Conclusion

이찬호
국방인공지능응용학과

AlexNet – Architecture

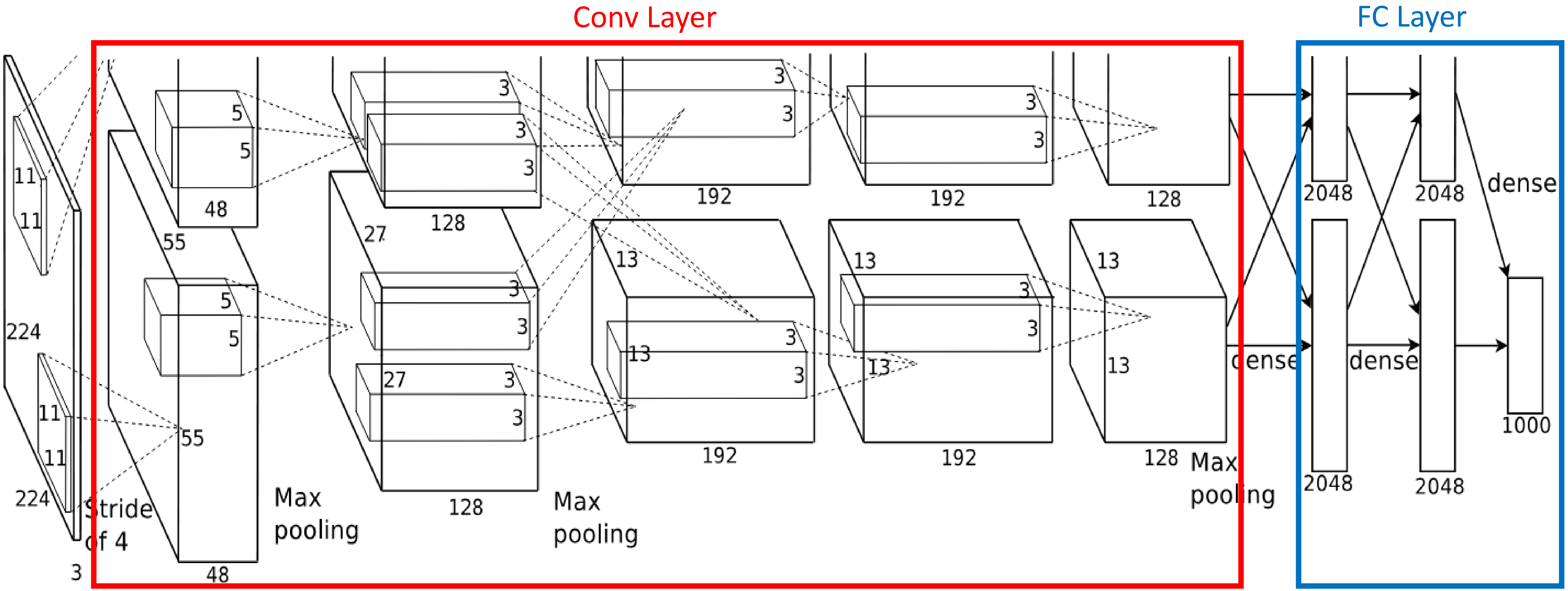


Fig 1. AlexNet Architecture

AlexNet – Architecture

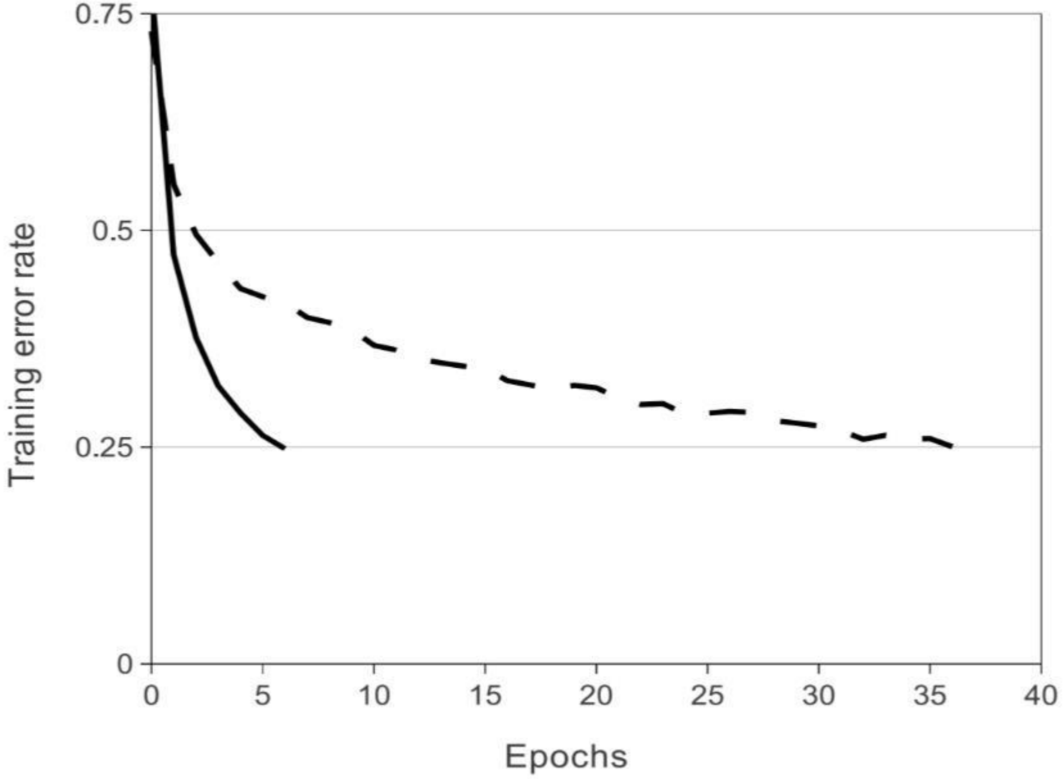


Fig 2. ReLU(실선) VS tanh(점선)

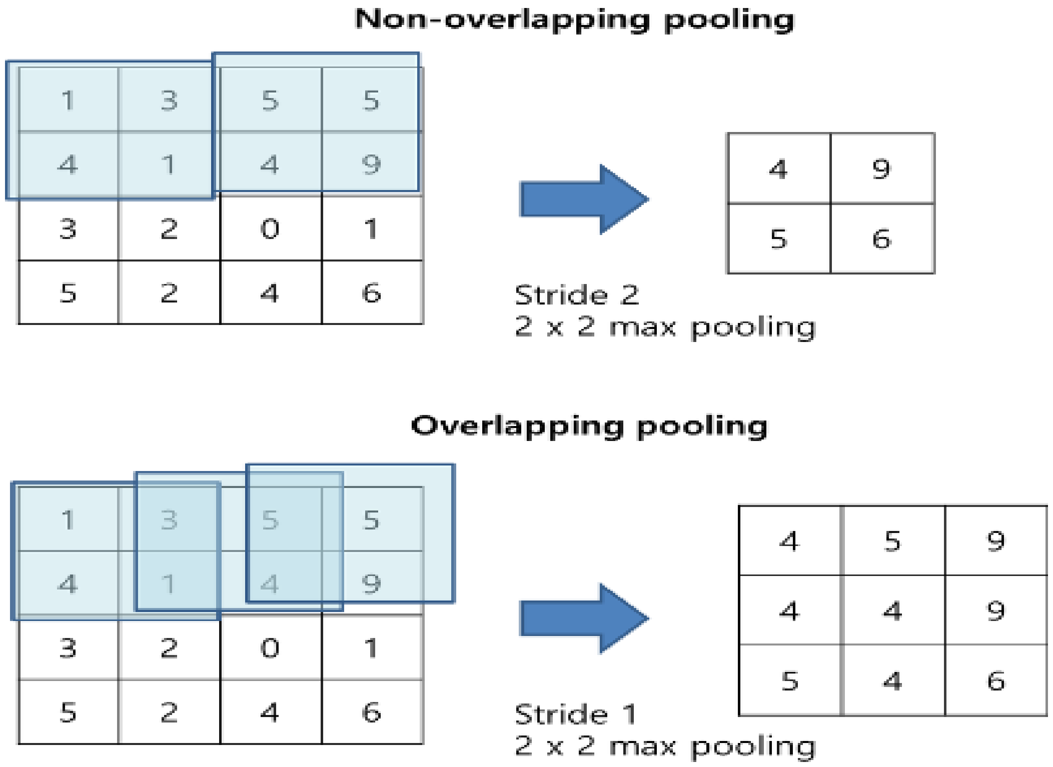


Fig 3. Overlapping Pooling

AlexNet – Architecture

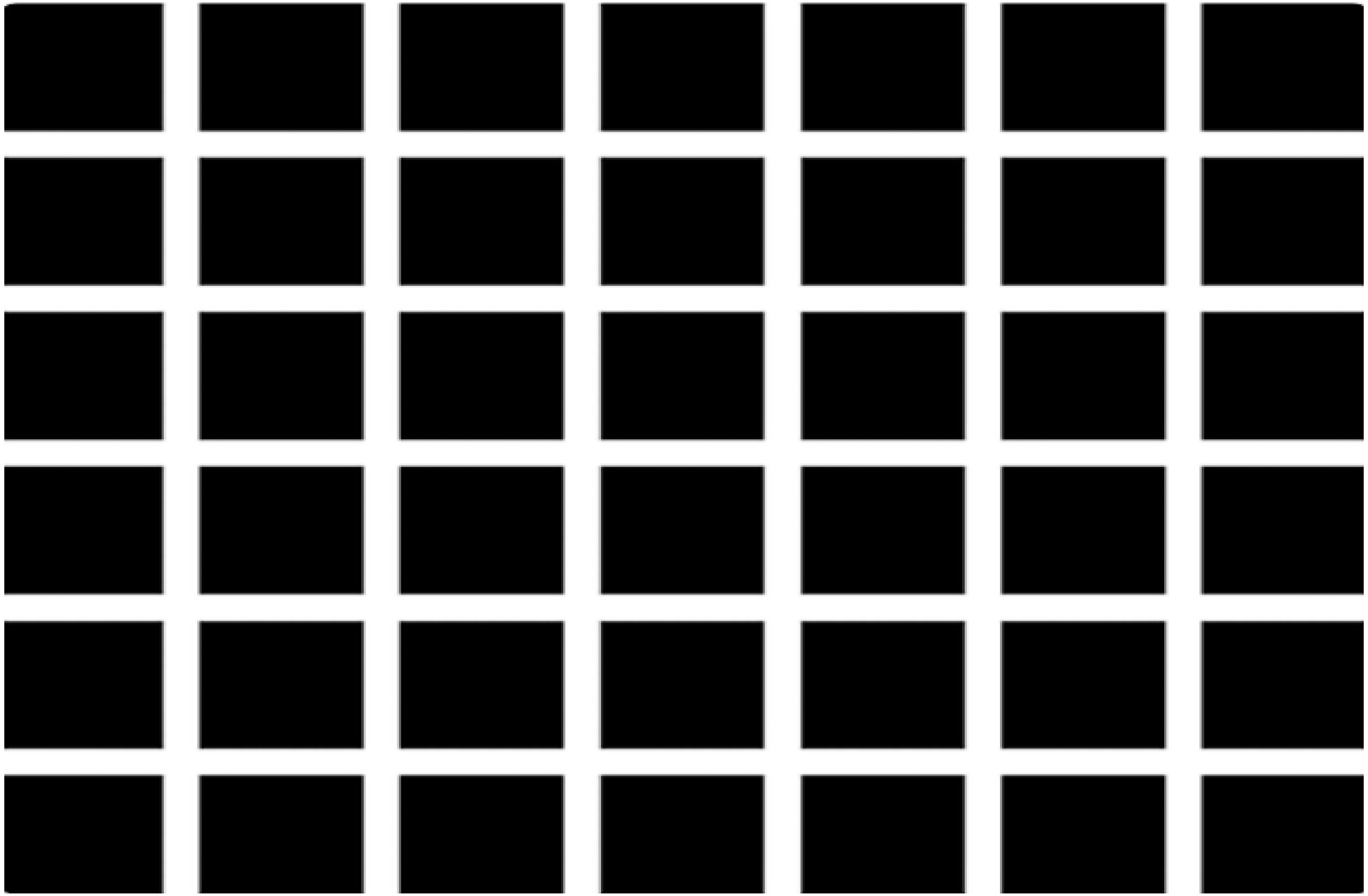


Fig 4. Local Response Normalization

AlexNet – Reducing Overfitting

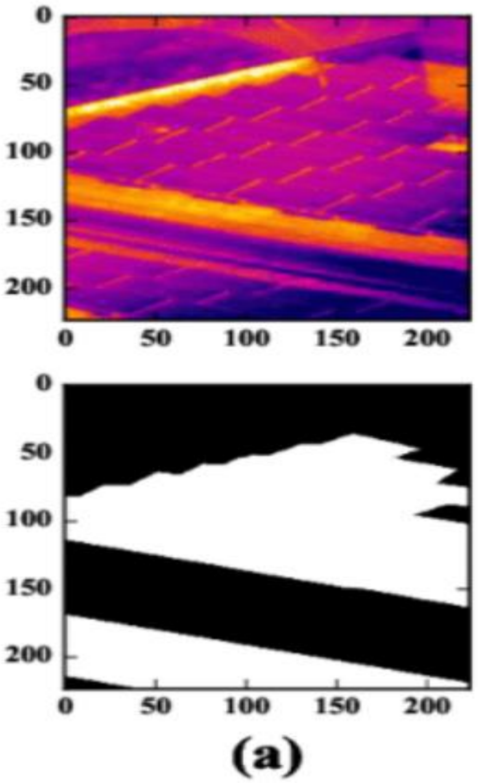


Fig 5. Random Crop

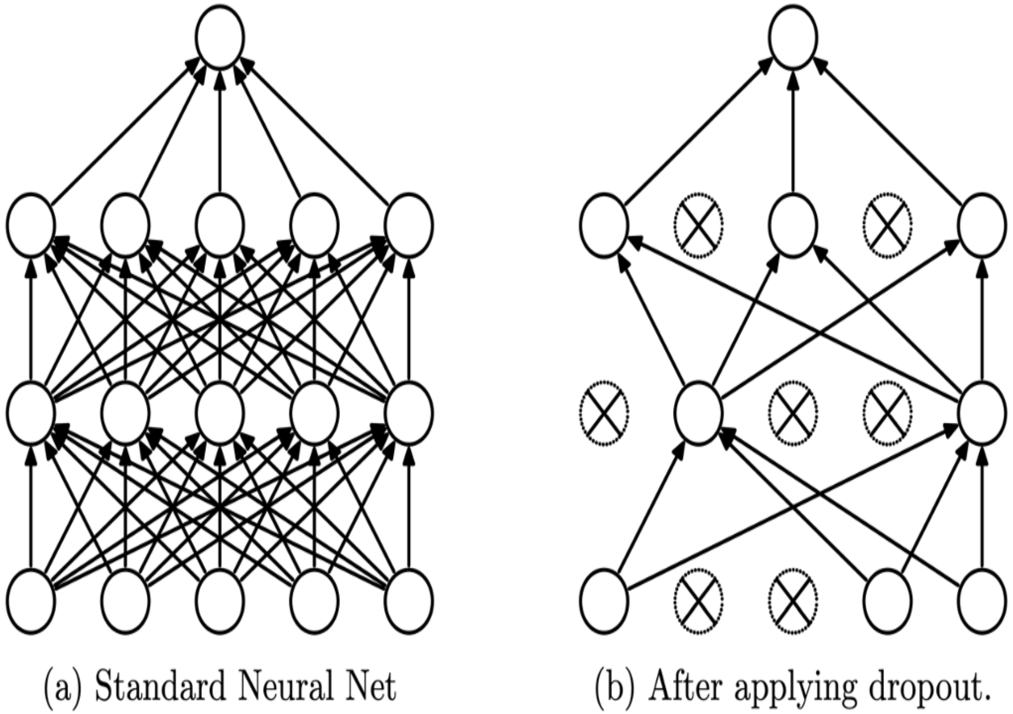
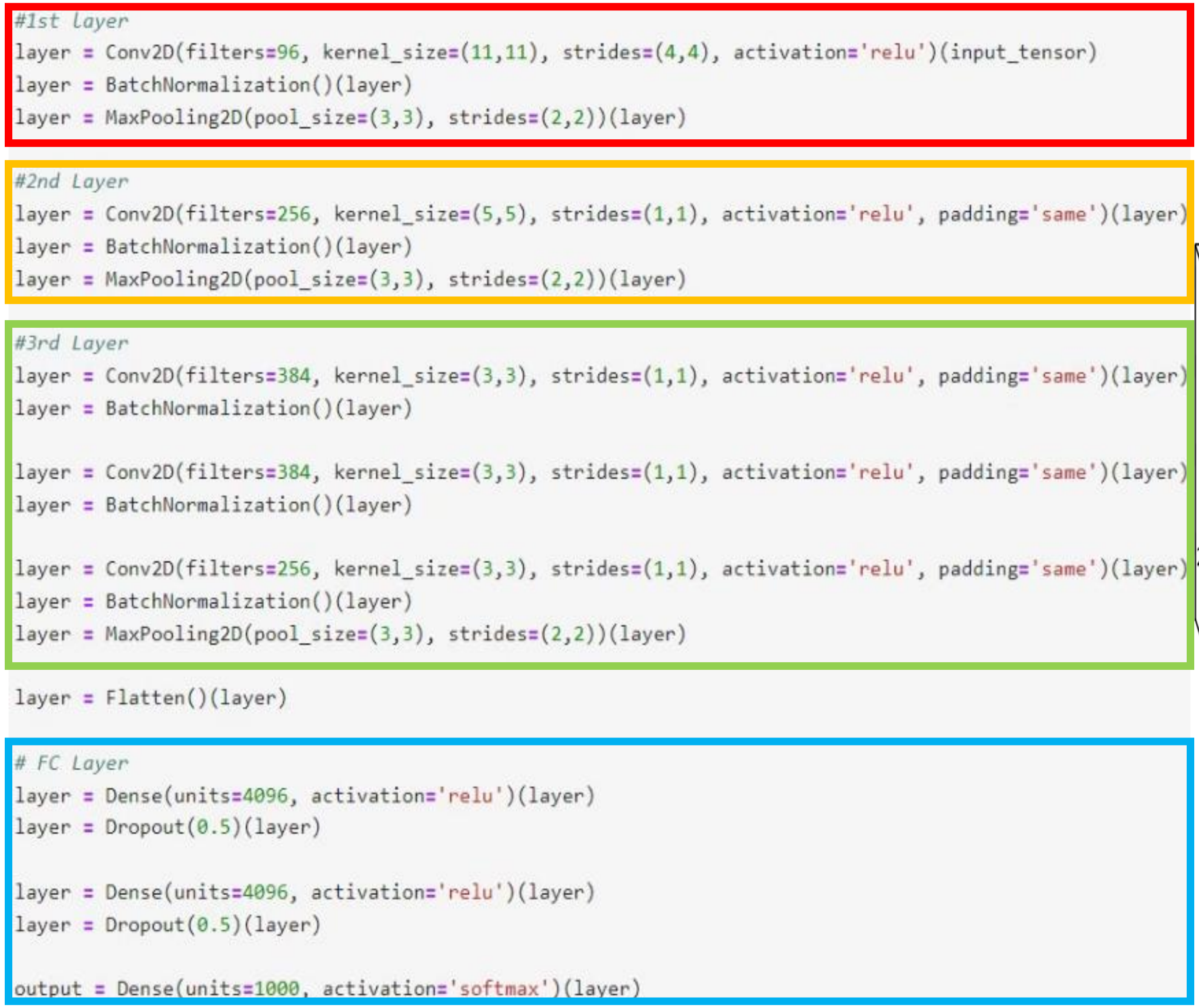


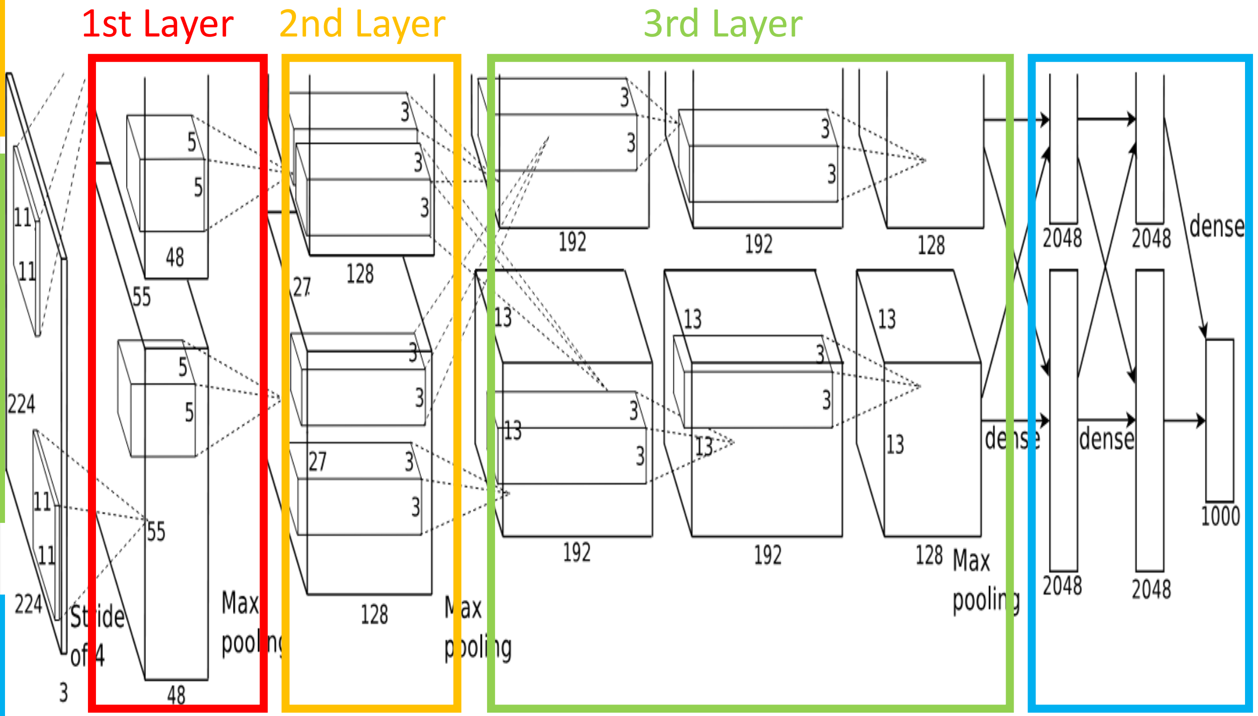
Fig 6. DropOut

AlexNet – Code

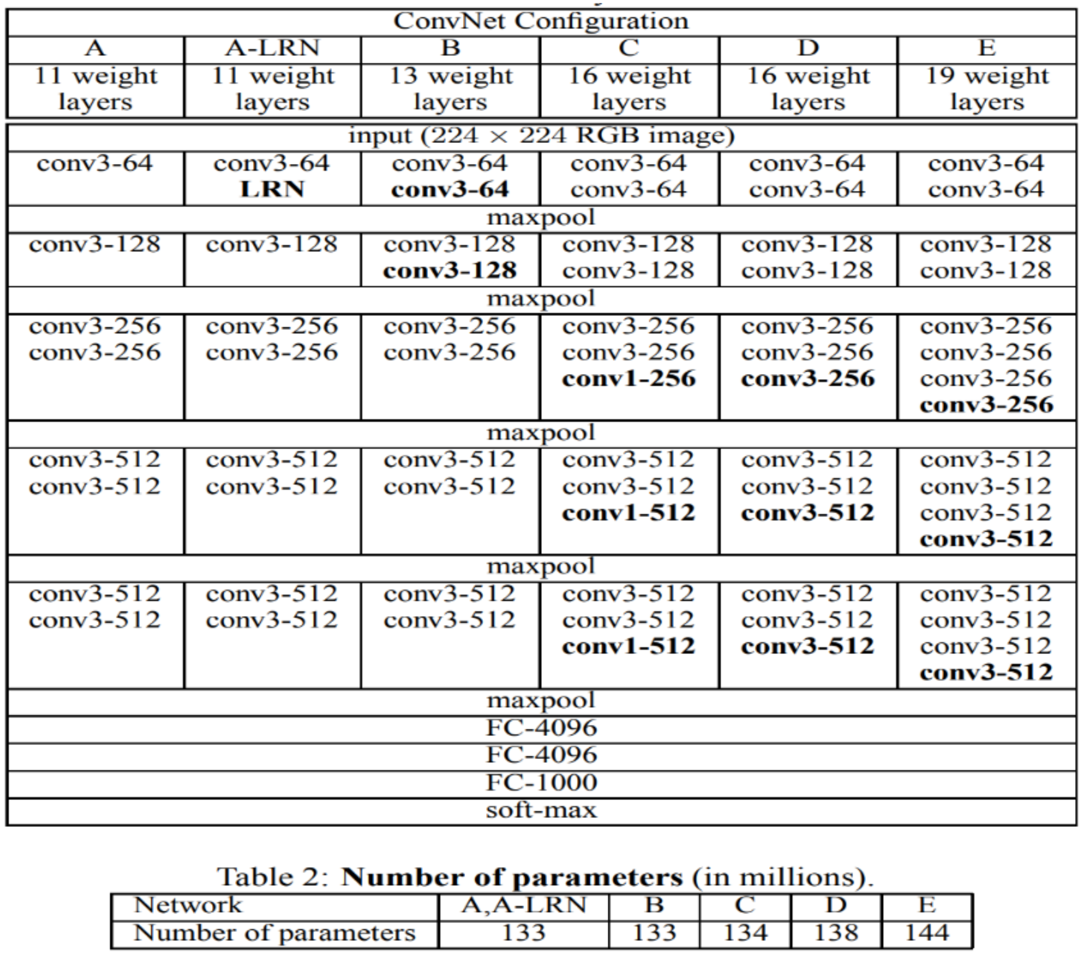


```
#1st layer
layer = Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu')(input_tensor)
layer = BatchNormalization()(layer)
layer = MaxPooling2D(pool_size=(3,3), strides=(2,2))(layer)

#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



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

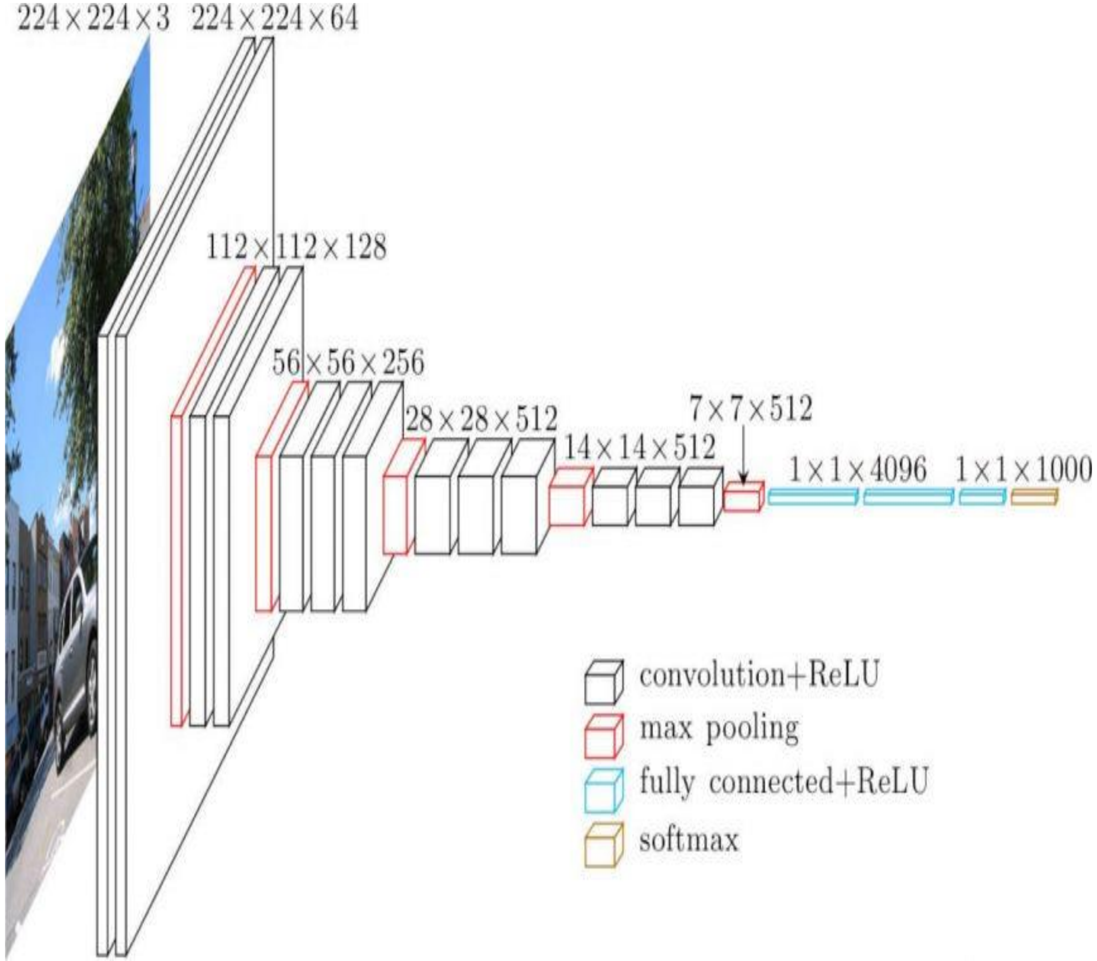
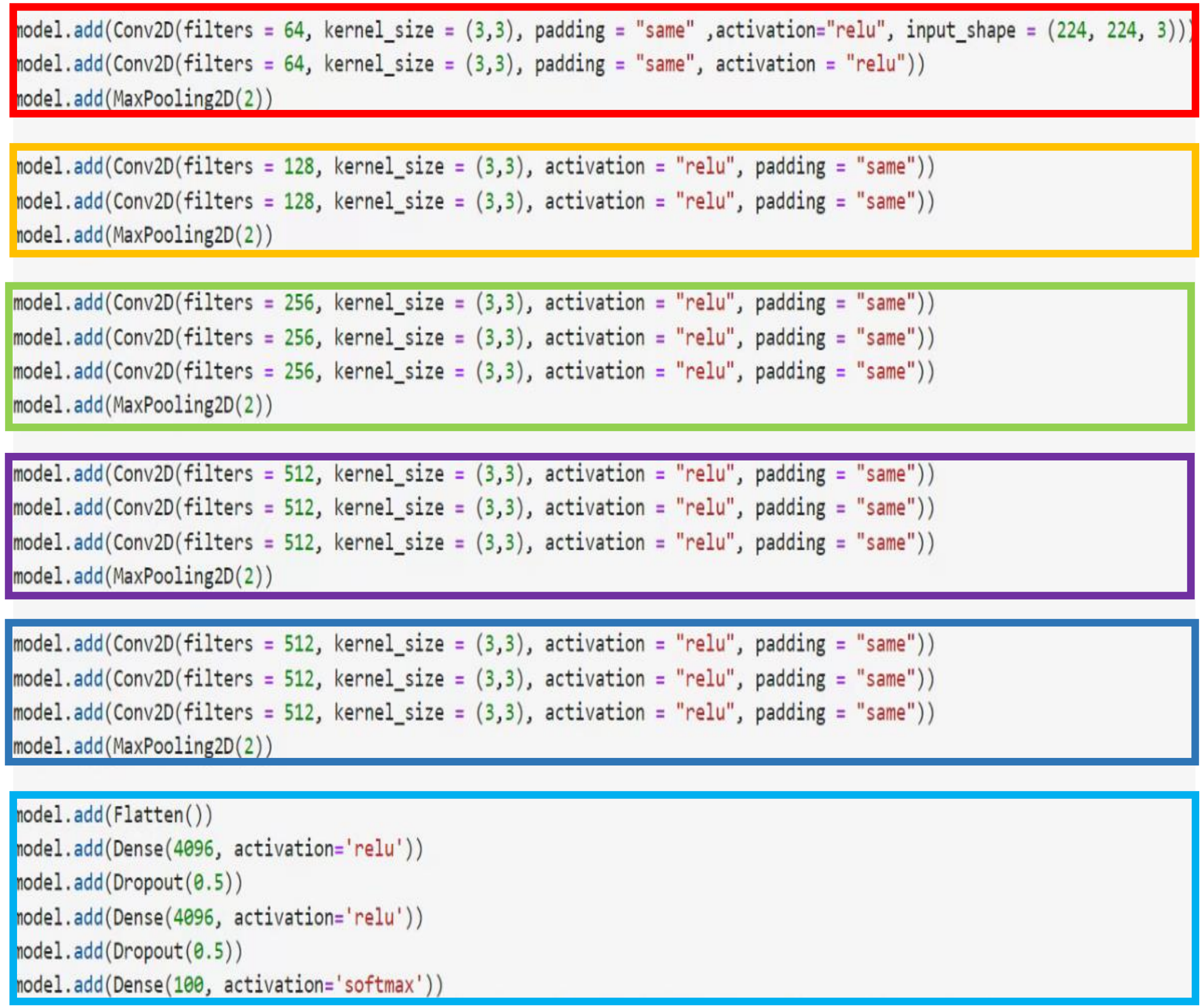


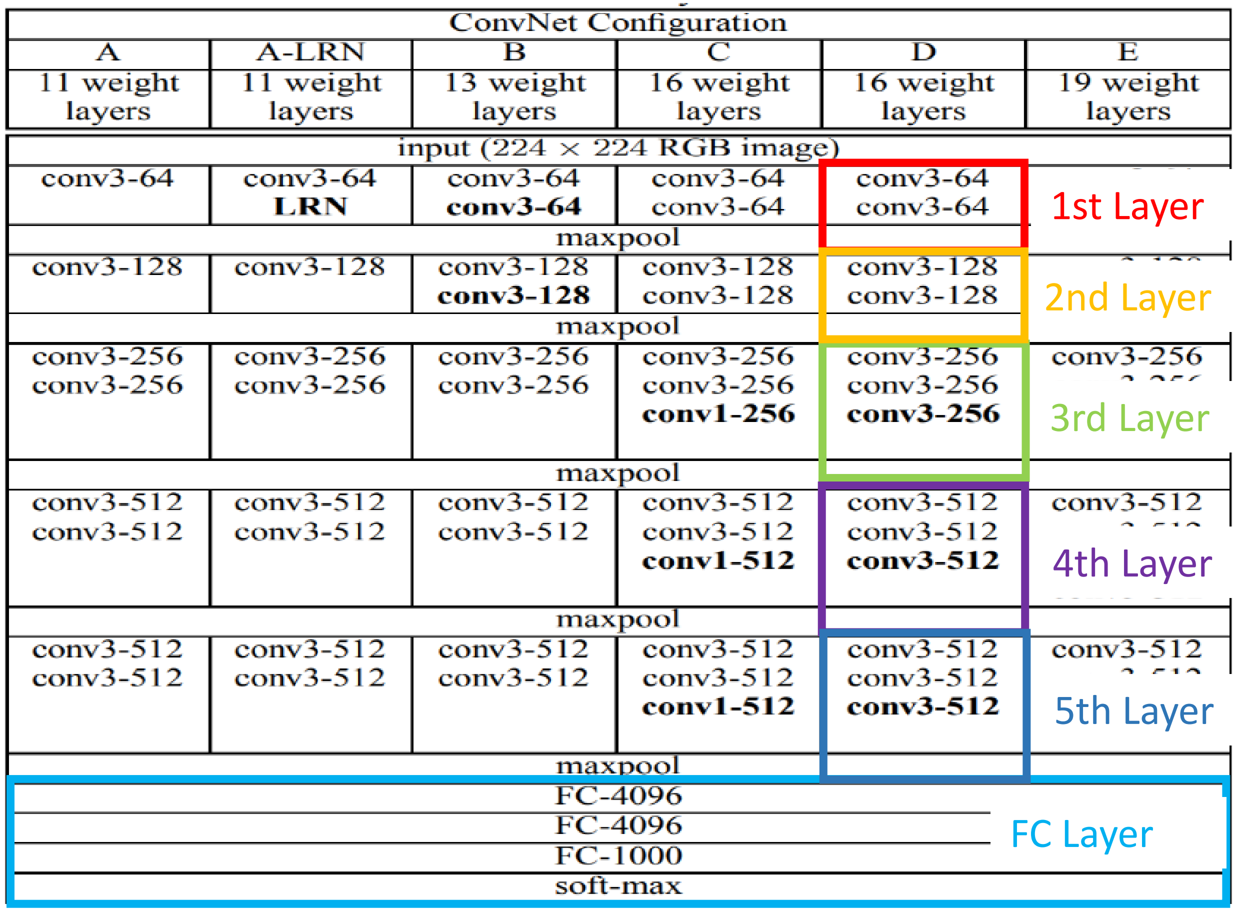
Fig 9. VGG-16 Architecture

VGG – 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'))
```

Fig 10. VGG-16 구현 코드



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	
maxpool					
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
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
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
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

1st Layer
2nd Layer
3rd Layer
4th Layer
5th Layer
FC Layer

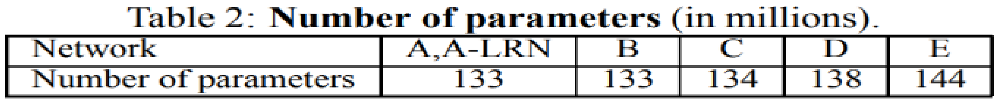
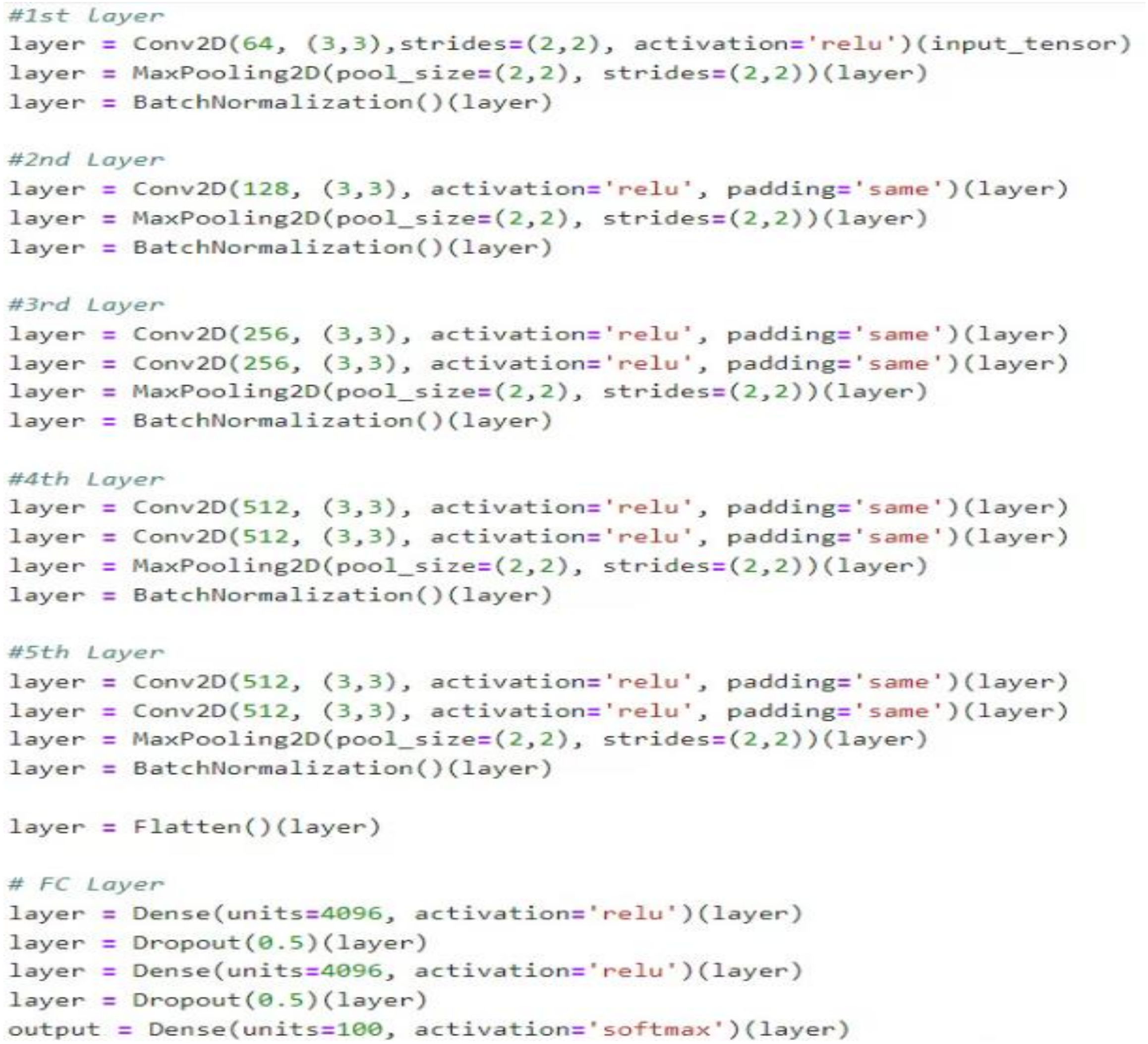


Table 2: **Number of parameters** (in millions).

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

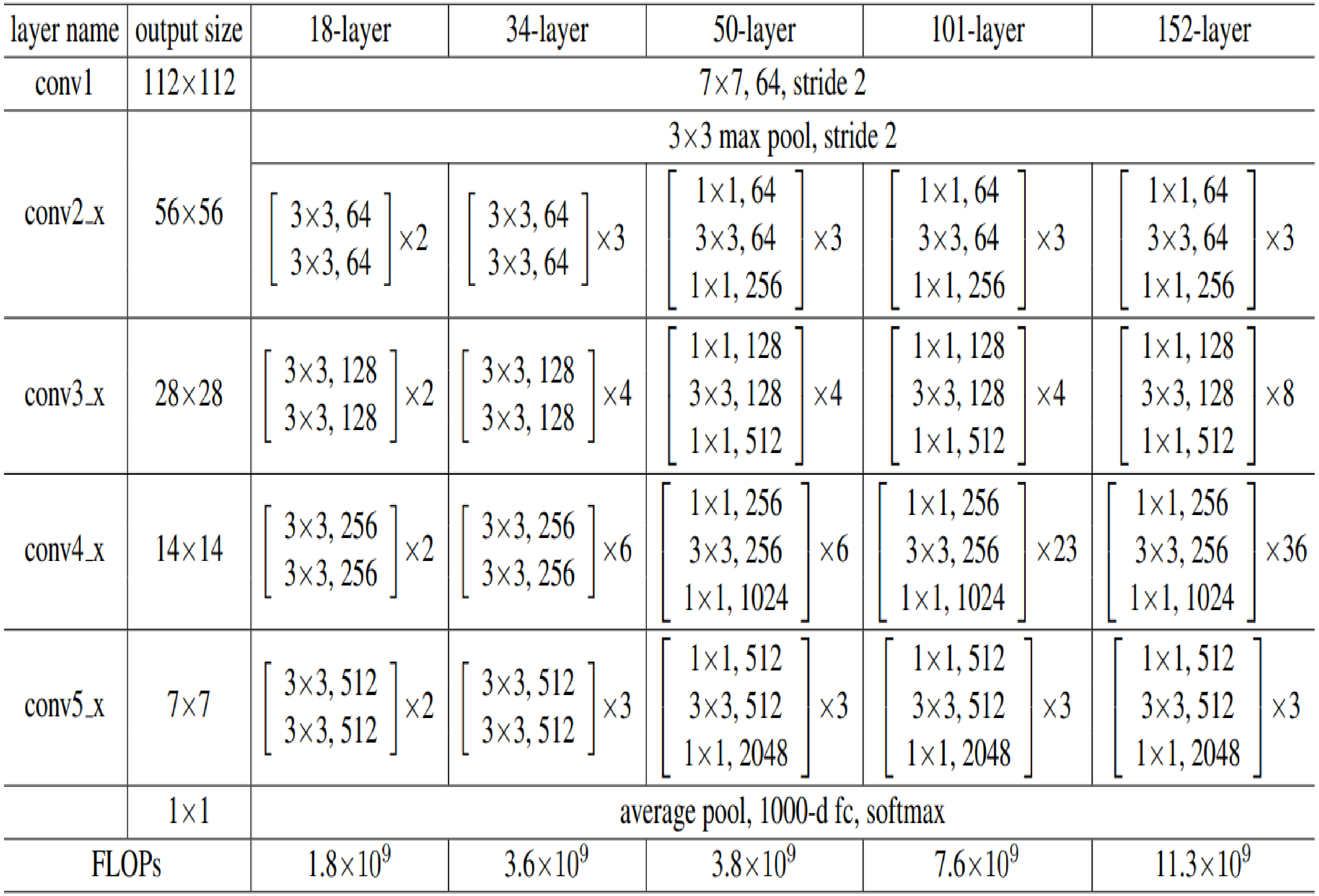
AlexNet + VGG – Code



```
#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

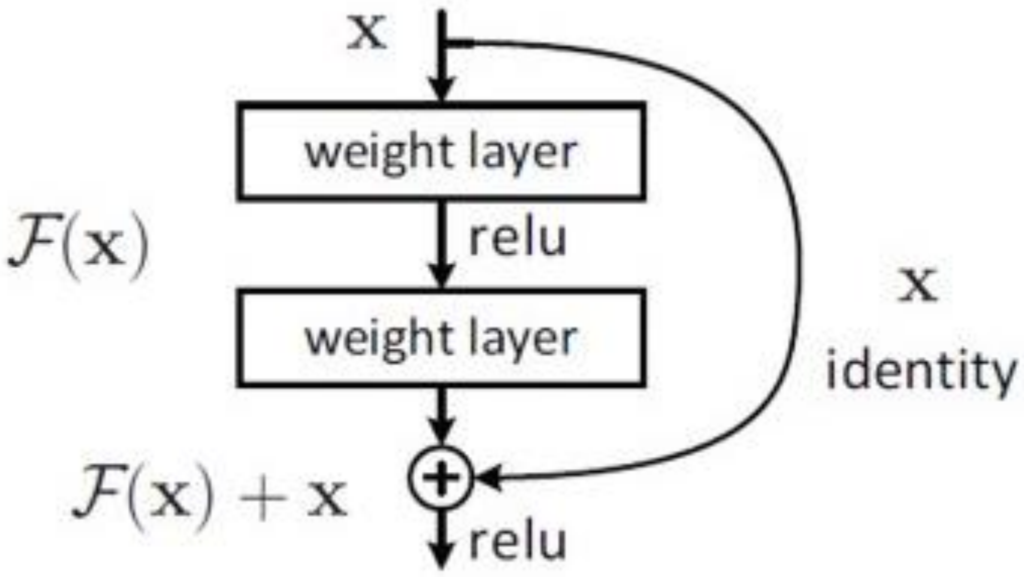
ResNet – Architecture



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

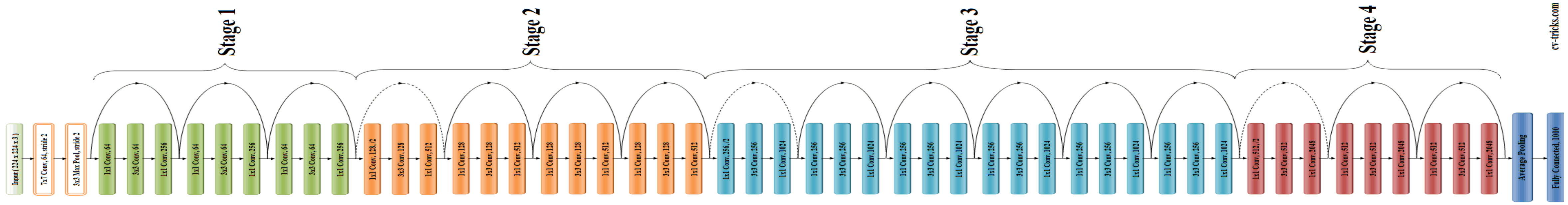
<FIG 12. ResNet Architecture>

ResNet – Residual Learning Framework



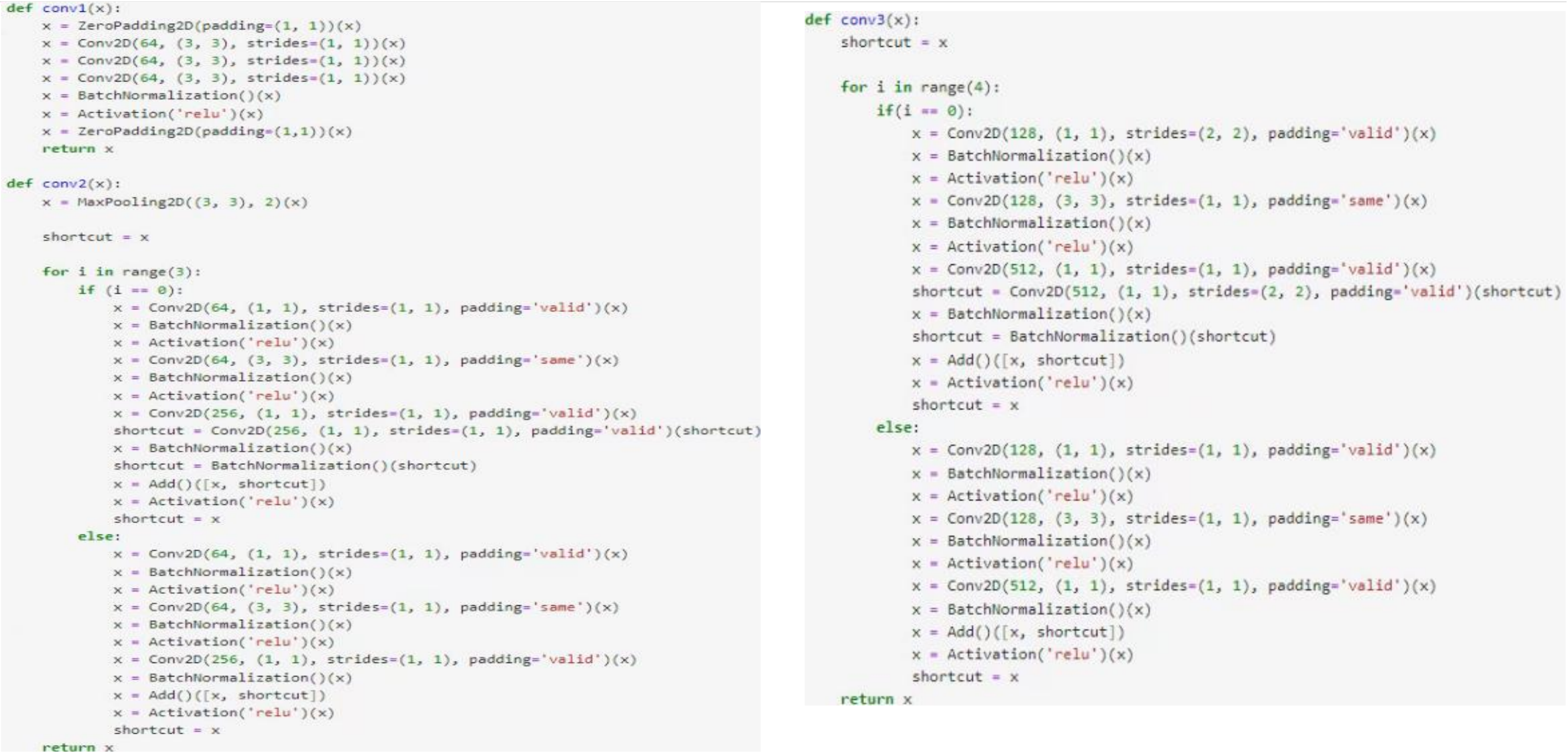
x
weight layer
$\mathcal{F}(\mathrm{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathrm{x}) + \mathrm{x}$
relu

<FIG 13. Residual Learning Framework>



<FIG 14. ResNet-50 Architecture>

AlexNet + VGG + ResNet – Code

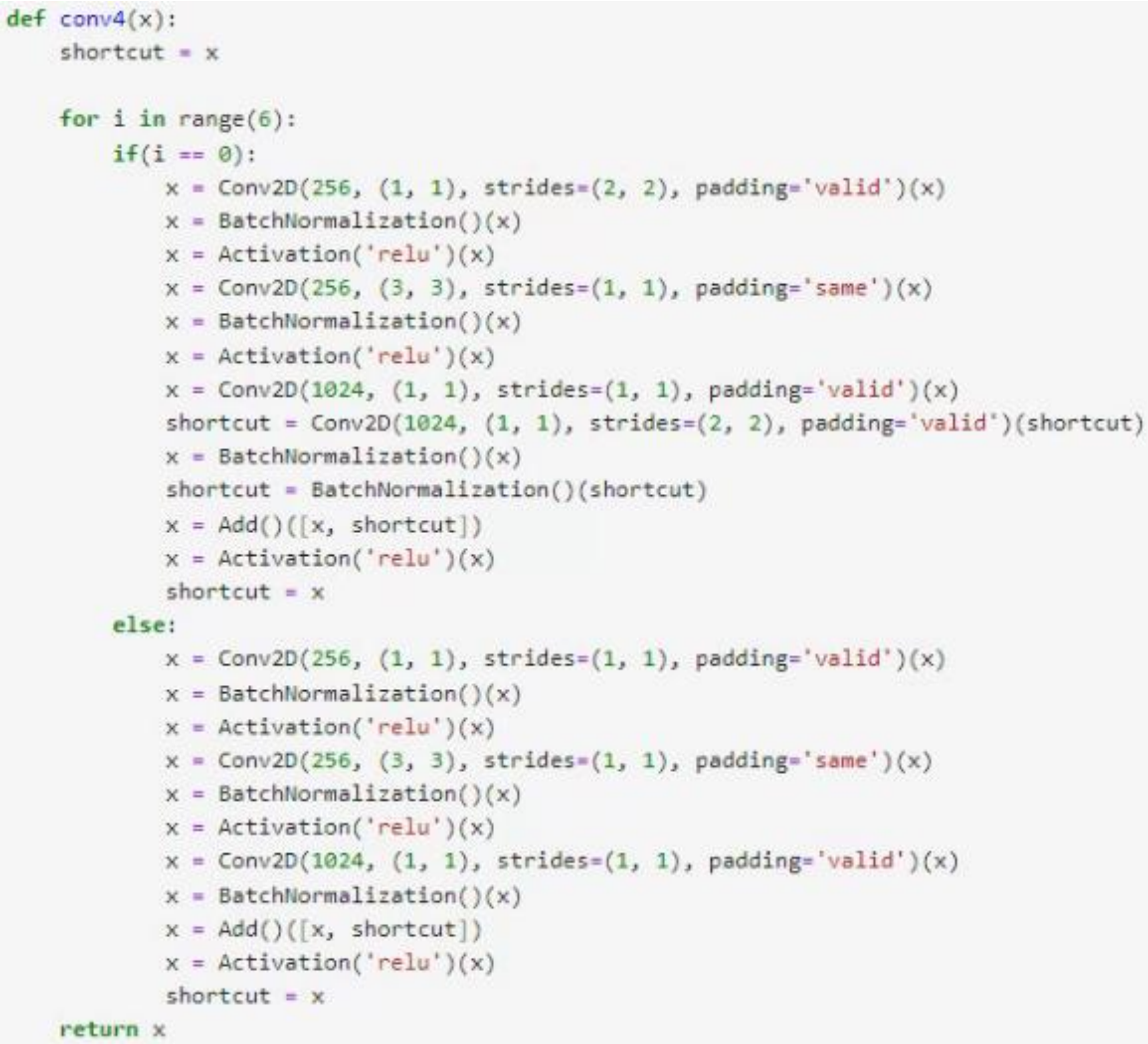


```python
def conv1(x):
    x = ZeroPadding2D(padding=(1, 1))(x)
    x = Conv2D(64, (3, 3), strides=(1, 1))(x)
    x = Conv2D(64, (3, 3), strides=(1, 1))(x)
    x = Conv2D(64, (3, 3), strides=(1, 1))(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = ZeroPadding2D(padding=(1,1))(x)
    return x

def conv2(x):
    x = MaxPooling2D((3, 3), 2)(x)

    shortcut = x

    for i in range(3):
        if (i == 0):
            x = Conv2D(64, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(64, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(64, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(64, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

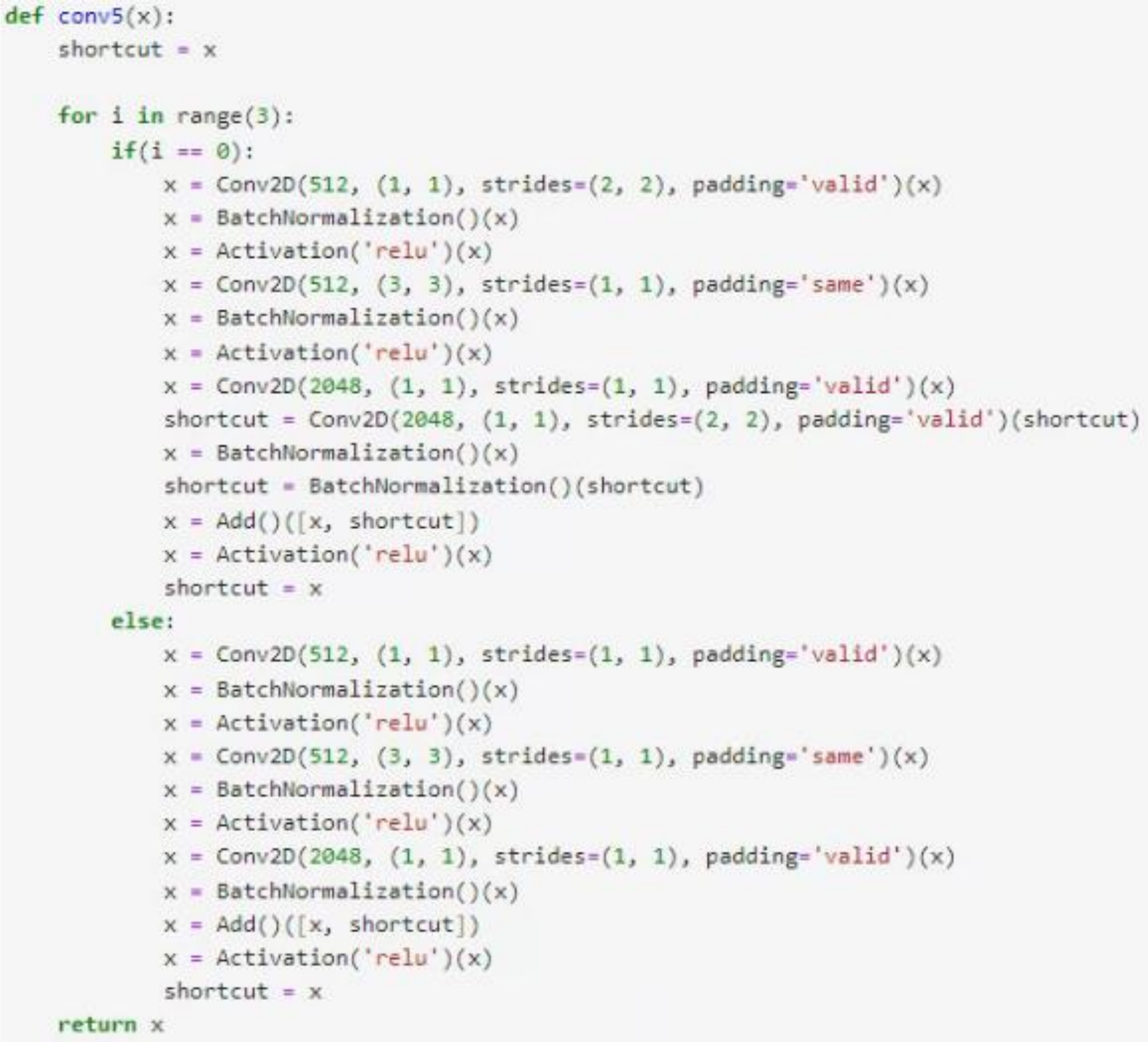
```python
def conv3(x):
    shortcut = x

    for i in range(4):
        if(i == 0):
            x = Conv2D(128, (1, 1), strides=(2, 2), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(128, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(512, (1, 1), strides=(2, 2), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(128, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(128, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

<FIG 15. Custom + ResNet Code>

AlexNet + VGG + ResNet – Code



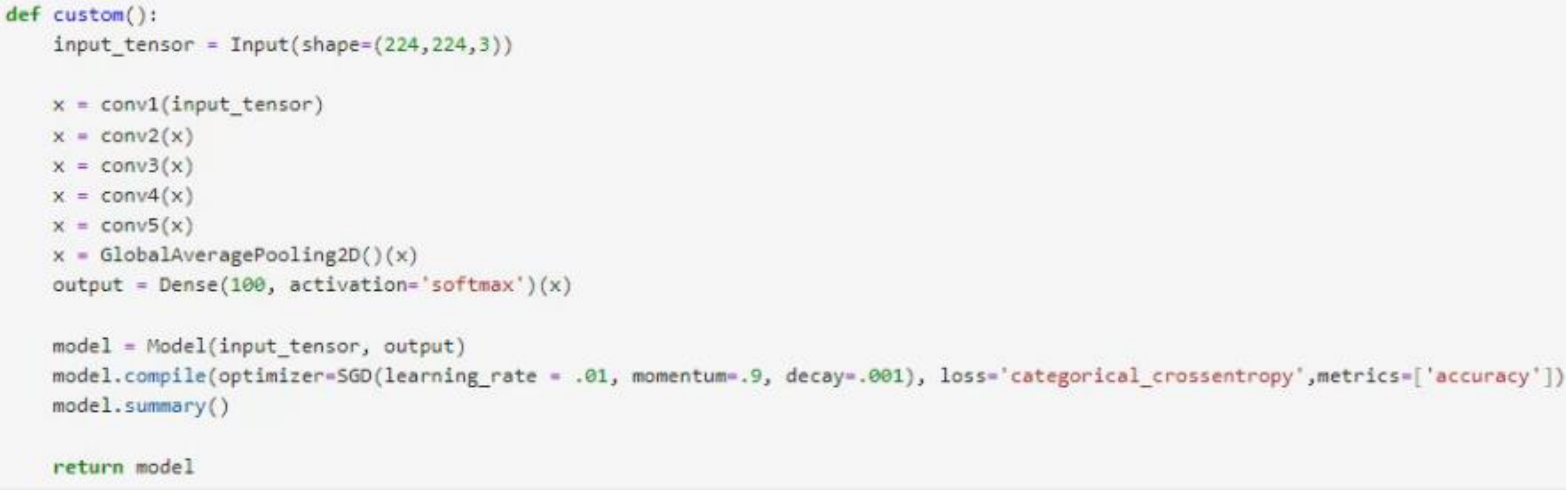
```python
def conv4(x):
    shortcut = x

    for i in range(6):
        if(i == 0):
            x = Conv2D(256, (1, 1), strides=(2, 2), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(1024, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(1024, (1, 1), strides=(2, 2), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(1024, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```



```python
def conv5(x):
    shortcut = x

    for i in range(3):
        if(i == 0):
            x = Conv2D(512, (1, 1), strides=(2, 2), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(2048, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(2048, (1, 1), strides=(2, 2), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(2048, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

<FIG 16. Custom + ResNet Code>

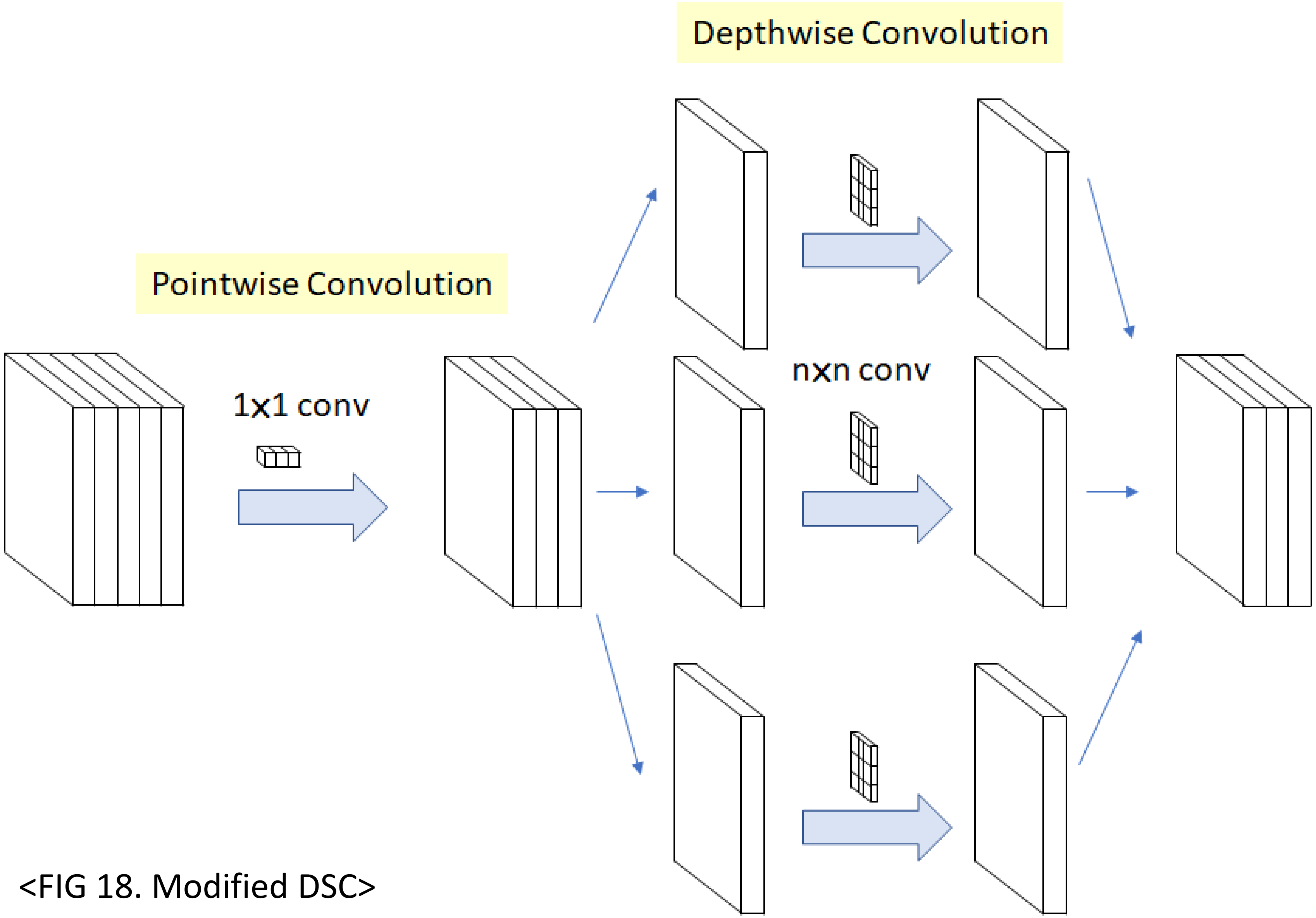
AlexNet + VGG + ResNet – Code



```
def custom():
    input_tensor = Input(shape=(224,224,3))

    x = conv1(input_tensor)
    x = conv2(x)
    x = conv3(x)
    x = conv4(x)
    x = conv5(x)
    x = GlobalAveragePooling2D()(x)
    output = Dense(100, activation='softmax')(x)

    model = Model(input_tensor, output)
    model.compile(optimizer=SGD(learning_rate = .01, momentum=.9, decay=.001), loss='categorical_crossentropy',metrics=['accuracy'])
    model.summary()

    return model
```

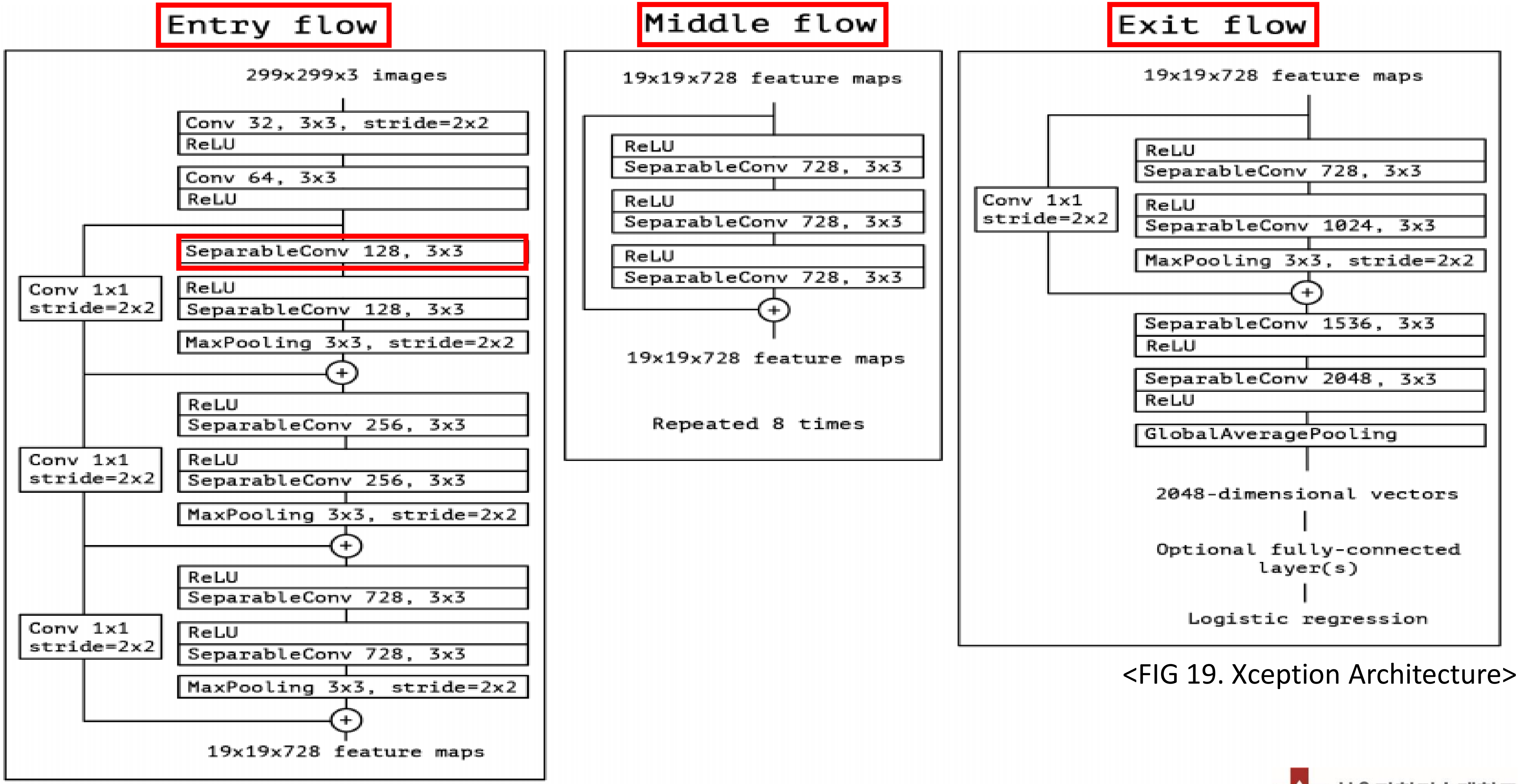
<FIG 17. Custom + ResNet Code>

Xception – Depthwise Separable Convolution



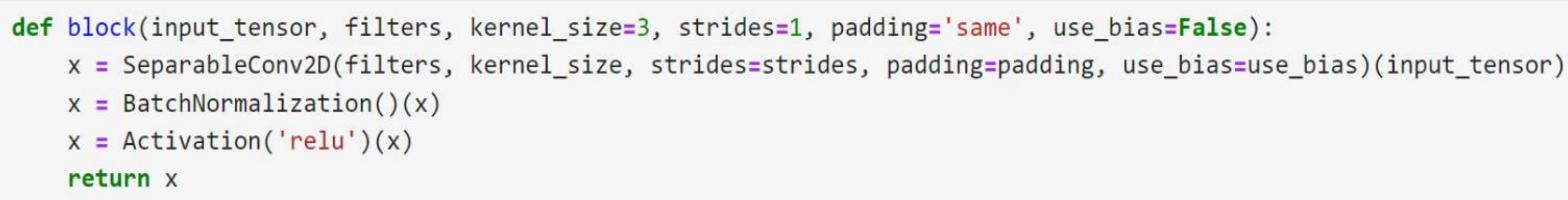
<FIG 18. Modified DSC>

Xception – Architecture



<FIG 19. Xception Architecture>

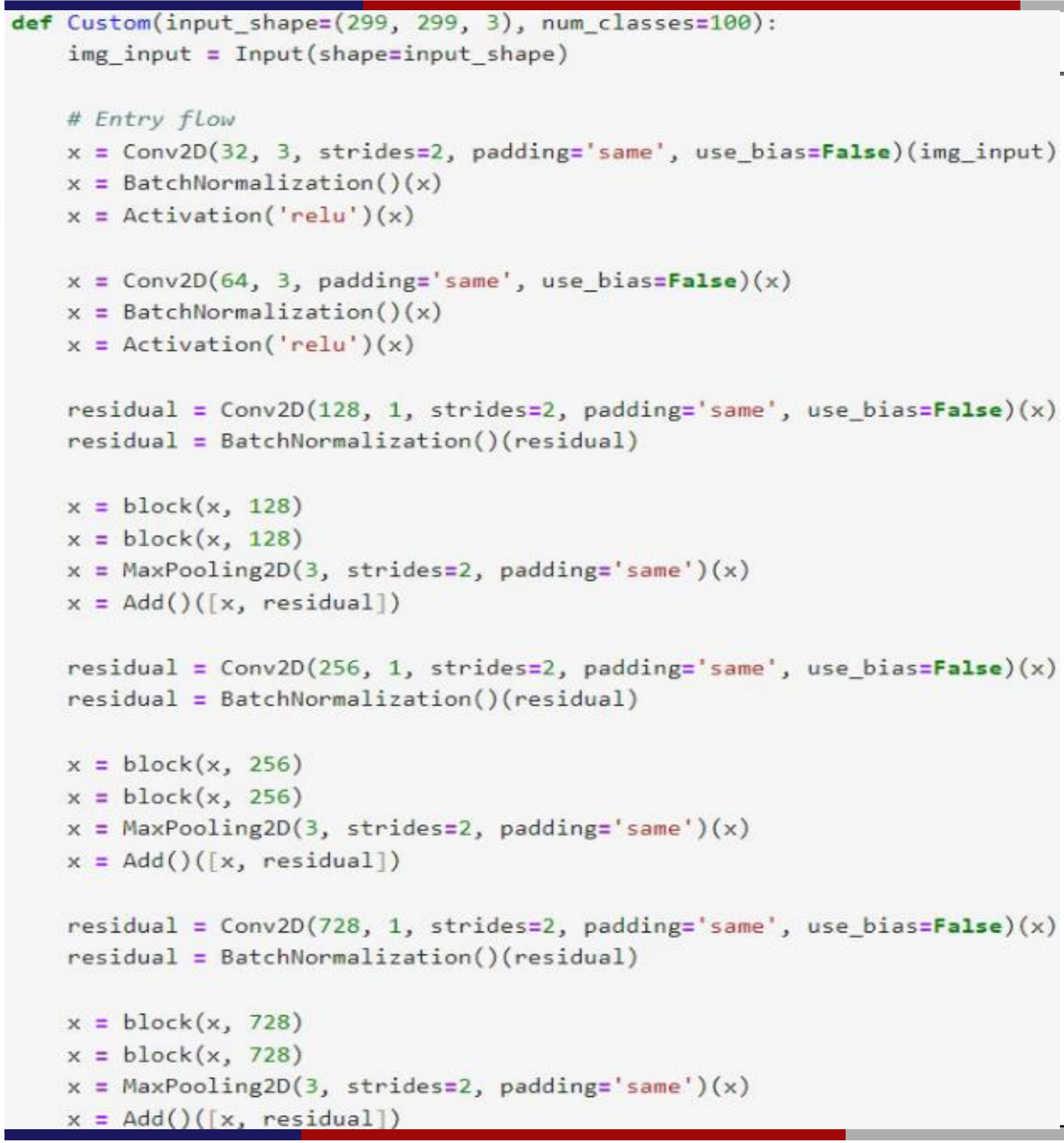
Xception – Code (SeparableConv)



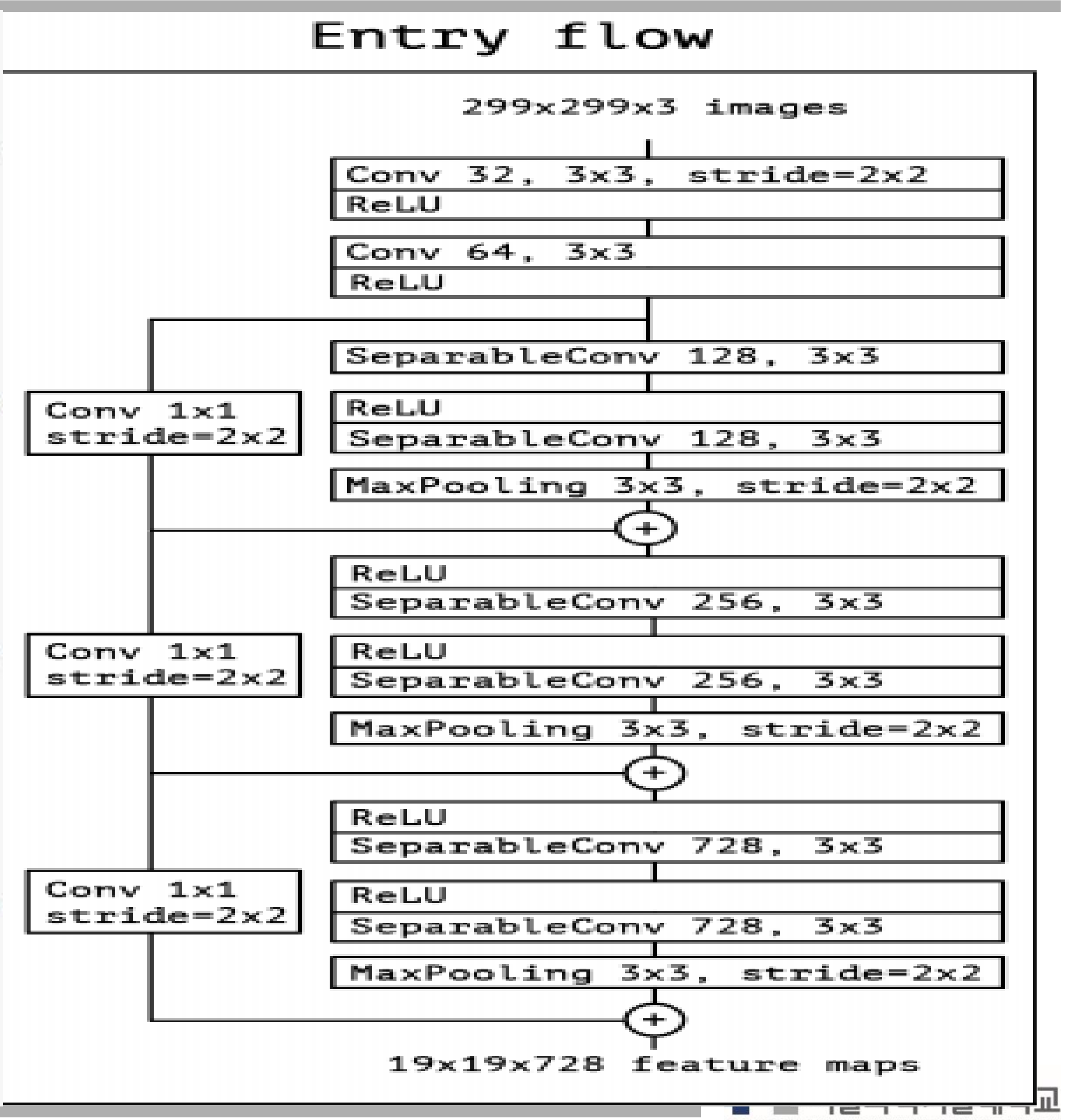
```python
def block(input_tensor, filters, kernel_size=3, strides=1, padding='same', use_bias=False):
    x = SeparableConv2D(filters, kernel_size, strides=strides, padding=padding, use_bias=use_bias)(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x
```

<FIG 20. SeparableConv Code>

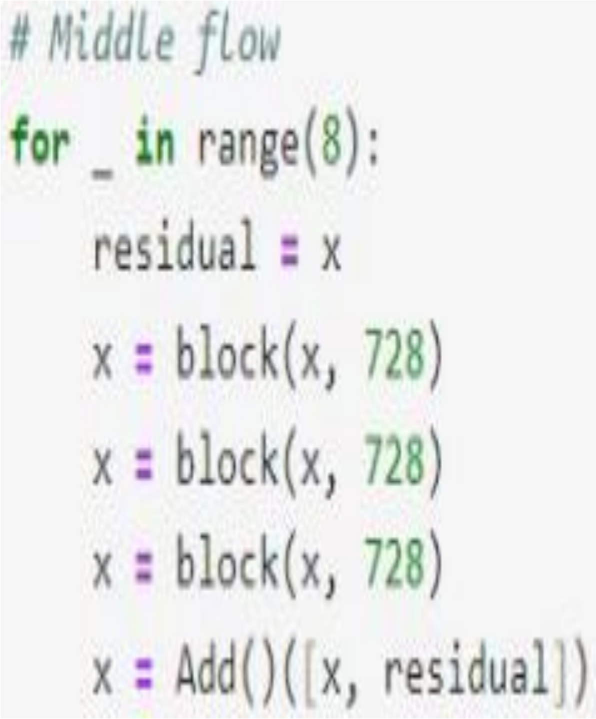
Xception – Code (Entry Flow)



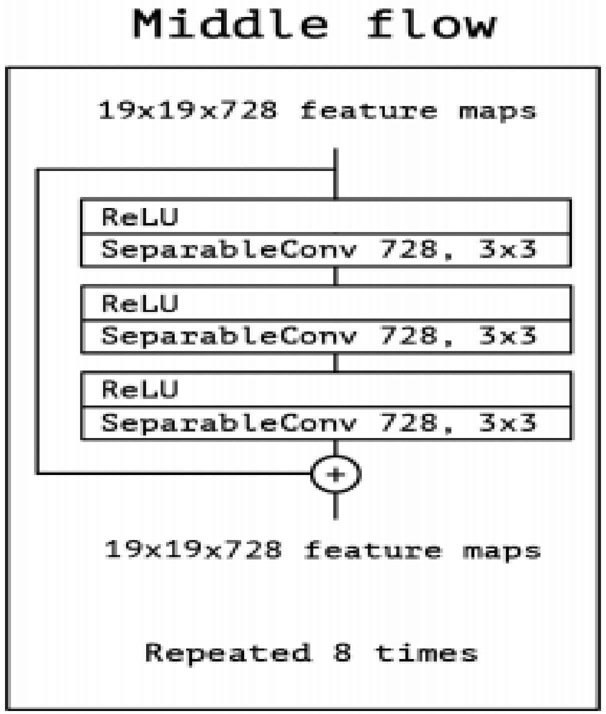
```
def Custom(input_shape=(299, 299, 3), num_classes=100):
    img_input = Input(shape=input_shape)

    # Entry flow
    x = Conv2D(32, 3, strides=2, padding='same', use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(64, 3, padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    residual = Conv2D(128, 1, strides=2, padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = block(x, 128)
    x = block(x, 128)
    x = MaxPooling2D(3, strides=2, padding='same')(x)
    x = Add()([x, residual])

    residual = Conv2D(256, 1, strides=2, padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = block(x, 256)
    x = block(x, 256)
    x = MaxPooling2D(3, strides=2, padding='same')(x)
    x = Add()([x, residual])

    residual = Conv2D(728, 1, strides=2, padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = block(x, 728)
    x = block(x, 728)
    x = MaxPooling2D(3, strides=2, padding='same')(x)
    x = Add()([x, residual])
```



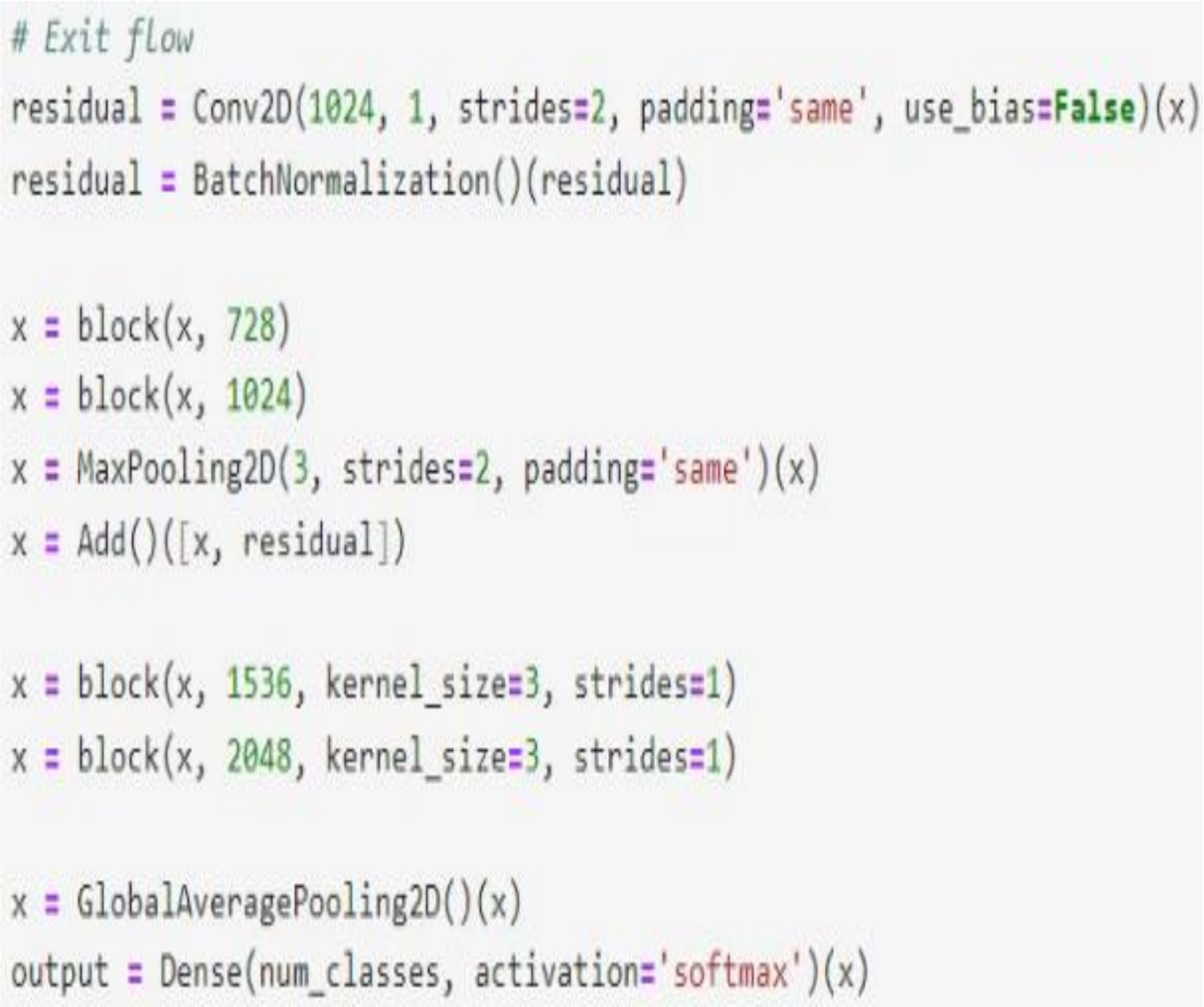
Xception – Code (Middle Flow)



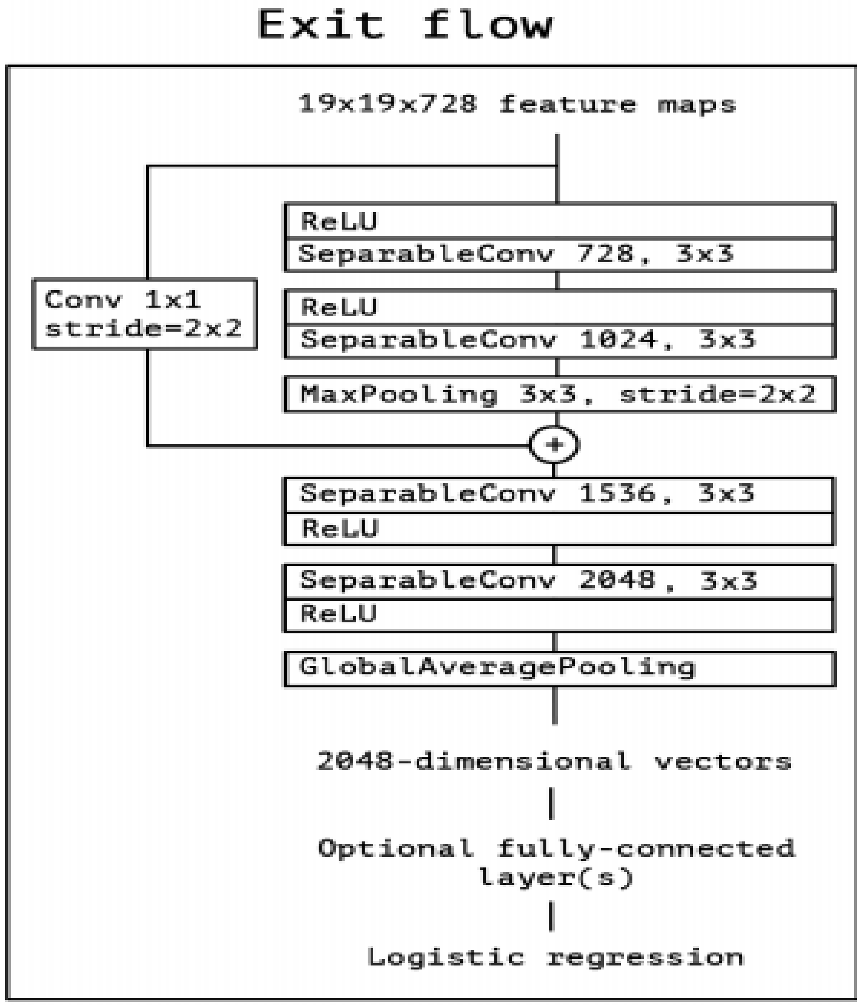
```
# Middle flow
for _ in range(8):
    residual = x
    x = block(x, 728)
    x = block(x, 728)
    x = block(x, 728)
    x = Add()([x, residual])
```



Xception – Code (Entry Flow)



```
# Exit flow
residual = Conv2D(1024, 1, strides=2, padding='same', use_bias=False)(x)
residual = BatchNormalization()(residual)

x = block(x, 728)
x = block(x, 1024)
x = MaxPooling2D(3, strides=2, padding='same')(x)
x = Add()([x, residual])

x = block(x, 1536, kernel_size=3, strides=1)
x = block(x, 2048, kernel_size=3, strides=1)

x = GlobalAveragePooling2D()(x)
output = Dense(num_classes, activation='softmax')(x)
```

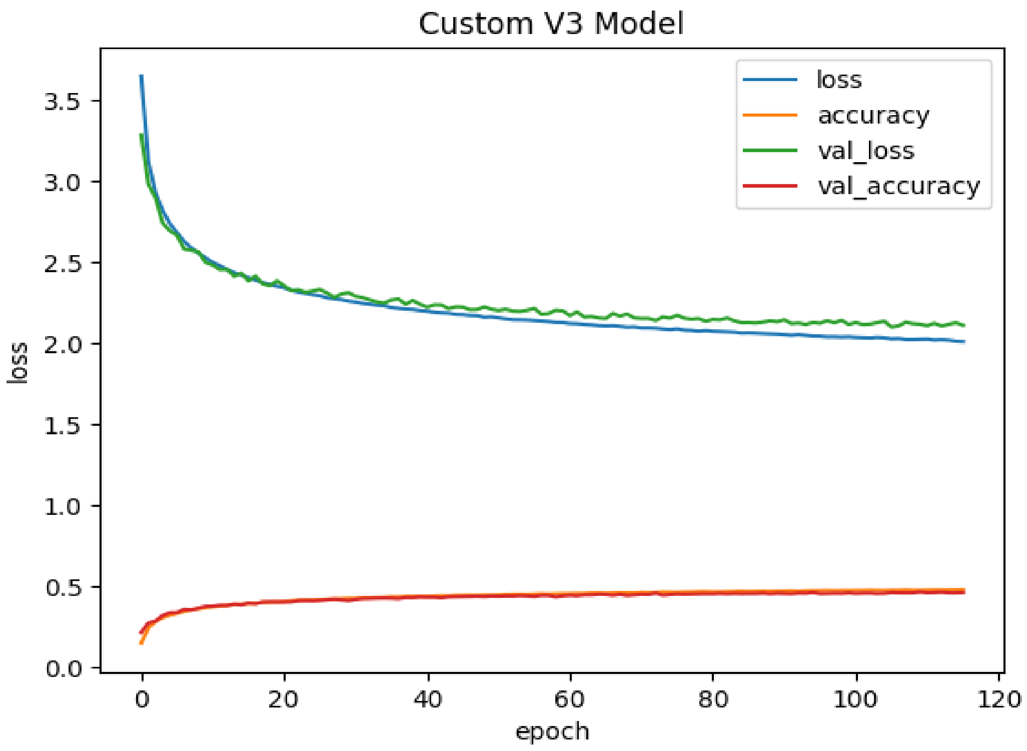


Result

Name	Params	Loss	Val_Loss	ACC	Val_ACC
AlexNet+ VGG	45,295,884	1.1039	1.5973	0.6834	0.5788

Name	Params	Loss	Val_Loss	ACC	Val_ACC
AlexNet+ VGG+ ResNet	23,858,788	2.1762	2.8649	0.4408	0.4858

Name	Params	Loss	Val_Loss	ACC	Val_ACC
Xception	21,066,380	2.0273	2.0999	0.4765	0.4646



<FIG 21. AlexNet+VGG+ResNet>

<FIG 22. Xception>

THANKS FOR YOUR ATTENTION

이찬호
국방인공지능응용학과