VGG(make_layers(cfg))
```

```
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

Make layers
- **‘M’ : Maxpool**
- **Else: Convolution + ReLu**

```
class VGG(nn.Module):
    def __init__(self, features):
        super(VGG, self).__init__()
        self.features = features
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(512, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(512, 10),
        )
        # Initialize weights
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
                m.bias.data.zero_()

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

Forward

```
# Forward pass
outputs = model(images)
```

VGG Networks – *Practice*

Train “VGG-16” model with “CIFAR-10” datasets

- Optimize parameters with Adam optimizer and cross Entropy Loss
  - Use “VGG-16” model with torch.nn library
  - Get “CIFAR-10” Dataset with torchvision library
- Procedure
  - Load the trained model (which is given)
  - Train it with CPU or GPU
  - You can use a trained checkpoint parameters of 250 epochs. You will train model only 1 epoch.

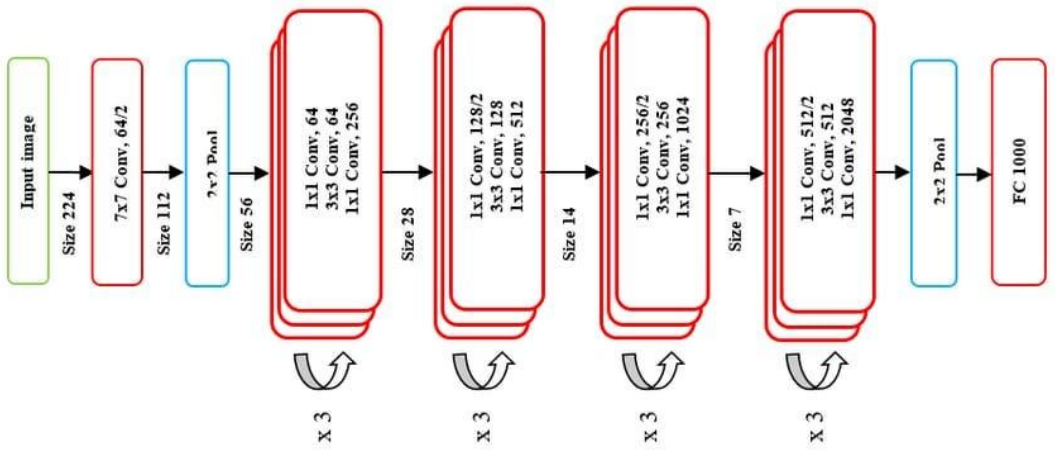
```
model = vgg16().to(device)
PATH = './vgg16_epoch250.ckpt'

checkpoint = torch.load(PATH, map_location=torch.device('cpu'))
model.load_state_dict(checkpoint)
```

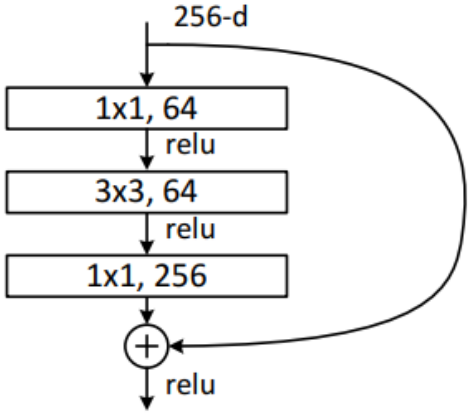
```
# Hyper-parameters
num_epochs = 1
learning_rate = 0.001

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

ResNet – *Question*

ResNet

- ResNet is a deep convolutional neural network architecture developed by researchers at Microsoft Research.
- ResNet introduced the concept of "residual learning" to address the problem of vanishing gradients in very deep networks.
- The network uses skip connections to bypass one or more layers.
- https://arxiv.org/abs/1512.03385





Note) The homework codes are different from the original Resnet50 structure. When you do homework, please refer to the table on page 13,14 not left figure.

ResNet – *Question*

ResNet

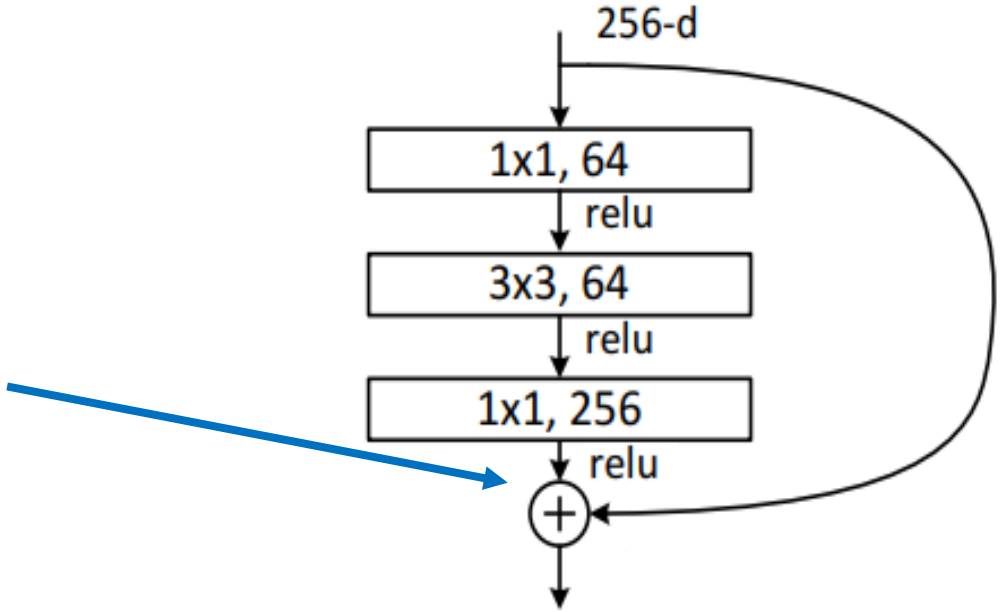
- Design Characteristics
  - The ResNet family consists of several models, such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, with varying depths.
  - Each residual block contains two or more convolutional layers and a shortcut connection that adds the input of the block directly to its output.
  - This design enables very deep networks without the degradation problem, making it possible to build networks with hundreds of layers.

ResNet – *Question*

Implement “ResNet-50”

- ✓ **Question1**: Implement the “bottleneck building block”
- Bottleneck building block (residual block)
  - For each residual function F, we use a stack of 3 layers. The three layers are 1x1, 3x3 and 1x1 convolutions.
  - Input: x
  - Output: $\mathrm{F}(x) + x$

Note) This structure is different form the original Building block on the Resnet. In the original ResNet, “relu” is executed after ⊕ operation”.



ResNet – *Question*

Implement “ResNet-50”

- ✓ **Question2**: Implement the “ResNet50_layer4”
- Network Architecture
  - It is different from the original ResNet50.
  - Complete the network code by looking at the table.
  - Fill in the #blank# in the code.

You need to accurately structure the network architecture in the table!

The code may still run even if the structure differs, but points may be deducted.

Layer number	Network	Output Image size
Layer 1	7x7 conv, channel = 64, stride = 2 3x3 max pool, stride = 2	8 x 8
Layer 2	[1x1 conv, channel = 64, 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 2 [1x1 conv, channel = 64, stride = 2 3x3 conv, channel = 64, 1x1 conv, channel = 256] x 1	4 x 4
Layer 3	[1x1 conv, channel = 128, 3x3 conv, channel = 128, 1x1 conv, channel = 512] x 3 [1x1 conv, channel = 128, stride = 2 3x3 conv, channel = 128, 1x1 conv, channel = 512] x 1	2 x 2
Layer 4	[1x1 conv, channel = 256, 3x3 conv, channel = 256, 1x1 conv, channel = 1024] x 6	2 x 2
	AvgPool	1 x 1
	Fully connected layer	?

ResNet – *Question*

Train “ResNet-50” model with “CIFAR-10” datasets

- Optimize parameters with Adam optimizer and cross Entropy Loss
  - Use “ResNet-50_layer4” model that you completed
  - Get “CIFAR-10” Dataset with torchvision library
- Procedure
  - Load the trained model (which is given)
  - Train it with CPU or GPU
  - You can use a trained checkpoint parameters of 285 epochs. You will train model only 1 epoch.

```
model = ResNet50_layer4().to(device)
PATH = './resnet50_epoch285.ckpt' # test acc would be almost 80

checkpoint = torch.load(PATH, map_location=torch.device('cpu'))
model.load_state_dict(checkpoint)
```

```
# Hyper-parameters
num_epochs = 1
learning_rate = 0.001

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

CNN Implementation

- You must submit **“resnet50_skeleton.py”** along with the **report**
  (Do not modify the name of the “resnet50_skeleton.py”)
- Include a **1 page** report in **CVPR** format that describes your code, results, and discussions.
- The report should be written in **English**.

CVPR format : https://cvpr.thecvf.com/Conferences/2025/AuthorGuidelines
→ Download CVPR 2025 Author Kit

CNN Implementation

Please do NOT copy your friends' and internet sources.

Please start your assignment EARLY.

"Late submissions will not be accepted"