합성곱 신경망 (Convolution Neural Network)

3. Case Study

MNIST Dataset

CNN을 이용한 이미지 분류의 가장 기초적인 예시로서 MNIST 데이터 셋을 사용해보자.

데이터 셋은 각 픽셀이 0부터 255 사이의 값을 가지는 28×28 픽셀의 흑백 손글씨 이미지로 하나의 이미지는 (28, 28, 1) 차원을 가진다. 0부터 9까지의 숫자를 나타내는 총 10개의 클래스로 구성되어 있다. Keras에서 제공하는 MNIST 데이터 셋은 60,000개의 훈련 및 검증 셋, 10,000개의 테스트 셋으로 분리되어 있다.

첨부파일 1. **MNIST-example**

CIFAR10 Dataset

첨부파일 2. **CIFAR10-example**

CIFAR10 데이터 셋은 총 60,000개의 32×32 저해상도 컬러 이미지를 포함하고 있다. 따라서 하나의 이미지는 R, G, B 3차원의 특징을 가지는 (32, 32, 3) 배열의 형태로 정의할 수 있다. 전체 데이터 셋에서 이미지가 속한 클래스는 총 10개로 각 클래스 당 6,000개의 이미지가 존재하며 이중 각 클래스당 5,000개는 훈련 및 검증용 셋에, 1,000개는 테스트 셋에 포함된다. 각각의 클래스는 서로 오버랩되지 않는다.

▷ **10 CLASSES**

Index	0	1	2	3	4	5	6	7	8	9
Label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck

앞에서 다룬 CNN의 층별 속성 및 구조를 변화시켜가며 다양한 모형을 고려하여 CIFAR10 데이터 셋에 적용해보기로 한다. 먼저 5만 개의 훈련 및 검증용 데이터 셋을 4만 개의 훈련 셋과 만 개의 검증용 셋으로 분리하여 모형 평가에 사용한다.

▷ OVERALL STRUCTURE

Layer	Number of Output Channels	Filter Size
Input	3	
Convolution 1	16	(3, 3)
Convolution 2	32	(3, 3)
Convolution 3	64	(3, 3)
Dense	128	-
Output	10	-

적합할 CNN 모형의 전반적인 구조는 총 세 블록의 합성곱(Convoution)-풀링(Pooling) 층을 필수적으로 거쳐 최종 출력(Output) 층과 완전 연결되도록 설정하였다.

먼저 각 블록당 하나의 합성곱 층을 거쳐 출력된 특징 맵을 일렬로 배열하고 마지막 출력 층과 곧장 완전 연결함으로써 클래스 분류가 이루어지도록 하는 비교적 간단한 구조부터, 각 블록의 연속된 합성곱 층의 수를 증가시키고 일렬로 배열한 특징 맵과 최종 출력 층 사이에 완전 연결 은닉 층(Fully-Connected Hidden Layer)을 추가하는 등 점점 모형의 구조를 깊게 설계하여 결과의 차이를 확인하였다. 모형이 깊어짐에 따라 발생할 수 있는 과대적합의 위험을 해소하기 위해 일부 모형에서는 **배치 정규화** 또는 **드롭아웃** 방법을 적용하였다.

▷ HYPERPARAMETER

epochs	batch size	activation	kernel intializer	dropout rate
100	32	'relu'	'he_uniform'	0.1

▷ CALLBACKS

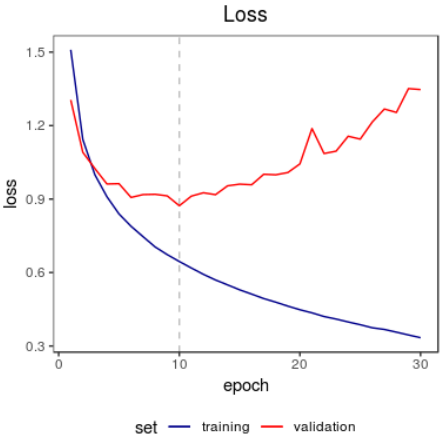
모든 모형은 최대 100번의 반복 동안 하나의 epoch 안에서 여러 번의 미니 배치를 반복해 가중치를 업데이트한다. 즉 배치 사이즈를 32로 설정할 경우 학습에 사용하는 훈련 이미지는 총 4만 개이므로 epoch 하나당 40000/32=1250번의 미니배치가 진행된다. 검증용 셋에 대한 loss와 accuracy의 평가는 epoch 하나가 종료될 때마다 진행되며 Keras의 Callback 옵션을 통해 학습의 조기 종료 및 최적 모형 저장 등을 수행할 수 있다.

- **callback_model_checkpoint**(filepath, monitor = 'val_loss', save_best_only = FALSE, period = 1)
  지정한 경로에 모형의 체크포인트를 저장하여 **save_best_only**=TRUE로 설정할 경우 총 반복횟수(epochs) 중 설정한 측정치(monitor) 기준으로 최적의 모형을 저장하도록 한다. monitor로 설정할 측정치는 validation loss 또는 accuracy가 될 수 있다.

- **callback_early_stopping**(monitor = 'val_loss', min_delta = 0, patience = 0)
  monitor로 설정한 측정치가 지정한 patience 이상의 epoch 동안 향상되지 않을 경우 학습을 조기 종료한다.

예를 들어 다음과 같이 callback 옵션을 설정하였다고 하자. 지정하지 않은 다른 옵션들은 default 값으로 가정한다.

```
callback_model_checkpoint(filepath, monitor = 'val_loss', save_best_only = TRUE)
callback_early_stopping(monitor = 'val_loss', patience = 20)
```



위 옵션을 지정한 모형은 학습이 진행됨에 따라 각 epoch의 마지막에 검증용 셋에 대해 평가되는 loss를 모니터링한다. 각 epoch 종료 시마다 현재의 loss가 이전 epoch의 loss보다 감소했다면 checkpoint에 지정한 경로에 현재의 모형을 저장한다. (감소하지 않았다면 저장하지 않고 다음 epoch를 진행한다.) 오른쪽의 그림은 전체 epochs를 100으로 설정하고 학습을 시작한 모형의 히스토리 예시이다. 10번째 epoch에서 loss가 가장 작으며 이후 20번의 epoch 동안 loss가 10번

째 loss보다 감소하지 않았다. 따라서 30번째 epoch를 마지막으로 전체 학습이 중단되었음을 알 수 있다. 최종적으로 10번째 epoch에 학습된 모형 및 가중치가 checkpoint의 경로에 저장된다.

본 예시에서는 'Validation Accuracy'를 기준으로 20번의 epoch 동안 모형이 향상되지 않을 경우 학습을 종료하고 최적 모형을 저장하도록 하였다. R Keras를 이용한 모든 모형 적합의 코드 및 결과는 첨부파일에 포함하였으며, 이 중 일부를 아래에 요약하였다.

1. BASIC MODEL

Block 1		Block 2		Block 3		Flatten	Output
Convolution	Pooling	Convolution	Pooling	Convolution	Pooling		

총 3번의 합성곱-최대 풀링을 거쳐 일렬 배열 후 최종 출력층과 완전 연결된다.

- DEFINE

```
# Define
model1 <- keras_model_sequential() %>%
  layer_conv_2d(input_shape = c(img_size, img_size, 3),
                16, c(3, 3), activation = 'relu', padding = 'same',
                kernel_initializer = 'he_uniform',
                name = '1_conv') %>%
  layer_max_pooling_2d(c(2, 2), padding = 'same', name = '1_pool') %>%
  layer_conv_2d(32, c(3, 3), activation = 'relu', padding = 'same',
                kernel_initializer = 'he_uniform',
                name = '2_conv') %>%
  layer_max_pooling_2d(c(2, 2), padding = 'same', name = '2_pool') %>%
  layer_conv_2d(64, c(3, 3), activation = 'relu', padding = 'same',
                kernel_initializer = 'he_uniform',
                name = '3_conv') %>%
  layer_max_pooling_2d(c(2, 2), padding = 'same', name = '3_pool') %>%
  layer_flatten(name = 'flatten') %>%
  layer_dense(n_class, activation = 'softmax', name = 'output')

# Compile
model1 %>% compile(
  loss = 'categorical_crossentropy',
  metrics = 'accuracy',
  optimizer = optimizer_rmsprop(lr = 1e-3, decay = 1e-6)
)
```

```
## Model
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## 1_conv (Conv2D)              (None, 32, 32, 16)        448
## _________________________________________________________________
## 1_pool (MaxPooling2D)        (None, 16, 16, 16)        0
## _________________________________________________________________
## 2_conv (Conv2D)              (None, 16, 16, 32)        4640
## _________________________________________________________________
## 2_pool (MaxPooling2D)        (None, 8, 8, 32)          0
## _________________________________________________________________
## 3_conv (Conv2D)              (None, 8, 8, 64)          18496
## _________________________________________________________________
## 3_pool (MaxPooling2D)        (None, 4, 4, 64)          0
## _________________________________________________________________
## flatten (Flatten)            (None, 1024)              0
## _________________________________________________________________
## output (Dense)               (None, 10)                10250
## =================================================================
## Total params: 33,834
## Trainable params: 33,834
## Non-trainable params: 0
## _________________________________________________________________
```

- TRANING

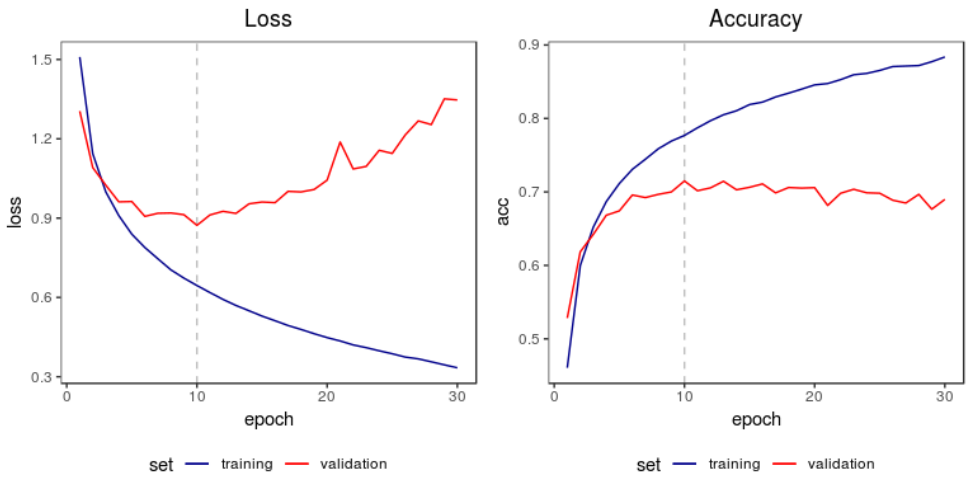
```
modelpath <- 'Model/model-user-he-1.hdf5'
saving <- callback_model_checkpoint(modelpath, monitor = 'val_acc', save_best_only = TRUE)
stopping <- callback_early_stopping(monitor = 'val_acc', patience = 20)
```

```
epochs <- 100
batch_size <- 32

# Start Training
history1 <- model1 %>% fit(
  Xtrain, Ytrain, validation_data = list(Xval, Yval),
  epochs = epochs, batch_size = batch_size,
  callbacks = list(saving, stopping), verbose = 2
)

# Save History to a CSV file
write.csv(as.data.frame(history1$metrics), 'Model/history-user-he-1.csv', row.names = FALSE)
```

- HISTORY



- **EVALUATION**

```
# Evaluation Table for Each Sample Set
evaluate.table <- function(model) {
  result_train <- model %>% evaluate(Xtrain, Ytrain, verbose = 2) %>% unlist
  result_val <- model %>% evaluate(Xval, Yval, verbose = 2) %>% unlist
  result_te <- model %>% evaluate(Xtest, Ytest, verbose = 2) %>% unlist
  as.data.frame(rbind(
    train = result_train, validation = result_val, test = result_te
  ))
}
```

```
model1 <- load_model_hdf5(modelpath)
```

checkpoint에 저장된 최적 모형을 다시 불러와 테스트 셋에 대한 검증을 수행한다.

```
result1 <- evaluate.table(model1)
result1
```

	loss	acc
train	0.5409	0.8176
validation	0.8728	0.7149
test	0.8683	**0.7104**

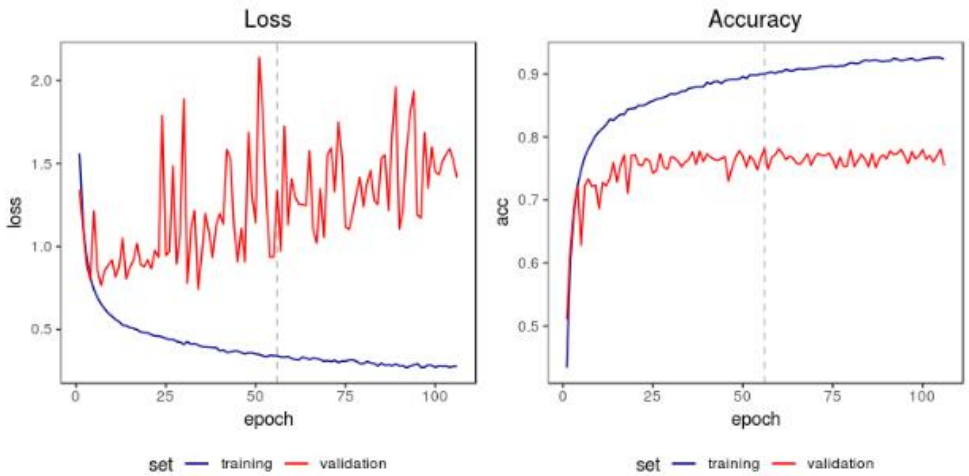
2. DEEP MODEL

Block 1				Block 2				Block 3				Flatten	Dense	Dense	Output
Conv	Conv	Conv	Pool	Conv	Conv	Conv	Pool	Conv	Conv	Conv	Pool				

블록 하나당 연속된 세 개의 합성곱 층 및 최대 풀링 층을 포함하며 총 세 블록을 거쳐 일렬 배열 후 마지막 출력 층 이전에 2개의 완전연결 은닉층을 거쳐 분류되도록 한다. 완전 연결 은닉층을 추가한 깊은 모형들의 경우는 총 epochs 수를 300, patience를 50으로 조금 더 길게 설정하였다.

- 각 합성곱 층에서 활성화 함수에 입력되기 전 배치 정규화 스텝을 거치도록 설정하였다.
- 각 완전 연결 은닉 층 이후에는 드롭아웃(rate=0.1) 층을 추가하여 학습마다 노드 중 10%를 제거하도록 하였다.

- **HISTORY**



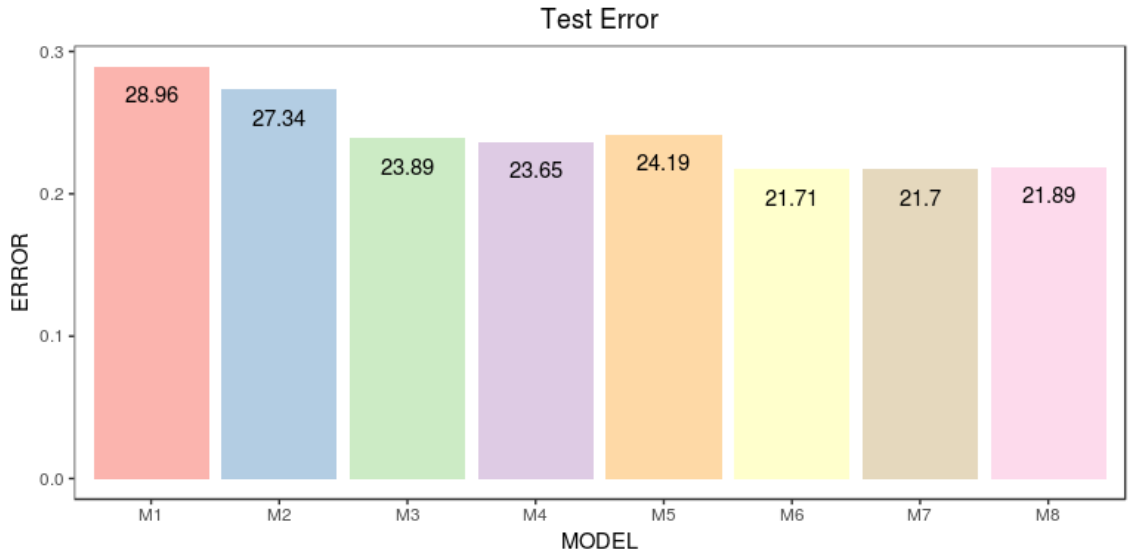
- **EVALUATION**

	loss	acc
train	0.2965	0.9184
validation	1.3346	0.7821
test	1.3401	**0.7811**

▷ **RESULT SUMMARY**

위에서 가장 간단한 모형과 가장 복잡한 모형의 결과만을 비교해보면 합성곱 층의 추가, 배치 정규화 및 드롭아웃을 이용한 과대적합 방지 등을 통해 테스트 셋에 대한 정확도가 약 7% 가까이 향상된 것을 확인할 수 있다.

아래는 본 예시에서 고려한 모든 모형에 대한 테스트 이미지의 오분류율을 비교한 결과이다.



MODEL	NC	BN	FC	ERROR
M1	1	NO	0	0.2896
M2	2	NO	0	0.2734
M3	2	YES	0	0.2389
M4	2	YES	1	0.2365
M5	2	YES	2	0.2419
M6	3	YES	0	0.2171
M7	3	YES	1	0.2170
M8	3	YES	2	0.2189

- **NC**: 블록 당 연속된 합성곱 층의 수
- **BN**: 합성곱 층의 배치 정규화 여부
- **FC**: 마지막 출력 층 이전의 완전 연결 은닉층 수

테스트 셋에 대한 오분류율이 가장 작은 모형은 블록 당 세 개의 합성곱 층과 배치 정규화를 거쳐 은닉 층 없이 최종 출력층으로 완전연결되는 **M6** 모형과 이에 완전 연결 은닉 층 하나를 추가한 **M7** 모형이다.

블록 당 합성곱 층의 수가 증가할수록 테스트 셋에 대한 정확도가 향상되었다. 또한 M2과 M3의 차이에서 **배치 정규화**를 진행함으로써 정확도를 높이고 적절한 과대적합 방지 효과를 얻을 수 있음을 확인할 수 있다. 그러나 M3와 M4 및 M5, M6와 M7 및 M8을 비교해볼 때 본 예시에서 모형의 마지막 단에 완전연결 층을 추가함에 따른 정확도 향상의 효과는 미미하였다.

CIFAR10 데이터 셋을 이용해 합성곱 신경망의 다양한 구조를 도입하여 정확도를 향상시켜 보았다. 데이터 및 다양한 하이퍼파라미터, 옵티마이저 등의 설정에 따라 최적의 성능을 보이는 모형의 구조는 정해져 있지 않으므로, 모형의 옵션을 점진적으로 조정해보고 Cross-Validation을 통해 최적의 모형을 선택하도록 할 수 있다.

4. 클래스 활성도 시각화: CAM 및 Grad-CAM

CNN이 이미지(또는 시퀀스)에서 특정 클래스를 식별하기 위해 중요하게 여기는 특징이 정확히 입력 데이터에서 어디에 위치하고 있는지를 역으로 추적하여 이를 시각적으로 확인함으로써 이미지의 어떤 부분이 각 클래스 분류에 영향을 미치는지를 확인할 수 있다. 즉 이것은 각 클래스 별로 입력 이미지에서 활성화되는 부분을 시각화하는 방법이다. 이를 통해 올바르게 분류된 이미지에 대해서는 해당 클래스를 나타내는 중요한 특징이 무엇인지, 오분류된 이미지에 대해서는 오분류에 결정적인 영향을 끼친 부분이 어디인지와 같은 정보를 얻을 수 있다.

Layer	Output Shape
Input	(None, 28, 28, 1)
Conv2D(filters=**32**, kernel_size=**(3,3)**, activation='**relu**', input_shape=(28,28,3))	(None, 26, 26, 32)
...	
Conv2D(filters=**64**, kernel_size=**(3,3)**, activation='**relu**')	(None, 11, 11, 64)
GlobalAveragePoolind2D	(None, 64)
Dense(units=10, activation='**softmax**')	(None, 10)

A. Class Activation Mapping (CAM)

CNN의 마지막 층에서 출력된 특징 맵에 Global Average Pooling(GAP)을 적용한 클래스 활성도 시각화(CAM)는 앞서 예시를 통해 설명한 Flatten 층 대신에 GAP 층을 사용하여 특징 맵을 일렬 배열로 만든다.

이 GAP 층에서는 마지막 합성곱 층에서 출력되는 특징 맵이 k개라고 할 때, 각 특징맵 별로 원소를 모두 더하여 하나로 평균 낸 총 k개의 값을 일렬로 배열한다. 이 값들은 마지막 출력 층과 완전 연결되어 최종 클래스를 분류하는 데 사용된다.

k번째 특징 맵의 값들 중 (x, y) 위치의 값을 $f_k(x,y)$라고 표현한다. 그렇다면 GAP를 거쳐 얻어지는 F_k는 하나의 특징 맵 내의 모든 x, y 위치의 값을 다 더한(평균) 값으로 class c에 대해 계산되는 스코어 S_c는 다음과 같이 표현할 수 있다. 여기서 w_k^c는 k번째 특징맵의 GAP 값 F_k와 class c에 대응되는 가중치이며 w_k^c가 클수록 F_k가 class c에 미치는 영향이 커지게 되므로 본질적으로 class c에 대한 특징 맵 k의 중요도를 나타낸다고 할 수 있다.

$$(1)\ F_k = \sum_x \sum_y f_k(x,y)$$

$$(2)\ S_c = \sum_k (class\ feature\ weight) \cdot (average\ of\ feature\ map)$$

$$= \sum_k w_k^c \cdot F_k$$

$$= \sum_k w_k^c \cdot \sum_x \sum_y f_k(x,y)$$

$$= \sum_k \sum_x \sum_y w_k^c \cdot f_k(x,y)$$

$$= \sum_x \sum_y \sum_k w_k^c \cdot f_k(x,y) = \sum_y \sum_k M_c(x,y)$$

$$\rightarrow \hat{p}_c = \frac{\exp(S_c)}{\sum_c \exp(Sc)}$$

이렇게 $M_c(x,y)$는 직접적으로 (x, y)에 위치한 값이 입력 이미지를 c라는 class로 분류하는데 미치는 중요도를 의미하게 되며 모든 x, y 위치에 대해 class c에 대한 활성도 맵을 얻을 수 있다. 만약 패딩을 하지 않아 마지막 특징 맵의 크기가 입력 이미지보다 축소된 크기라면 이를 다시 원래의 이미지와 동일하게 리사이징한 후 이미지와 겹쳐 이미지의 어떤 부분이 특정 카테고리와 연관성이 높은지를 시각적으로 확인할 수 있게 한다.

B. Gradient-weighted CAM

위와 같이 CAM은 클래스 분류를 위한 출력 층 이전의 마지막 층이 GAP로 이루어진 특수한 구조에서 구현 가능하다. 일반적으로 성능이 높은 많은 CNN의 구조는 합성곱 층 이후에 몇 개의 완전 연결 층을 쌓은 구조로 이루어져 있어 CAM을 사용하고자 할 경우 이를 GAP로 대체하고 다시 학습시켜야 하는 문제가 있다. 이러한 제약 없이 다양한 구조의 CNN에서도 활성도 시각화를 구현할 수 있도록 일반화된 방법인 Grad-CAM이 개발되었다.

Grad-CAM은 네트워크에서 마지막 합성곱 계층과 클래스 출력 계층의 Score 사이에서 구조에 영향을 받지 않고 그래디언트를 이용해 클래스 별 특정 위치의 활성도를 계산할 수 있어 보다 범용적으로 사용 가능한 방법이다.

▷ **GRAD-CAM PROCESS with R**

```
x <- img
# 입력 이미지 x에 대한 예측확률 벡터 (각 클래스에 대한 스코어)
pred_prob <- model %>% predict(x)

# 스코어 가장 큰 클래스
c <- which.max(pred_prob)

# ------------------------ 연산을 위해 호출할 Tensor 정의 ------------------------ #
# 모형의 마지막 합성곱 층
last_conv_layer <- model %>% get_layer(layer_name)

# 예측 벡터의 class c 항목
x_output <- model$output[,c]

# 입력 이미지 x에 대한 모형의 출력 값을
# 마지막 합성곱 층에서 출력되는 특징 맵의 각 원소에 대해 편미분한 그래디언트
grads <- k_gradients(loss = x_output,
                     variables = last_conv_layer$output)[[1]]

# 개별 특징 맵 별 평균 그래디언트(강도)*
# '하나의 특징맵이 클래스 c로 분류하는 데 기여하는 정도'를 의미한다.
pooled_grads <- k_mean(grads, axis = c(1, 2, 3))
# ------------------------------------------------------------------------------ #

# 정의한 Tensor들에 대해 실제로 값을 얻을 수 있도록 keras 함수를 정의한다.
iterate <- k_function(inputs = list(model$input),
                      outputs = list(pooled_grads, last_conv_layer$output[1,,,]))

# 함수를 실행해 실제로 입력 이미지 x에 대한 값을 얻는다.
c(pooled_grads_value, conv_layer_output_value) %<-% iterate(list(x))

# 마지막 합성곱 층에서 출력되는 특징 맵의 개수
num_maps <- last_conv_layer$output_shape[[4]]

# 마지막 합성곱 층에서 출력된 각 특징 맵의 원소에 계산된 평균 그래디언트(강도)를 곱해준다.
for (i in 1:num_maps) {
  conv_layer_output_value[,,i] <- conv_layer_output_value[,,i] * pooled_grads_value[[i]]
}
```

```r
# 특징 맵의 각 위치에 대해서 채널 방향으로 평균을 구한다. (channel-wise mean)
# 클래스 활성화 기여도를 나타내는 히트맵***
heatmap <- apply(conv_layer_output_value, c(1, 2), mean)

# 시각화를 위해 양수만을 취하고 0-1 사이의 값으로 변환한다.
heatmap <- pmax(heatmap, 0)
heatmap <- heatmap / max(heatmap)
```

▷ **EXAMPLE**

R Keras 어플리케이션에서 제공하는 ImageNet의 사전 학습된 VGG16 모형을 이용해 인터넷 상의 몇 가지 고해상도 이미지에 대해 Grad-CAM Process를 적용하여 히트맵을 계산하였다. 또한 히트맵을 원본 이미지에 겹쳐 그리기 위해 Python OpenCV를 활용하였다.

첨부파일 3. **Grad-CAM-example**

첨부파일 4. **CAM-OpenCV**

- RESULT

African Elephant **Golden Retriever** **Racer**

- 아프리카 코끼리 클래스로 분류된 코끼리 이미지의 경우 아프리카 코끼리로 분류함에 있어 코끼리의 코와 귀 부분이 강하게 활성화되었음을 알 수 있다.

- 골든 리트리버 클래스로 분류된 강아지 이미지의 경우 세 마리의 눈 부분이 모두 활성화되어 있다.

- 모형에서 자동차 이미지를 경주용 자동차(racer) 클래스로 분류하였으며 이에 강하게 기여한 부분은 자동차의 헤드 라이트 부분이 특히 활성화되었다.