AlexNet + VGG

서울과학기술대학교 국방인공지능응용학과
이찬호

Part 1

AlexNet –Architecture

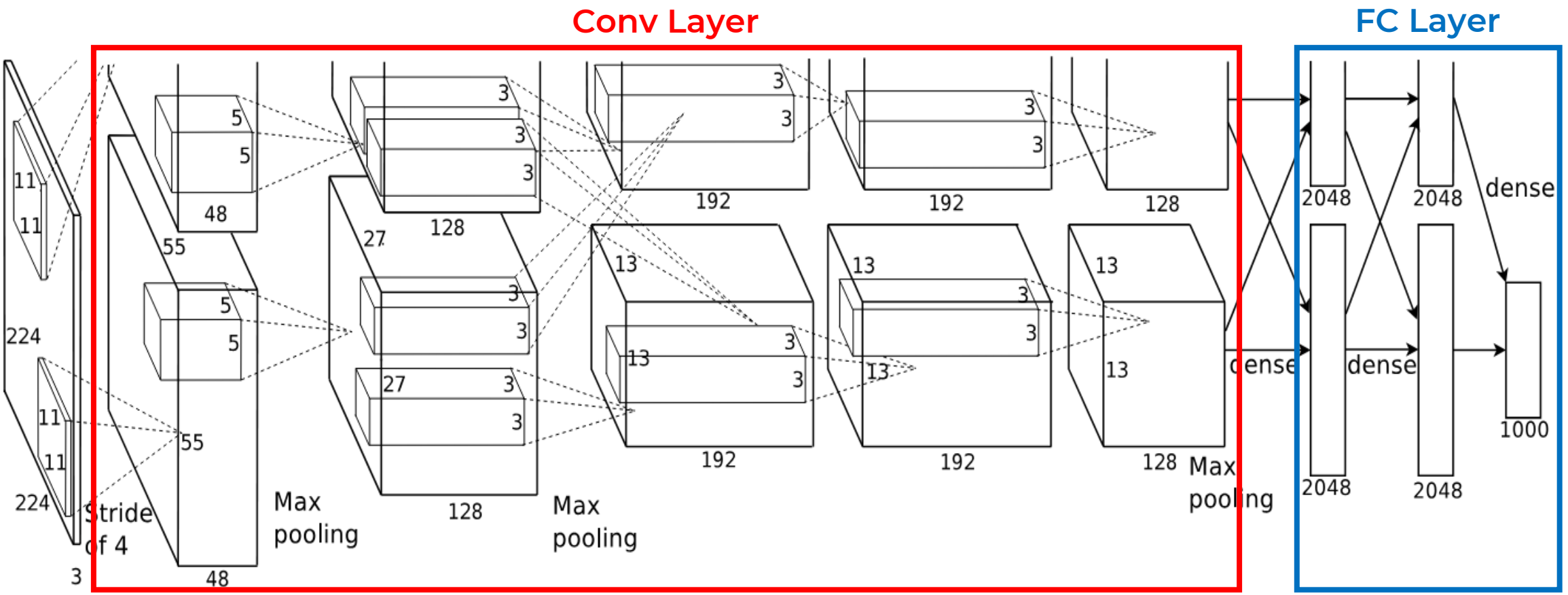


Fig 1. AlexNet Architecture

Part 1

AlexNet –Architecture

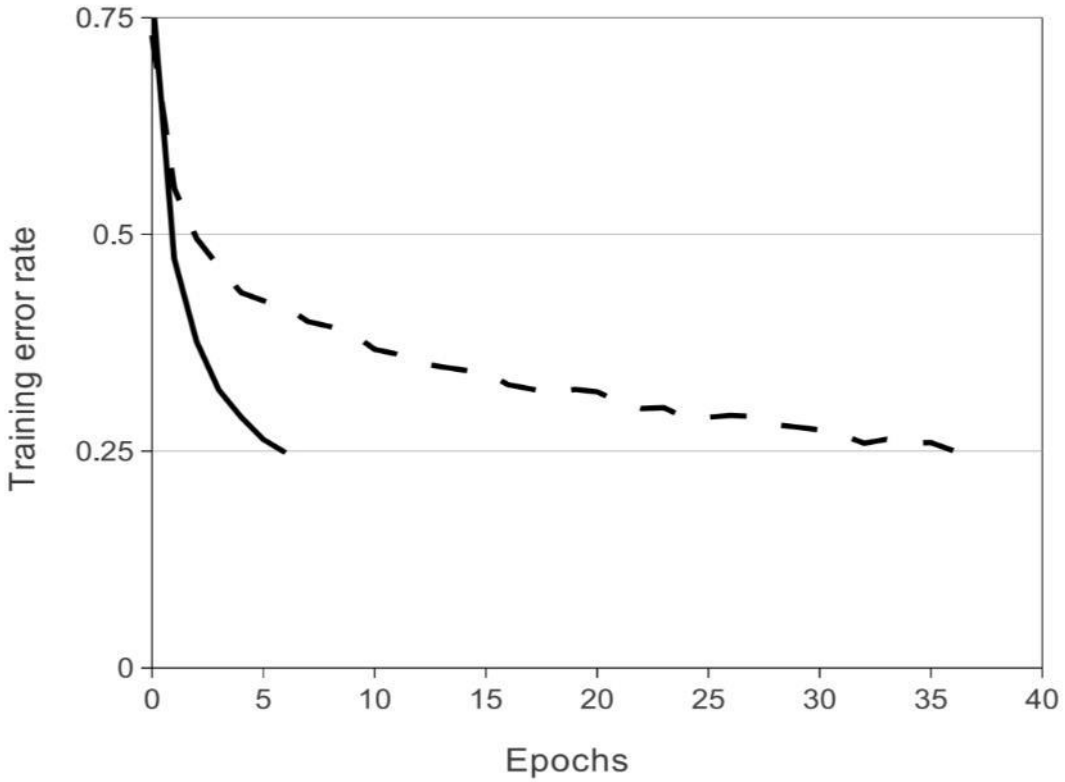


Fig 2. ReLU(실선) VS tanh(점선)

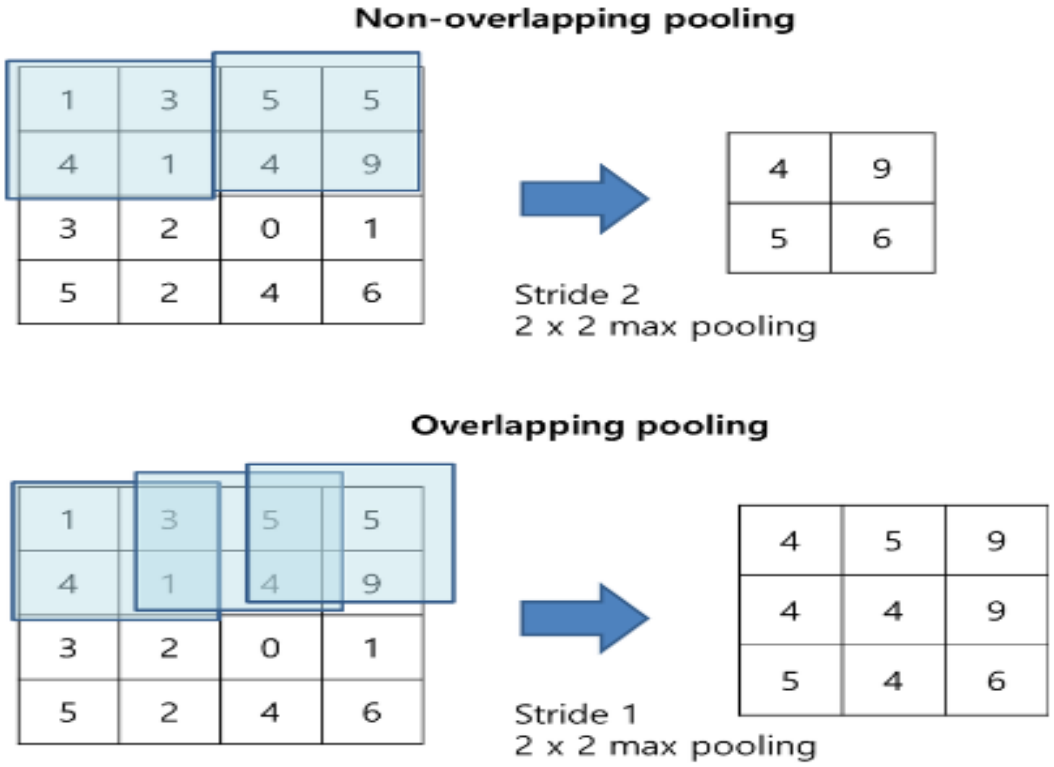


Fig 3. Overlapping Pooling

Part 1

AlexNet –Architecture

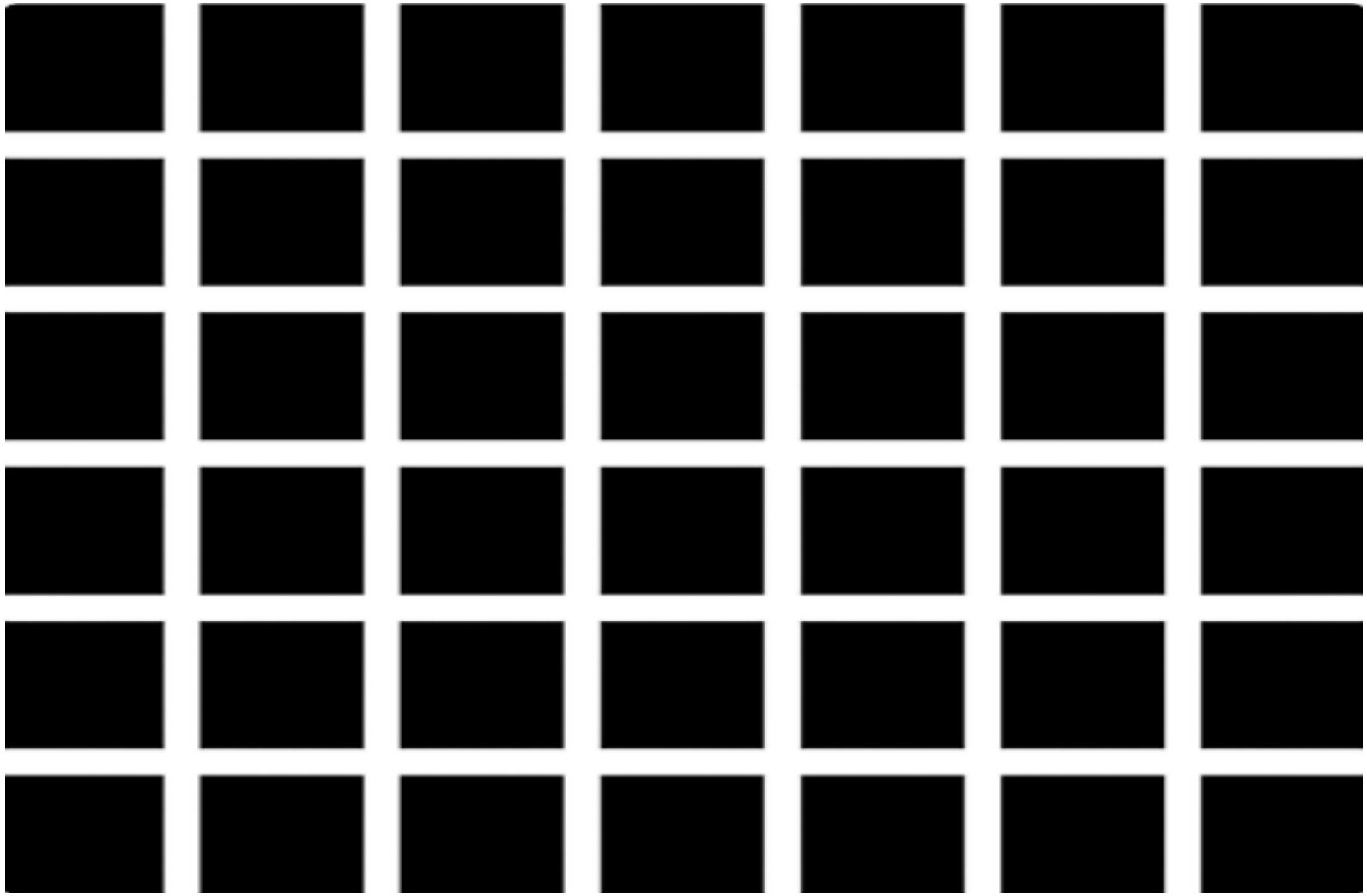


Fig 4. Local Response Normalization

Part 1

AlexNet – Reducing Overfitting

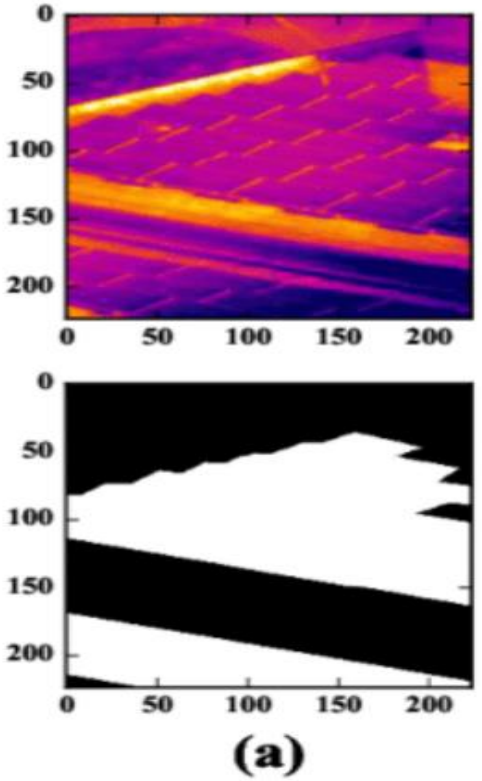


Fig 5. Random Crop

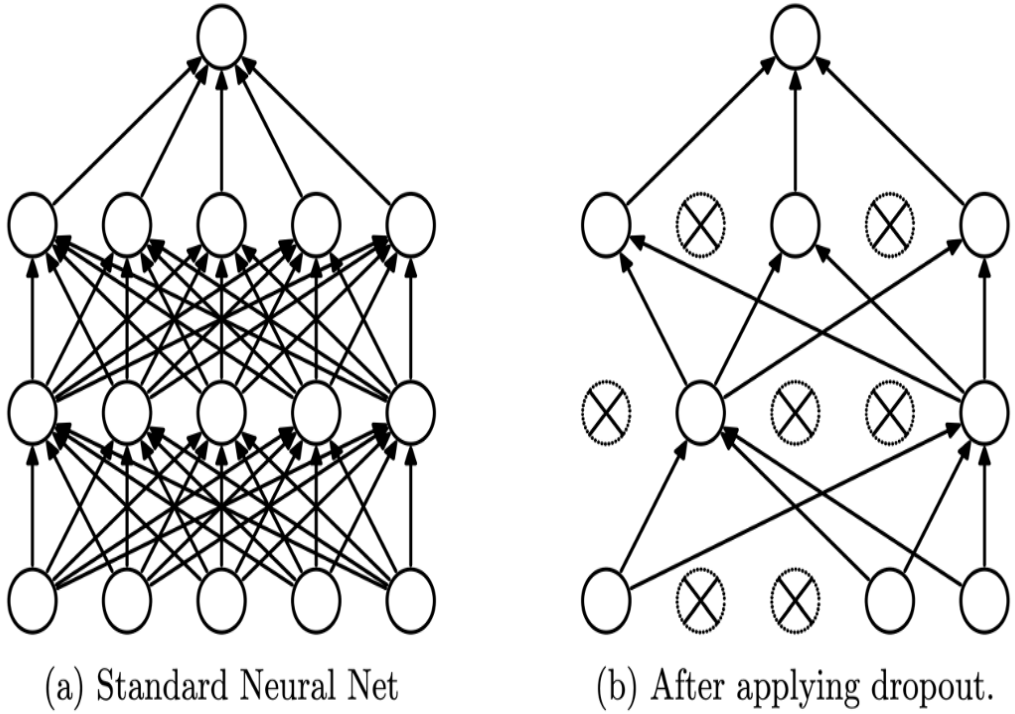


Fig 6. DropOut

Part 1

AlexNet –Code

```
#1st Layer
layer = Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu')(input_tensor)
layer = BatchNormalization()(layer)
layer = MaxPooling2D(pool_size=(3,3), strides=(2,2))(layer)

#2nd Layer
layer = Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding='same')(layer)
layer = BatchNormalization()(layer)
layer = MaxPooling2D(pool_size=(3,3), strides=(2,2))(layer)

#3rd Layer
layer = Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same')(layer)
layer = BatchNormalization()(layer)

layer = Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same')(layer)
layer = BatchNormalization()(layer)

layer = Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same')(layer)
layer = BatchNormalization()(layer)
layer = MaxPooling2D(pool_size=(3,3), strides=(2,2))(layer)

layer = Flatten()(layer)

# FC Layer
layer = Dense(units=4096, activation='relu')(layer)
layer = Dropout(0.5)(layer)

layer = Dense(units=4096, activation='relu')(layer)
layer = Dropout(0.5)(layer)

output = Dense(units=1000, activation='softmax')(layer)
```

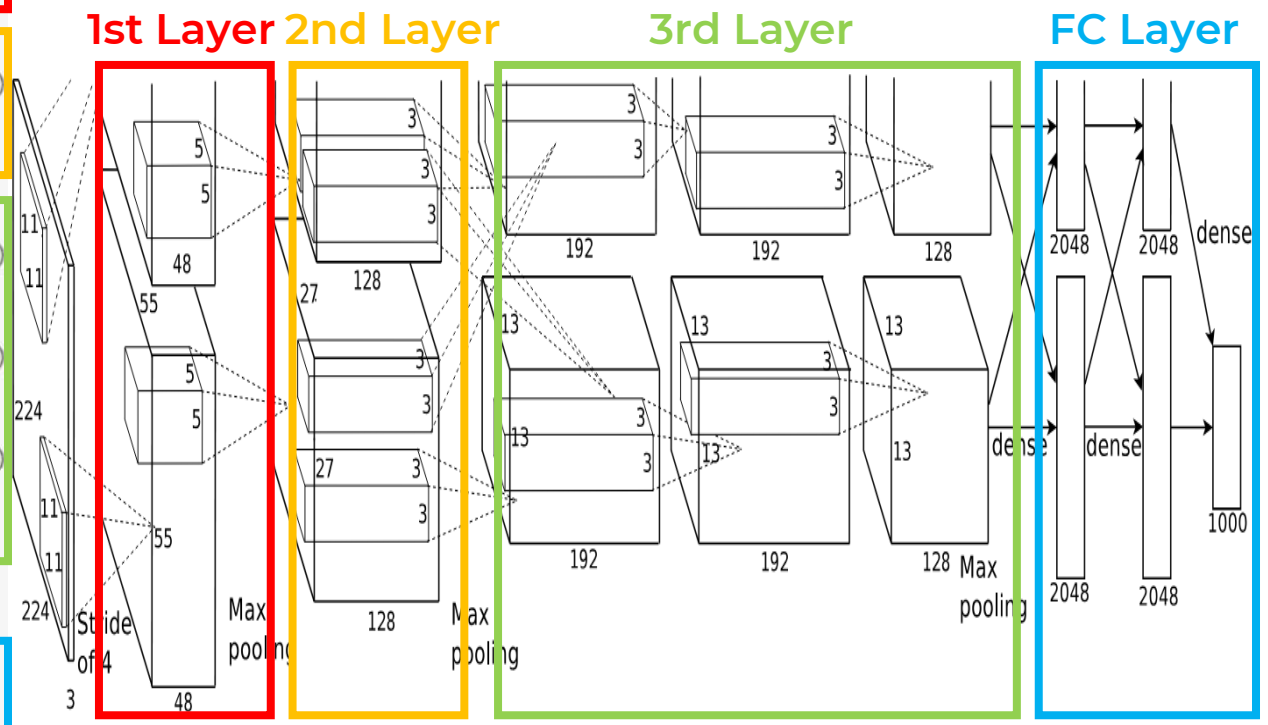


Fig 7. AlexNet 구현 코드

Part 2

VGG –Architecture

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					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Fig 8. VGG Architecture

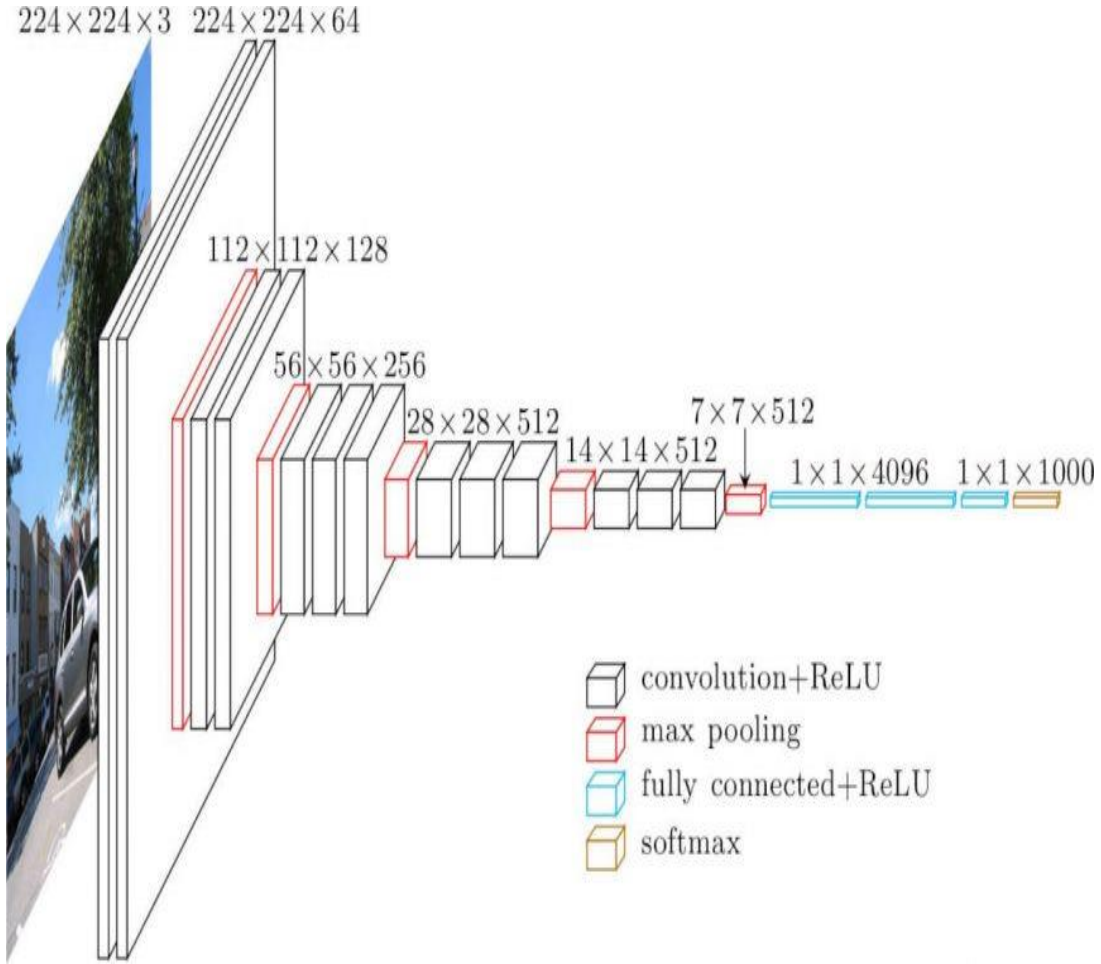


Fig 9. VGG-16 Architecture

Part 2

VGG-16 –Code

```
model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = "same" ,activation="relu", input_shape = (224, 224, 3)))
model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = "same", activation = "relu"))
model.add(MaxPooling2D(2))
```

```
model.add(Conv2D(filters = 128, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 128, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(MaxPooling2D(2))
```

```
model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(MaxPooling2D(2))
```

```
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(MaxPooling2D(2))
```

```
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
model.add(MaxPooling2D(2))
```

```
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='softmax'))
```

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	1st Layer
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	2nd Layer
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 3rd Layer
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 4th Layer
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 5th Layer
maxpool					
FC-4096					
FC-4096					FC Layer
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Fig 10. VGG-16 구현 코드

Part 3

AlexNet + VGG –Code

```
#1st Layer
layer = Conv2D(64, (3,3),strides=(2,2), activation='relu')(input_tensor)
layer = MaxPooling2D(pool_size=(2,2), strides=(2,2))(layer)
layer = BatchNormalization()(layer)

#2nd Layer
layer = Conv2D(128, (3,3), activation='relu', padding='same')(layer)
layer = MaxPooling2D(pool_size=(2,2), strides=(2,2))(layer)
layer = BatchNormalization()(layer)

#3rd Layer
layer = Conv2D(256, (3,3), activation='relu', padding='same')(layer)
layer = Conv2D(256, (3,3), activation='relu', padding='same')(layer)
layer = MaxPooling2D(pool_size=(2,2), strides=(2,2))(layer)
layer = BatchNormalization()(layer)

#4th Layer
layer = Conv2D(512, (3,3), activation='relu', padding='same')(layer)
layer = Conv2D(512, (3,3), activation='relu', padding='same')(layer)
layer = MaxPooling2D(pool_size=(2,2), strides=(2,2))(layer)
layer = BatchNormalization()(layer)

#5th Layer
layer = Conv2D(512, (3,3), activation='relu', padding='same')(layer)
layer = Conv2D(512, (3,3), activation='relu', padding='same')(layer)
layer = MaxPooling2D(pool_size=(2,2), strides=(2,2))(layer)
layer = BatchNormalization()(layer)

layer = Flatten()(layer)

# FC Layer
layer = Dense(units=4096, activation='relu')(layer)
layer = Dropout(0.5)(layer)
layer = Dense(units=4096, activation='relu')(layer)
layer = Dropout(0.5)(layer)
output = Dense(units=100, activation='softmax')(layer)
```

Fig 11. AlexNet + VGG 구현 코드

Part 4

AlexNet + VGG –Data Set

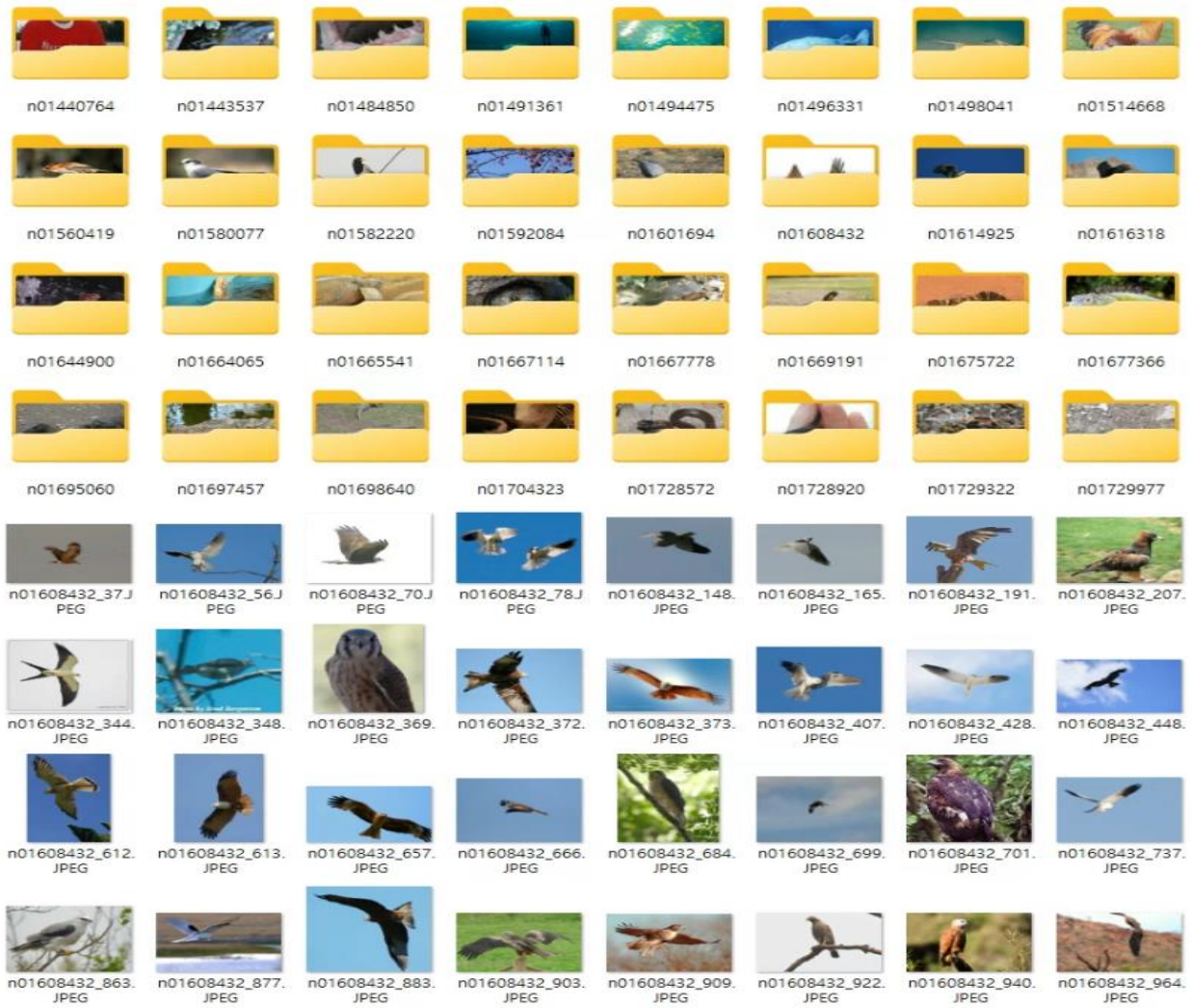
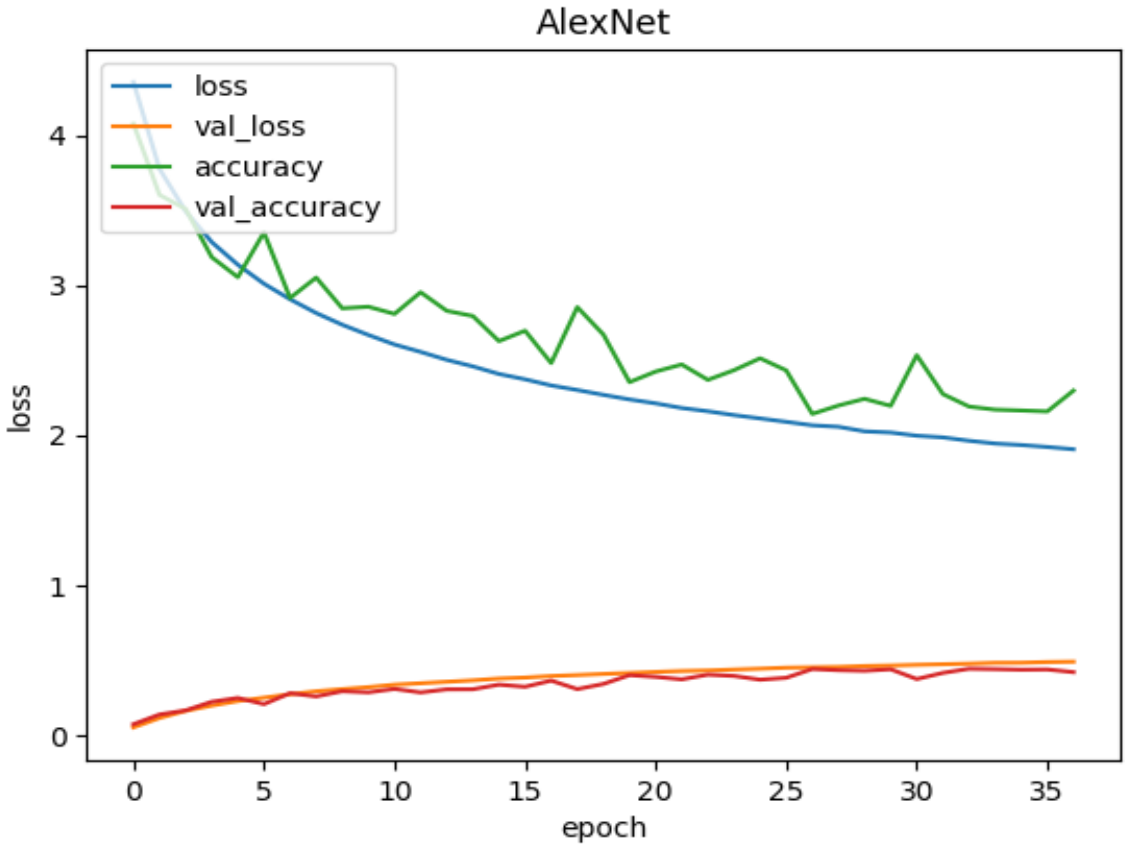


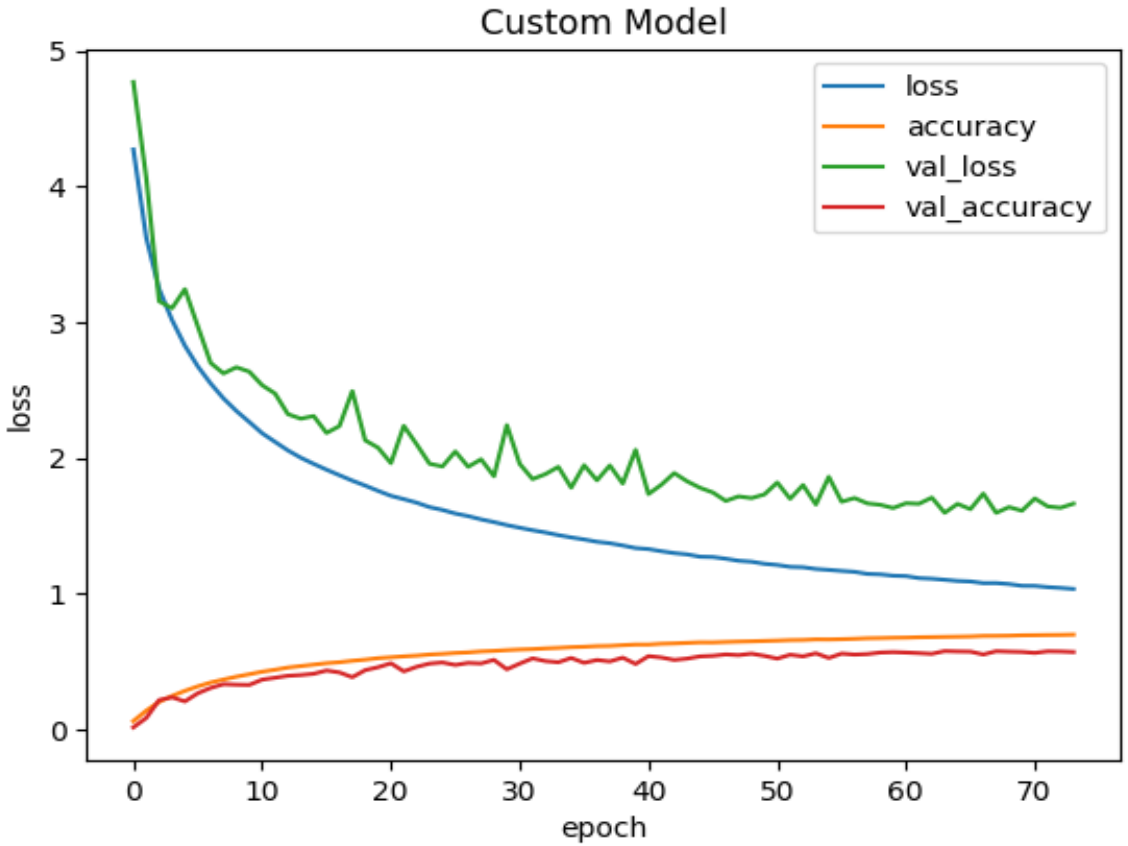
Fig 12. Train Class 일부 **(**위**) & Img** 중 일부 **(**아래**)**

Part 5

AlexNet + VGG –Result



Params	Loss	Val_Loss	ACC	Val_ACC
58,696,548	2.0653	2.1413	0.4529	0.4432



Params	Loss	Val_Loss	ACC	Val_ACC
45,295,844	1.1039	1.5973	0.6834	0.5788

THANK YOU FOR YOUR ATTENTION