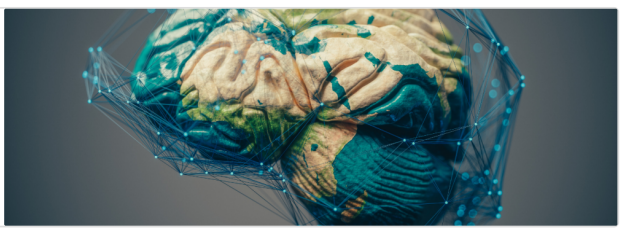
# XAI_01. Explainable AI : Scene Classification with ResNet-18 and Grad-CAM Visualization

| | |
|---|---|
| 생성일 | @2022년 8월 16일 오전 11:17 |
| 유형 | 머신러닝/딥러닝 |
| 작성자 | 동훈 오 |

Explainable AI: Scene Classification with ResNet-18 and Grad-CAM Visualization

Scene Classification is a special task in Computer Vision. Unlike Object Classification, which focuses on classifying prominent objects in the foreground, Scene Classification uses the layout of objects within the scene, in addition to the ambient context, for classification ( King et al, 2017).

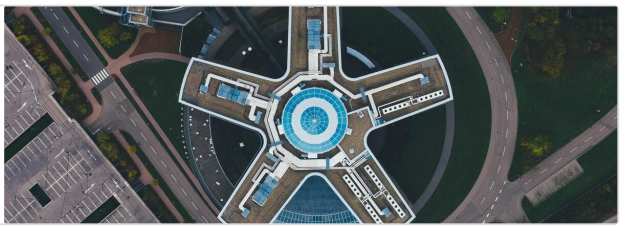
https://medium.com/mlearning-ai/explainable-ai-scene-classification-with-resnet-18-and-grad-cam-visualization-17ae6d65bb0



Intel Image Classification

Image Scene Classification of Multiclass

https://www.kaggle.com/datasets/puneet6060/intel-image-classification/code



https://github.com/baotramduong/Explainable-AI-Scene-Classification-and-GradCam-Visualization

colab 에서 ‘intel image classification kaggle 데이터셋 사용’ 관련 레포

Intel-Image-Classification-Kaggle/intel image classification.py at master · kpanwala/Intel-Image-Classification-Kaggle

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters. Learn more about bidirectional Unicode characters You can't perform that action at this time. You signed in with another tab or window.

https://github.com/kpanwala/Intel-Image-Classification-Kaggle/blob/master/intel%20image%20classification.py

kpanwala/**Intel-Image-Classification-Kaggle**

This is image data of Natural Scenes around the world.

1 0 0 0

## Introduction

‘장면 분류’ 는 컴퓨터 비전에서 특별한 task 입니다. 장면 분류는 분류를 위해 주변 컨텍스트 외에도 장면 내의 객체 레이아웃을 사용합니다.(King et al, 2017) 이 프로젝트는 위성 이미지에서 풍경 유형을 감지하는 데 실제로 사용될 수 있습니다.

Explainable AI 또는 XAI 는 AI 시스템이 어떻게 결정을 내렸는지 인간에게 설명하는 것을 목표로 하는 머신 러닝의 새로운 분야입니다. ‘설명 가능성’은 개발자가 시스템이 예상대로 작동하는지, 규제 표준을 충족해야 하는지 또는 결정의 영향을 받는 사람들이 해당 결과에 도전하거나 변경하도록 하는데 도움이 될 수 있습니다.

## 예제 프로젝트

CNN 출력을 설명함으로써 XAI 에 대해 알아보도록 하겠습니다. 예제 프로젝트에서는 CNN 출력을 설명하는 방식으로, Grad-CAM (Gradient-Weighted Class Activation Mapping)을 사용합니다.

### 문제 설명

이 프로젝트에서 우리는 이미지에서 풍경 유형을 감지하기 위해 Residual Blocks 를 사용하여 CNN 을 구축하고 훈련할 것입니다. 또한 Grad-CAM 을 통해 입력 영역을 시각화하고 CNN 모델이 어떻게 생각하고 결정을 내리는지 설명하는 데 파악할 것입니다.

### 데이터

- 96*96 크기의 회색조 이미지를 사용합니다.

- 분류 클래스는 6가지 입니다. building, forest, glacier, mountain, sea, street
- 훈련 데이터셋은 14,034개의 이미지, 테스트 셋은 3,000개의 이미지로 구성
- 6개의 클래스에 따라 이미지 또한 6개의 하위 폴더에 저장되어 있습니다.

### 성능지표

- 정확도 점수
- 정밀도 점수
- 재현율
- F1 score
- 혼동 행렬(confusion matrix) : TP, TN, FP, FN

### 라이브러리 import

```
#import the necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from IPython.display import display
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LearningRateScheduler
import os
```

### 데이터 가져오기

( colab 상에서 예제를 진행하기 때문에 데이터 경로 지정 및 가져오는 부분은 다를 수 있다. )

```
#check the number of images in training, validation and test dataset

train = []
test = []

#os.listdir returns the list of files in the folder, in this case image class names
for i in os.listdir('./seg_train'):
  train_class = os.listdir(os.path.join('seg_train', i))
  train.extend(train_class)
  test_class = os.listdir(os.path.join('seg_test', i))
  test.extend(test_class)

print('Number of train images : {} \nNumber of test images : {}'.format(len(train), len(test)))
```

### 데이터 시각화

6개의 클래스에서 각 5 개의 샘플 이미지를 시각화 해 본다. ⇒ 6행 * 5열의 이미지 배치

```
#visualize the images in the dataset
fig, axs = plt.subplots(6,5, figsize=(32,32))
count = 0
for i in os.listdir('./seg_train'):
  # get the list of images in the particualr class
```

```
  train_class = os.listdir(os.path.join('seg_train',i))
  # plot 5 images per class
  for j in range(5):
    img = os.path.join('seg_train',i,train_class[j])

    img = PIL.Image.open(img)
    axs[count][j].set_title(i, fontsize = 30)
    axs[count][j].imshow(img)
  count += 1

fig.tight_layout()
```

클래스에 속하는 이미지 개수를 확인한다. 클래스가 비슷한 양의 이미지를 가지고 있어야 학습이 치우지 않는다.

```
#check the number of images in each class in the training dataset
No_images_per_class = []
Class_name = []
for i in os.listdir('./seg_train'):
  train_class = os.listdir(os.path.join('seg_train', i))
  No_images_per_class.append(len(train_class))
  Class_name.append(i)
  print('Number of images in {} = {} \n'.format(i, len(train_class)))
```

**결과**

- 건물 이미지 수 = 2191
- 숲의 이미지 수 = 2271
- 빙하의 이미지 수 = 2404
- 산의 이미지 수 = 2512
- 바다의 이미지 수 = 2274
- 거리의 이미지 수 = 2382

각 클래스마다 비슷한 양의 이미지를 가지고 있다.

## Data Augmentation

강력한 이미지 분류기를 구축하려면 일반적으로 심층 네트워크의 성능을 높인기 위해 이미지 증강이 필요하다. 이미지 증강은 각 훈련 인스턴스의 무작위 회전, 이동, 뒤집기와 같은 여러 처리를 통해 인공적으로 훈련 이미지를 생성한다. (train dataset 에서만 적용)

‘ImageDataGenerator’ 는 각 클래스 폴더 내의 모든 데이터에 자동으로 레이블을 지정한다.

```
# create run-time augmentation on training and test dataset
# for training datagenerator, we add normalization, shear angle, zooming range and horizontal flip
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        validation_split=0.15,
        horizontal_flip=True)

# for test datagenerator, we only normalize the data.
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
# create datagenerator for training, validation and test dataset.
#train
train_generator = train_datagen.flow_from_directory(
        'seg_train',
        target_size=(256, 256),
        batch_size=32,
        class_mode='categorical',
        subset ='training')

#validation
validation_generator = train_datagen.flow_from_directory(
        'seg_train',
```

```
        target_size=(256, 256),
        batch_size=32,
        class_mode='categorical',
        subset ='validation')

#test
test_generator = test_datagen.flow_from_directory(
        'seg_test',
        target_size=(256, 256),
        batch_size=32,
        class_mode='categorical')
```

## 합성곱 신경망 모델 : CNN

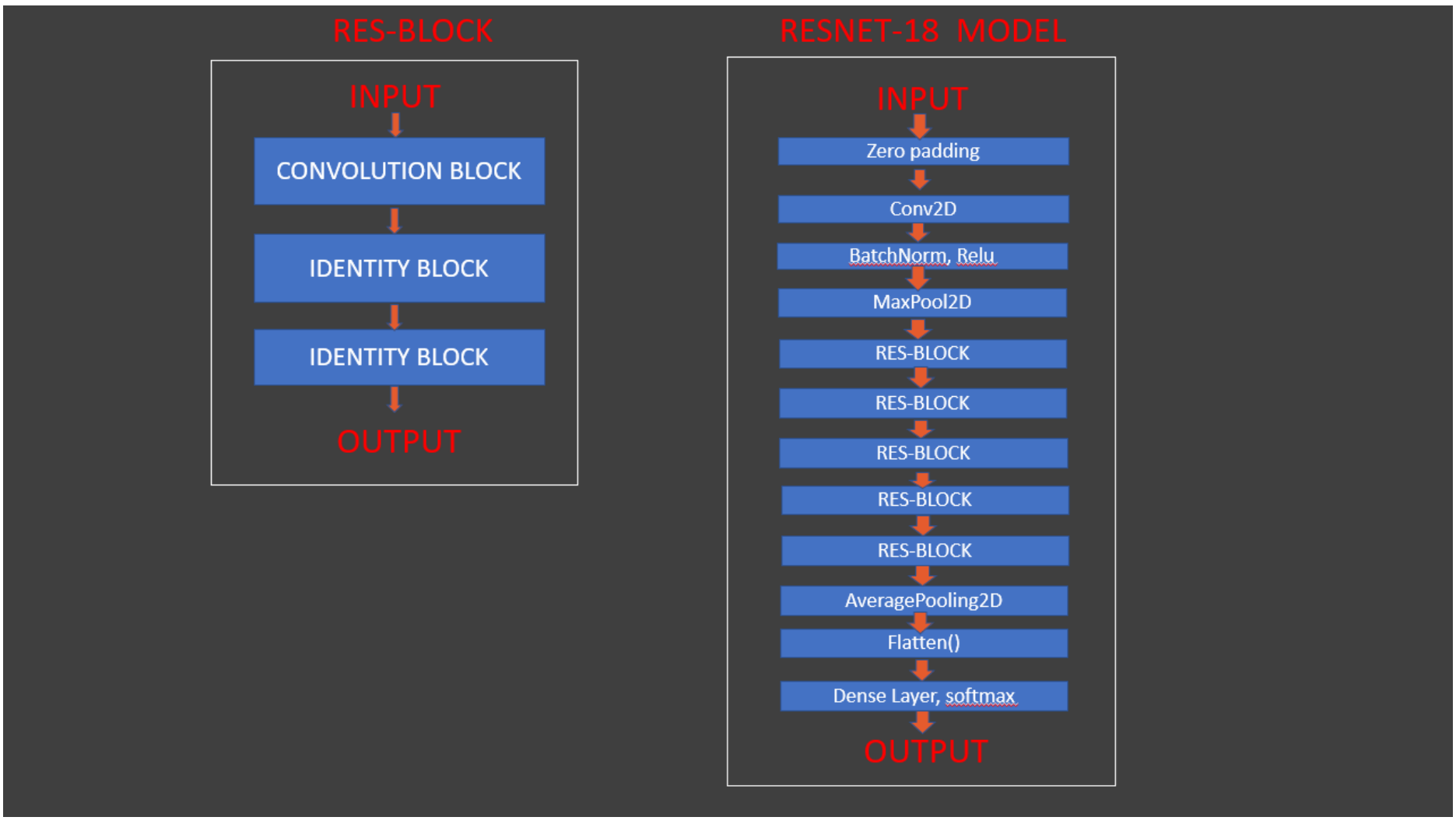
CNN 아키텍처는 convolutional layer, reLU, pooling layer, 그리고 마지막으로 완전히 연결된 Dense 레이어로 구성된다.

경사하강법으로 알려진 기술을 사용하여 인공 신경망을 훈련하기 때문에 하나의 convolutional layer + pooling layer 를 추가함에 따라 그라디언트 소실 문제가 발생하며 네트워크의 성능이 크게 저하된다. ResNet은 이러한 성능 저하 문제를 해결하기 위해 개발되었다.

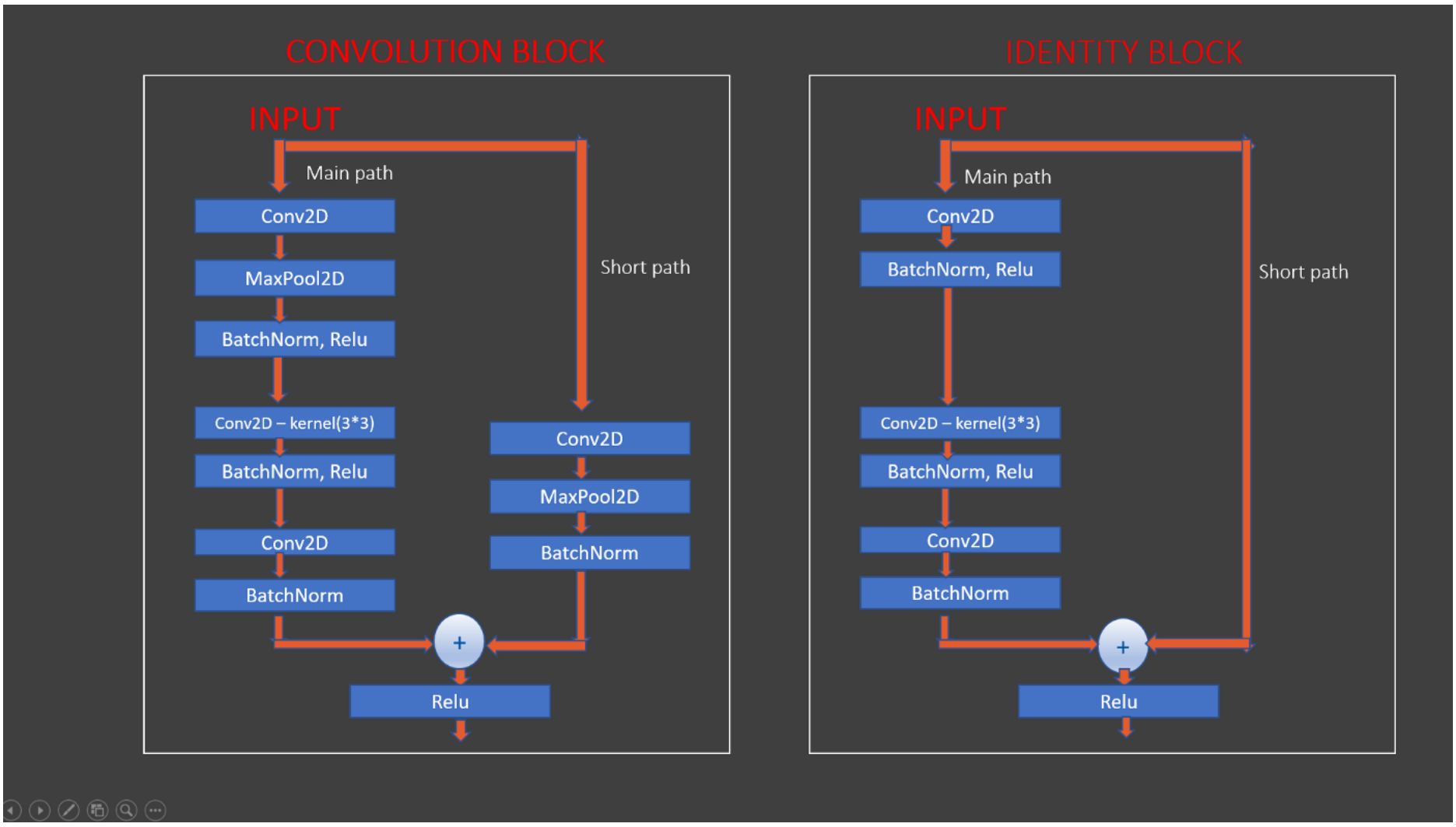
## ResNet 모델 구축

다음과 같은 아키텍처로 ResNet-18을 구축할 것이다. 이것은 Ryan Ahmed 의 ‘Guided Project’ 의 내용을 각색한 것이다.

(출처 : https://www.coursera.org/projects/scene-classification-gradcam)




RES-BLOCK 은 한 개의 CONVOLUTION BLOCK 과 2 개의 IDENTITY BLOCK 으로 구성된다. 각각의 BLOCK 의 구체적인 구성은 다음 이미지와 같다.




IDENTITY BLOCK 에서 Short path 는 입력이 그대로 출력으로 나간다.

RES_BLOCK 을 구현하기 위해 ‘Convolutional_block’, ‘Identity Block 1’, ‘Identity Block 2’ 를 구현해야 한다.

```
def res_block(X, filter, stage):

  # Convolutional_block
  X_copy = X

  f1 , f2, f3 = filter

  # Main Path
  X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_conv_a', kernel_initializer= glorot_uniform(seed = 0))(X)
  X = MaxPool2D((2,2))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_a')(X)
  X = Activation('relu')(X)

  X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+str(stage)+'_conv_b', kernel_initializer= glorot_uniform(seed = 0))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_b')(X)
  X = Activation('relu')(X)

  X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_conv_c', kernel_initializer= glorot_uniform(seed = 0))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_c')(X)



  # Short path
  X_copy = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_conv_copy', kernel_initializer= glorot_uniform(seed = 0))(X_copy)
  X_copy = MaxPool2D((2,2))(X_copy)
  X_copy = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_conv_copy')(X_copy)

  # ADD
  X = Add()([X,X_copy])
  X = Activation('relu')(X)


  # Identity Block 1
  X_copy = X


  # Main Path
  X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_identity_1_a', kernel_initializer= glorot_uniform(seed = 0))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_a')(X)
  X = Activation('relu')(X)

  X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+str(stage)+'_identity_1_b', kernel_initializer= glorot_uniform(seed = 0))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_b')(X)
  X = Activation('relu')(X)

  X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_identity_1_c', kernel_initializer= glorot_uniform(seed = 0))(X)
  X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_1_c')(X)

  # ADD
  X = Add()([X,X_copy])
```

```
X = Activation('relu')(X)


# Identity Block 2
X_copy = X


# Main Path
X = Conv2D(f1, (1,1),strides = (1,1), name ='res_'+str(stage)+'_identity_2_a', kernel_initializer= glorot_uniform(seed = 0))(X)
X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_a')(X)
X = Activation('relu')(X)

X = Conv2D(f2, kernel_size = (3,3), strides =(1,1), padding = 'same', name ='res_'+str(stage)+'_identity_2_b', kernel_initializer= glorot_uniform(seed = 0))(X)
X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_b')(X)
X = Activation('relu')(X)

X = Conv2D(f3, kernel_size = (1,1), strides =(1,1),name ='res_'+str(stage)+'_identity_2_c', kernel_initializer= glorot_uniform(seed = 0))(X)
X = BatchNormalization(axis =3, name = 'bn_'+str(stage)+'_identity_2_c')(X)

# ADD
X = Add()([X,X_copy])
X = Activation('relu')(X)

return X
```

## Model Architect

- input_shape = (256, 256, 3) RGB 값을 가진 이미지이다.
- ZeroPadding2D : zero padding 을 통해 residual 의 크기 유지

- Conv2D : 커널 사이즈 = (7, 7), 스트라이드 = (2, 2)
- kernel_initializer : glorot_uniform 디폴트 커널 이니셜라이저 사용.

- BatchNormalization :  배치 정규화를 활용하면 학습 속도를 높여 training 을 가속할 수 있다.
- activation = ‘relu’ 사용
- MaxPooling2D : pooling size (7, 7), 스트라이드 (2,2) 사용.

- res_block 4개 연속으로 추가.

- AveragePooling2D
- Flatten : input data X 에 대해 X.reshape(-1,1) 작업 진행.
- Dense : 6개의 뉴런으로 구성된 하나의 hidden layer 로 구성. 활성화 함수 softmax 사용.

```
input_shape = (256,256,3)

# Input tensor shape
X_input = Input(input_shape)

# Zero-padding
X = ZeroPadding2D((3,3))(X_input)


# 1 - stage
X = Conv2D(64, (7,7), strides= (2,2), name = 'conv1', kernel_initializer= glorot_uniform(seed = 0))(X)
X = BatchNormalization(axis =3, name = 'bn_conv1')(X)
X = Activation('relu')(X)
X = MaxPooling2D((3,3), strides= (2,2))(X)


# 2- stage
X = res_block(X, filter= [64,64,256], stage= 2)

# 3- stage
X = res_block(X, filter= [128,128,512], stage= 3)

# 4- stage
X = res_block(X, filter= [256,256,1024], stage= 4)

# 5- stage
X = res_block(X, filter= [512,512,2048], stage= 5)


# Average Pooling
X = AveragePooling2D((2,2), name = 'Averagea_Pooling')(X)
```

```
# Final layer
X = Flatten()(X)
X = Dense(6, activation = 'softmax', name = 'Dense_final', kernel_initializer= glorot_uniform(seed=0))(X)

#create model
model = Model( inputs= X_input, outputs = X, name = 'Resnet18')

#model summary
model.summary()
```

## Compile Model

- loss = ‘categorical_entropy’ 사용
- optimizer=’adam’ 사용 (미니배치 그레디언트에 적합)
- metrics = accuracy 사용

## train model

예시를 위해 epochs = 5 지정.

```
# compile
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics= ['accuracy'])

# early stopping to exit training if validation loss is not decreasing even after certain epochs (patience)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=15)

# save the best model with lower validation loss
checkpointer = ModelCheckpoint(filepath="weights.hdf5", verbose=1, save_best_only=True)

#fit
history = model.fit_generator(train_generator,
                              steps_per_epoch= train_generator.n // 32,
                              epochs = 5,
                              validation_data= validation_generator,
                              validation_steps= validation_generator.n // 32,
                              callbacks=[checkpointer , earlystopping])
```

## Model Evaluation

```
# evaluate the performance of the model
print('Train loss & accuracy:', model.evaluate(train_generator))
print('\n')
print('Test loss & accuracy:', model.evaluate(test_generator))
```

예제의 결과를 출력하자면 다음과 같다.

```
373/373 [==============================] - 199s 534ms/step - loss:
0.8251 - accuracy: 0.6991
Train loss & accuracy: [0.8250985741615295, 0.6991283893585205]


94/94 [==============================] - 1877s 20s/step - loss:
0.8385 - accuracy: 0.7003
Test loss & accuracy: [0.8385239839553833, 0.7003333568572998]
```

## Make prediction

- 레이블 마다 인덱스를 할당한다. Building : 0 , forest : 1, glacier : 2, mountain : 3, sea : 4, street : 5
- PIL 을 이용해 이미지를 open
- 이미지를 (256, 256) 으로 resize
- 이미지를 이미지 리스트에 append
- 이미지를 배열로 변환시킨다.
- 이미지를 정규화한다.
- 이미지를 4D array 로 reshape
- 이미지 리스트에 대한 예측을 하기 위해 predict() 사용
- 예측된 image label 을 얻기 위해 argmax 사용

```
# assign label names to the corresponding indexes
labels = {0: 'buildings', 1: 'forest', 2: 'glacier', 3:'mountain', 4: 'sea', 5:'street'}

# load images and their predictions
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
# import cv2

prediction = []
original = []
image = []
count = 0


for i in os.listdir('./seg_test'):
  for item in os.listdir(os.path.join('./seg_test', i)):
    # code to open the image
    img= PIL.Image.open(os.path.join('./seg_test', i, item))
    # resizing the image to (256,256)
    img = img.resize((256, 256))
    # appending image to the image list
    image.append(img)
    # converting image to array
    img = np.asarray(img, dtype = np.float32)
    # normalizing the image
    img = img / 255
    # reshaping the image into a 4D array
    img = img.reshape(-1, 256, 256, 3)
    # making prediction of the model
    predict = model.predict(img)
    # getting the index corresponding to the highest value in the prediction
    predict = np.argmax(predict)
    # appending the predicted class to the list
    prediction.append(labels[predict])
    # appending original class to the list
    original.append(i)
```

```
# test accuracy
score = accuracy_score(original, prediction)
print("Test Accuracy : {}".format(score))
```

```
>>>Test Accuracy : 0.7053333333333334
```

## Visualizing model prediction

```
import random

fig = plt.figure(figsize=(100,100))

for i in range(20):
    j = random.randint(0, len(image))
    fig.add_subplot(20, 1, i+1)
    plt.xlabel("Prediction: " + prediction[j] +"   Original: " + original[j])
    plt.imshow(image[j])

fig.tight_layout()
plt.show()
```

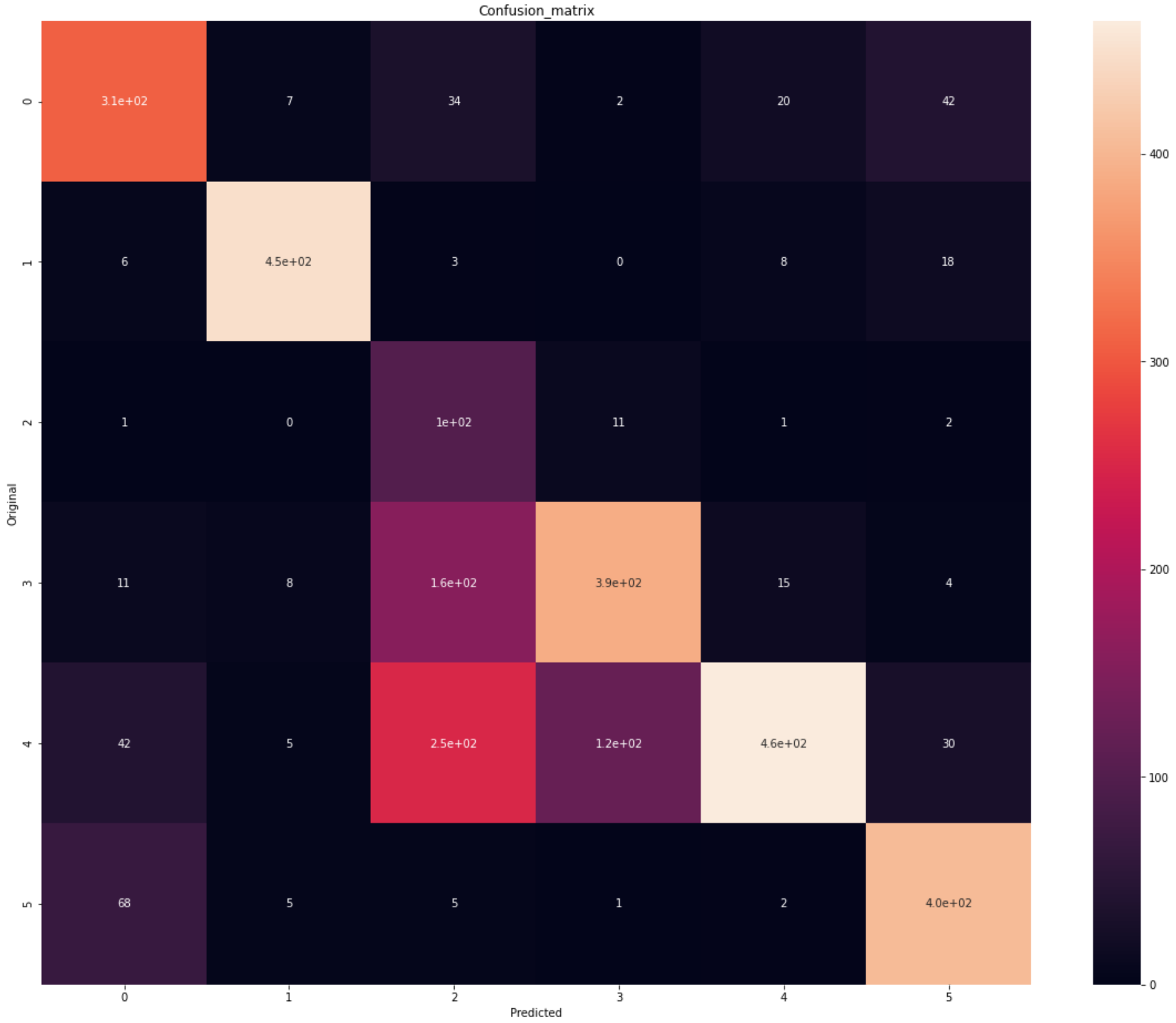
### Classificatin report

```
# print the classiication report
print(classification_report(np.asarray(prediction), np.asarray(original)))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| buildings | 0.71 | 0.75 | 0.73 | 414 |
| forest | 0.95 | 0.93 | 0.94 | 484 |
| glacier | 0.18 | 0.87 | 0.30 | 116 |
| mountain | 0.74 | 0.66 | 0.70 | 585 |
| sea | 0.91 | 0.51 | 0.65 | 915 |
| street | 0.81 | 0.83 | 0.82 | 486 |
| accuracy | | | 0.71 | 3000 |
| macro avg | 0.72 | 0.76 | 0.69 | 3000 |
| weighted avg | 0.81 | 0.71 | 0.73 | 3000 |

### Confusion Matrix

```
# plot the confusion matrix
plt.figure(figsize=(20,20))
cm = confusion_matrix(np.asarray(prediction), np.asarray(original))
ax = plt.subplot()
sns.heatmap(cm, annot=True, ax=ax)

ax.set_xlabel('Predicted')
ax.set_ylabel('Original')
ax.set_title('Confusion_matrix')
```



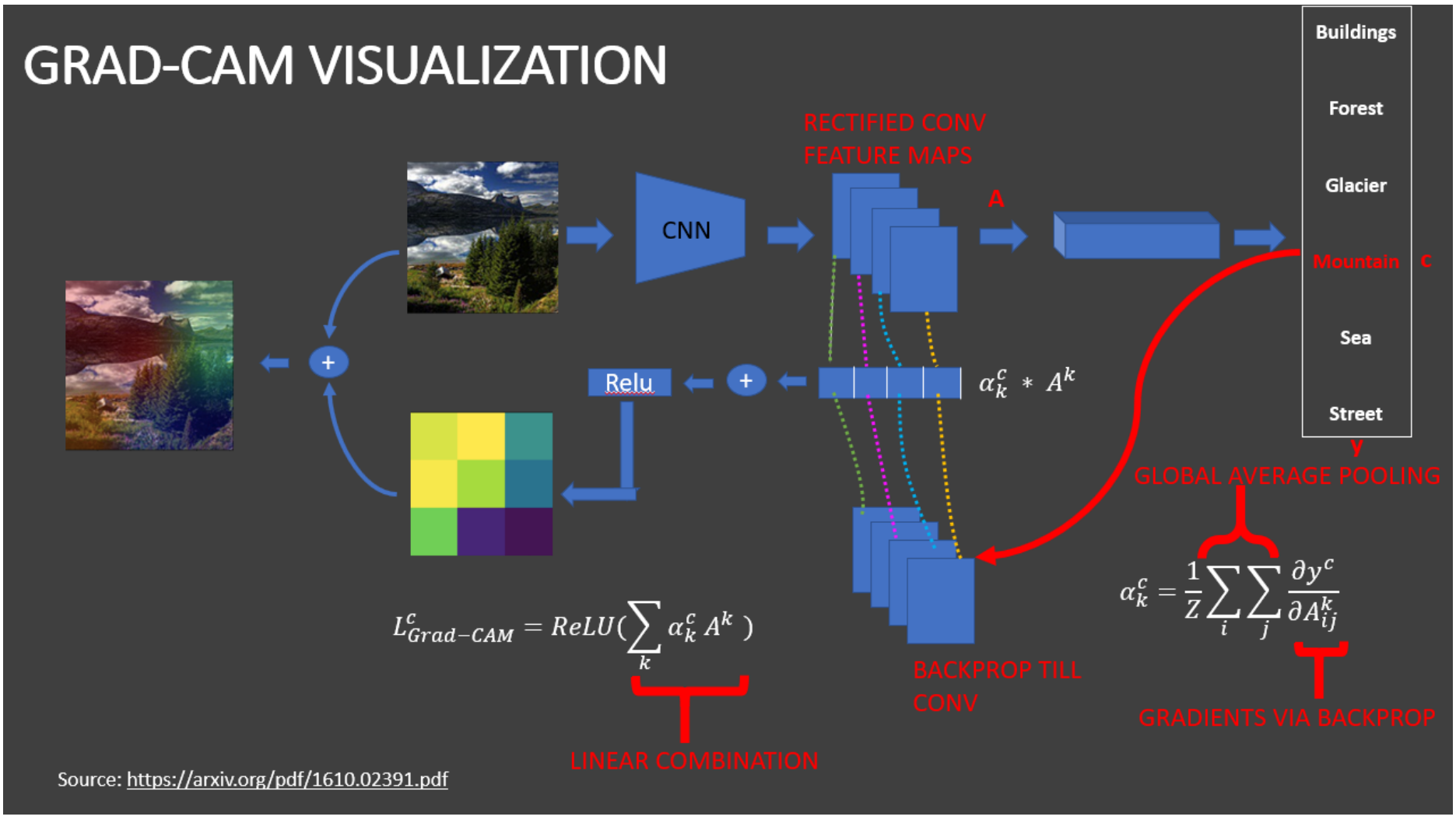

glacier, mountain, sea 가 종종 혼동되는 경향을 보인다.

## Visualize Activation Maps through Grad-CAM

수백만 개의 매개변수가 있는 대규모 모델에서 실제 출력에 기여한 가중치를 연관시키는 것은 어려운 문제이며, 모델 해석 가능성의 이러한 문제는 AI 결정을 신뢰하기 어렵게 만든다.

ResNet-18 모델이 어떻게 결정을 내렸는지 설명하기 위해 Grad-CAM을 사용하여 모델로 예측하는 데 기여한 입력 영역을 시각화할 것이다. Grad-CAM은 (1) 네트워크에서 최종 컨벌루션 레이어를 찾은 다음 (2) 해당 레이어로 흐르는 그라디언트 정보를 검사하여 작동합니다.(Rosebrock, 2020)

그런 다음, 그라디언트를 기반으로 중요도 점수를 계산하여 히트맵을 생성하고 주어진 클래스 레이블을 생성한 이미지 내의 중요한 영역을 강조 표시합니다. 간단히 말해 그라디언트를 가중치(grad-weights)로 사용하여 이미지의 중요한 영역을 강조 표시합니다.




**1단계 : 모델을 통해 이미지를 전달하여 예측**

- 이미지를 array 로 변환
- (256, 256, 3) 에서 (1, 256, 256, 3) 으로 이미지 reshape
- image / 255 로 정규화, 'img_scaled' 라는 값을 얻는다.

**2단계 : 2개의 모델을 만든다. final_conv_model, 그리고 classification_model**

| | | | |
|---|---|---|---|
| res_5_identity_2_c (Conv2D) | (None, 3, 3, 2048) | 1050624 | activation_35[0][0] |
| bn_5_identity_2_c (BatchNormali | (None, 3, 3, 2048) | 8192 | res_5_identity_2_c[0][0] |
| add_11 (Add) | (None, 3, 3, 2048) | 0 | bn_5_identity_2_c[0][0]<br>activation_33[0][0] |
| activation_36 (Activation) | (None, 3, 3, 2048) | 0 | add_11[0][0] |
| Averagea_Pooling (AveragePoolin | (None, 1, 1, 2048) | 0 | activation_36[0][0] |
| flatten (Flatten) | (None, 2048) | 0 | Averagea_Pooling[0][0] |
| Dense_final (Dense) | (None, 6) | 12294 | flatten[0][0] |

Classification Layers

(1) final_conv_model (input 부터 activation 까지)

- get_layer()를 사용해서 원래 모델의 마지막 합성곱 레이어인, res_5_identity_2_c 를 지정하고, final_conv 에 저장한다.
- The new `final_conv_model` consist of our original `ResNet-18` modelinputs `model.inputs` along with the final layer output `final_conv.output`. (이거 해석 어케 하는지 맥락이 안잡힘.)

(2) classification_model (activation 부터 classification 까지)

- 새로운 모델인 classification_model 의 입력은 final_conv 의 출력이다.
- classification_layers 는 예측을 만드는 레이어 이다.
- classification layer 를 통해 Average_Pooling 과 Dense_final 을 얻는 작업을 반복하고 결과를 새로운 모델에 append 시킨다.
- 그리고 나서 final_conv 에 도달할 때까지 역전파를 진행한다.

(3) final_conv_output 을 모니터 하기 위해 GradientTape 을 사용한다.

- GradientTape 은 첫 번째 모델인 final_conv_model 로부터 그레디언트를 되찾아온다.
- GradientTape 은 그레디언트를 기록하고 tape 에 저장한다.
- watch() 함수는 final_conv_output 을 모니터 한다.

(4) argmax 를 사용해 예측에서 최대 확률을 보여주는 인덱스를 찾는다.

- Pass feature map `final_conv_output` generated from the first model `final_conv_model` and feed it through the second model `classification_model` to generate `prediction`.
- Use `argmax` to make prediction on `prediction` and get `predicted_class_value`.

(5) Calculate the gradient using that is used to arrive at that predicted value with respect to feature map activation of the convolution layer:

- `gradient` extracts the desired gradients (the gradient of the loss from the output of the convolutional layer) from `tape` to get `gradient_channels`.

(6) To enhance the filter predicted value, we multiply that filter predicted value with the filter value in the last convolutional layer (Linear Combination).

(7) Perform weighted combination of activation maps and follow it by a ReLU to obtain the heatmap.

(8) Super-impose the feature heatmap onto the original image to see the activation locations in the image.

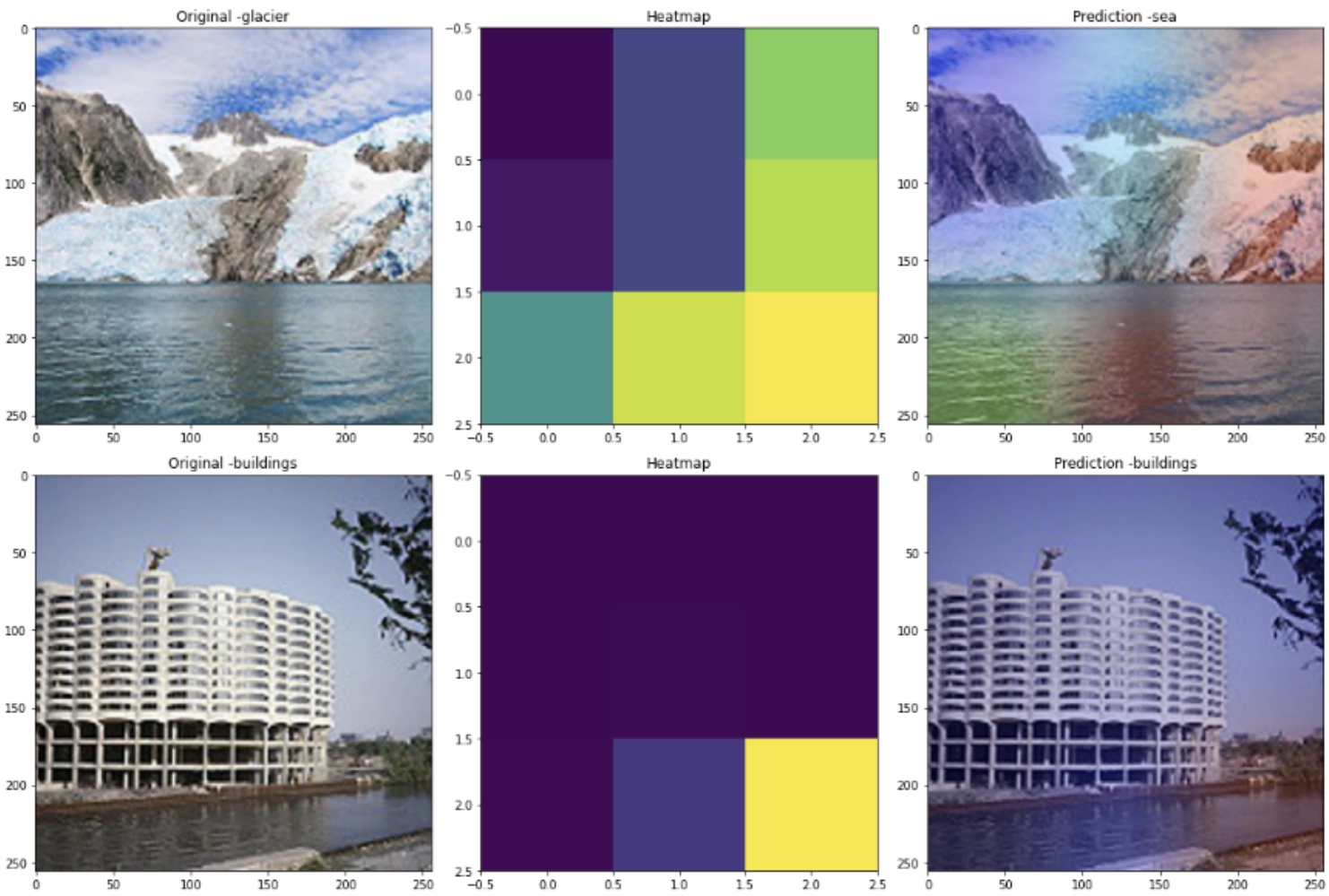
레퍼런스 : ‘Grad-CAM class activation visualization’ 에 관한 케라스 문서

```
def grad_cam(img):

  #convert the image to array of type float32
  img = np.asarray(img, dtype = np.float32)

  #reshape the image from (256,256,3) to (1,256,256,3)
  img = img.reshape(-1, 256, 256, 3)
#scale
img_scaled = img / 255


#name of the average pooling layer and dense final (you can see these names in the model summary)
classification_layers = ["Averagea_Pooling", "Dense_final"]

#last convolutional layer in the model
final_conv = model.get_layer("res_5_identity_2_c")
# create a model with original model inputs and the last conv_layer as the output
final_conv_model = keras.Model(model.inputs, final_conv.output)


#then we create the input for classification layer, which is the output of last conv layer
#in our case, output produced by the conv layer is of the shape (1,3,3,2048)
#since the classification input needs the features as input, we ignore the batch dimension
classification_input = keras.Input(shape = final_conv.output.shape[1:])


#iterate through the classification layers, to get the final layer and
#then append the layer as the output layer to the classification model.
temp = classification_input
for layer in classification_layers:
    temp = model.get_layer(layer)(temp)
classification_model = keras.Model(classification_input, temp)


#use gradient tape to monitor the 'final_conv_output' to retrive the gradients corresponding to the predicted class
with tf.GradientTape() as tape:
    #pass the image through the base model and get the feature map
    final_conv_output = final_conv_model(img_scaled)
```

```
    #asssign gradient tape to monitor the conv_output
    tape.watch(final_conv_output)

    #pass the feature map through the classification model and use argmax to get the
    #index of the predicted class and then use the index to get the value produced by final layer for that class
    prediction = classification_model(final_conv_output)
    predicted_class = tf.argmax(prediction[0][0][0])
    predicted_class_value = prediction[:,:,:,predicted_class]


#get the gradient corresponding to the predicted class based on feature map
#which is of shape (1,3,3,2048)
gradient = tape.gradient(predicted_class_value, final_conv_output)

#since we need the filter values (2048), we reduce the other dimensions,
#which would result in a shape of (2048,)
gradient_channels = tf.reduce_mean(gradient, axis=(0, 1, 2))

#convert the feature map produced by last conv layer(1,6,6,1536) to (6,6,1536)
final_conv_output = final_conv_output.numpy()[0]

gradient_channels = gradient_channels.numpy()

#multiply the filters in the feature map produced by final conv layer by the
#filter values that are used to get the predicted class. By doing this we inrease the
#value of areas that helped in making the prediction and lower the vlaue of areas, that
#did not contribute towards the final prediction
for i in range(gradient_channels.shape[-1]):
    final_conv_output[:, :, i] *= gradient_channels[i]

  #take the mean accross the channels to get the feature map
  heatmap = np.mean(final_conv_output, axis=-1)
  #normalizie the heat map between 0 and 1, to visualize it
  heatmap_normalized = np.maximum(heatmap, 0) / np.max(heatmap)
  #rescale and convert the type to int
  heatmap = np.uint8(255 * heatmap_normalized )
  #create the colormap
  color_map = plt.cm.get_cmap('jet')
  #get only the rb features from the heatmap
  color_map = color_map(np.arange(256))[:, :3]
  #apply color map to heatmap
  heatmap = color_map[heatmap]
  #convert the array to image, resize the image and then convert to array
  heatmap = keras.preprocessing.image.array_to_img(heatmap)
  heatmap = heatmap.resize((256, 256))
  heatmap = np.asarray(heatmap, dtype = np.float32)

  #add the heatmap on top of the original image
  final_img = heatmap * 0.4 + img[0]
  final_img = keras.preprocessing.image.array_to_img(final_img)

  return final_img, heatmap_normalized
```

6 가지 샘플에 대해 확인.

```
# visualize the images in the dataset
import random
fig, axs = plt.subplots(6,3, figsize = (16,32))
count = 0
for _ in range(6):
  i = random.randint(0, len(image))
  gradcam, heatmap = grad_cam(image[i])
  axs[count][0].title.set_text("Original -" + original[i])
  axs[count][0].imshow(image[i])
  axs[count][1].title.set_text("Heatmap")
  axs[count][1].imshow(heatmap)
  axs[count][2].title.set_text("Prediction -" + prediction[i])
  axs[count][2].imshow(gradcam)
  count += 1

fig.tight_layout()
```




노란색-녹색 영역은 모델이 주목하는 영역이다. 첫 번째 사진의 경우 모델은 바다 영역을 보고 있다. 그래서 original label 인 ‘glacier’ 대신 ‘sea’ 라고 분류했다.

---

# 구현 파일

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4bf3e8fe-b674-4df6-9d52-96a926c2e73b/XAI_Grad_CAM.ipynb

구글 colab 에서 작성했다. 캐글 데이터 사용.