

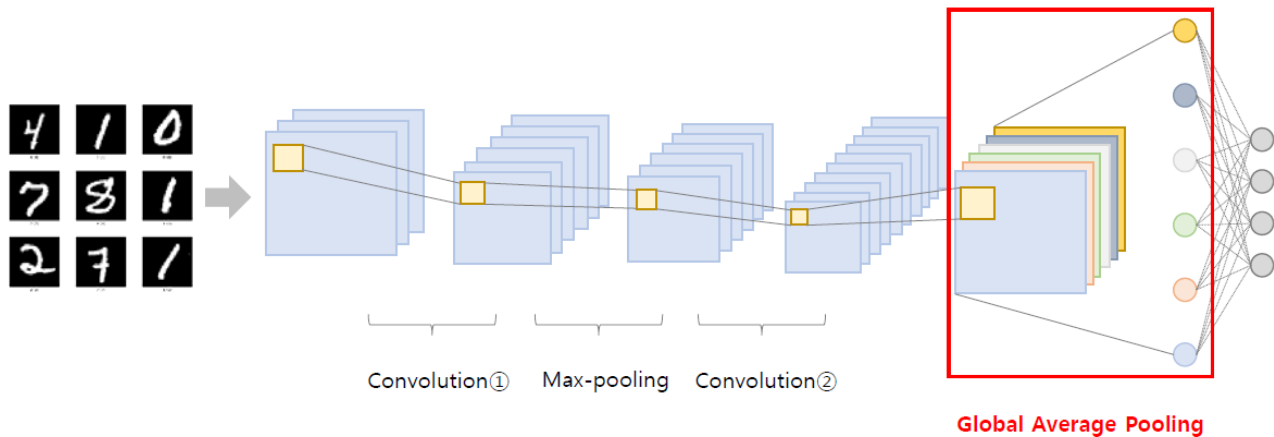
Grad-CAM

Prof. Hyunseok Oh

School of Mechanical Engineering
Gwangju Institute of Science and Technology

Limitation of CAM-Based XAI

- CAM은 Global Average Pooling(GAP)을 반드시 사용해야만 함.
 - 만약 CNN 마지막 Conv Layer의 Feature map에 대해 Flattening을 사용했다면, GAP로 대체해야 함.
 - 대체 후 FC Layer 부분의 Weight와 Bias를 Fine tuning을 통해 재학습 해야 함.
- 마지막 Conv Layer에 대해서만 Visualization 가능
 - 그러나, 모든 Conv Layer에 대해 시각화가 중요하거나 또는 필요할 수도 있음.



Grad-CAM

- Selvaraju et al. “Grad-CAM: Visual explanations from deep networks via gradient-based localization”, ICCV, 2017.

Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

Ramprasaath R. Selvaraju¹ Michael Cogswell² Abhishek Das² Ramakrishna Vedantam¹

Devi Parikh² Dhruv Batra²

¹Virginia Tech, ²Georgia Institute of Technology

{ram21, vrama91}@vt.edu

{cogswell, abhshkdz, vrama91, parikh, dbatra}@gatech.edu

Abstract

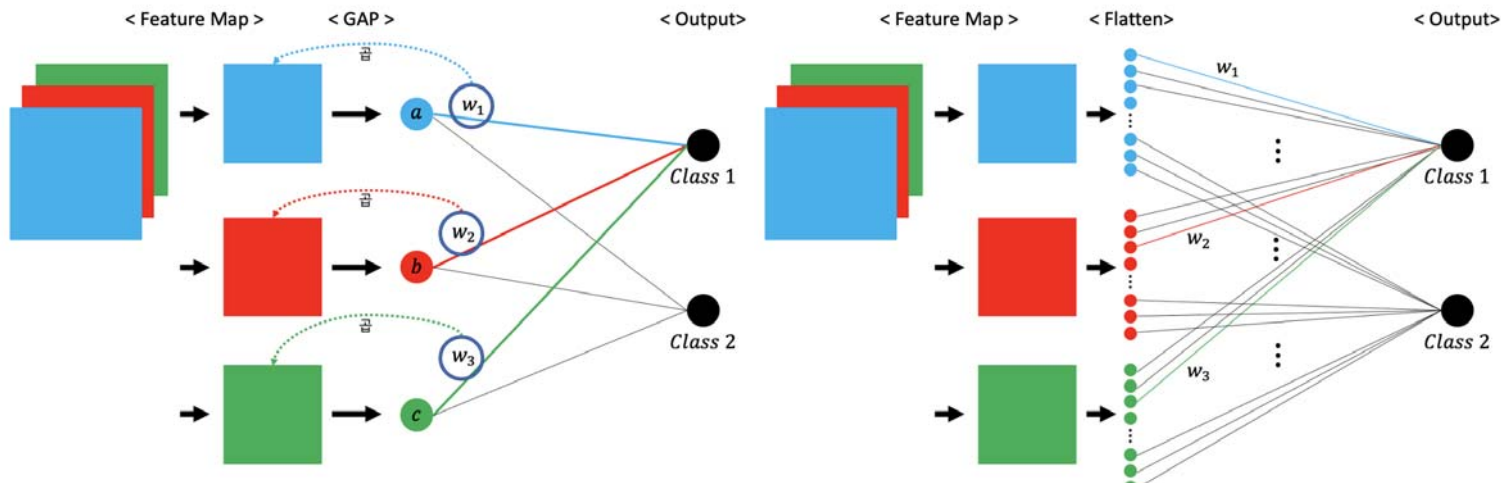
We propose a technique for producing ‘visual explanations’ for decisions from a large class of Convolutional Neural Network (CNN)-based models, making them more transparent. Our approach – Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept (say logits for ‘dog’ or even a caption), flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept. Unlike previous approaches, Grad-CAM is applicable to a wide variety of CNN model-families: (1) CNNs with fully-connected layers (e.g. VGG), (2) CNNs used for structured outputs (e.g. captioning), (3) CNNs used in

1. Introduction

Convolutional Neural Networks (CNNs) and other deep networks have enabled unprecedented breakthroughs in a variety of computer vision tasks, from image classification [27, 18] to object detection [16], semantic segmentation [31], image captioning [47, 7, 13, 23], and more recently, visual question answering [3, 15, 36, 41]. While these deep neural networks enable superior performance, their lack of decomposability into *intuitive and understandable* components makes them hard to interpret [30]. Consequently, when today’s intelligent systems fail, they fail spectacularly disgracefully, without warning or explanation, leaving a user staring at an incoherent output, wondering why.

Grad-CAM

- Gradient-weighted CAM(Grad-CAM)이란
 - GAP을 사용하지 않으므로, 기존 CNN 구조를 변형하지 않고 그대로 사용 가능함
 - CNN 구조에서 Gradient(특정 input이 특정 output에 주는 영향력의 크기)값을 계산하여, 판단의 근거를 Visualization하는 방법
 - 판단에 사용된 정보를 표현하여 실패한 모델의 진단 또한 가능
- CAM과 Grad-CAM 구조

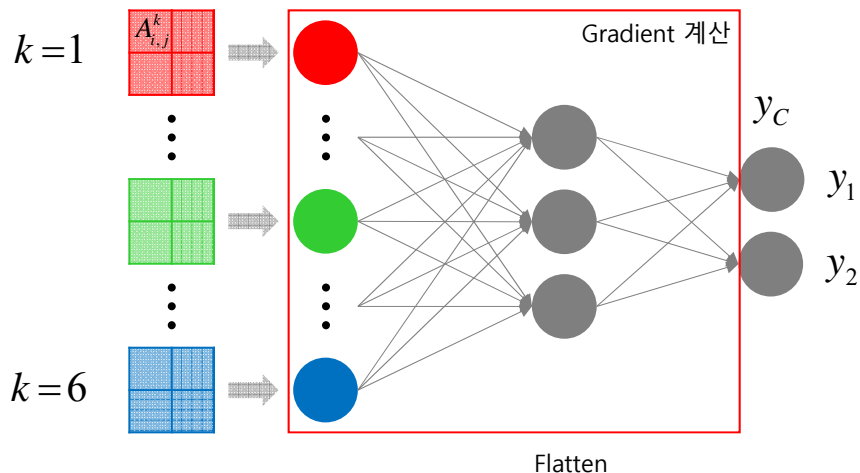


Grad-CAM 수학적 정의

- 분류에 대한 Grad-CAM 점수 계산 수식
 - CAM과 비교하였을 때, 학습을 통해 산출되던 가중치(Weight)를 Gradient를 통해 산출한 α 를 활용
 - 산출된 가중치 α 와 특징 맵의 각 요소를 활용하여 점수 계산

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

$$L_{Grad-CAM}^c = ReLU \sum_k \alpha_k^c A^k$$



c : 예측 클래스

α_k^c : c 클래스를 예측하는 k 번째 특징 맵 가중치

A^k : k 번째 특징 맵

$A_{i,j}$: 특징 맵 내 i, j 위치

Z : 특징 맵 별 요소의 총 갯수

y_c : Softmax 층을 통과하기 전 클래스 별 결과값

CAM 점수 계산 식

$$L_{CAM}^c = \sum_k w_k^c A^k$$

가중치를 위한 Gradient 계산

- 예측값에 대한 Gradient의 의미와 계산 방법
 - Softmax 층을 통과하기 전 클래스 결과값(y_c)에 대한 Convolution layer의 특성 맵인 A_k 의 영향도
 - 역전파를 활용하여 특성맵의 모든 (i,j) 의 뉴런에 대해 계산 수행
 - 텐서플로우에 내장된 GradientTape.gradient(예측, Layer)를 사용하여 Layer에서 예측된 값에 대한 Gradient를 계산

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

$$L_{Grad-CAM}^c = ReLU \sum_k \alpha_k^c A^k$$

c : 예측 클래스

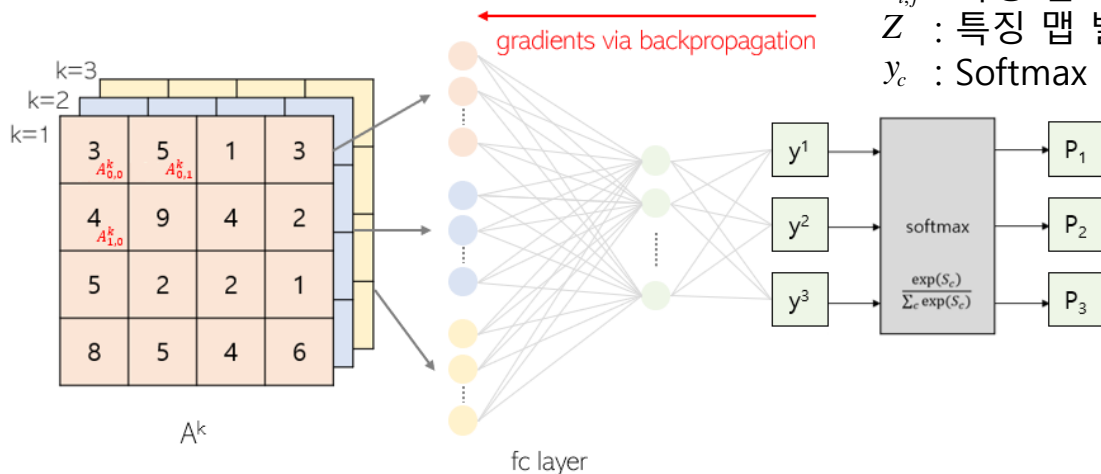
α_k^c : c 클래스를 예측하는 k 번째 특징 맵 가중치

A^k : k 번째 특징 맵

$A_{i,j}^k$: 특징 맵 내 i, j 위치

Z : 특징 맵 별 요소의 총 갯수

y_c : Softmax 층을 통과하기 전 클래스 별 결과값



특성 맵 Global Average Pooling

- 계산된 특성 맵의 가중치를 계산하기 위해 각 A_k 의 평균 값 사용
 - z 는 특징 맵의 행렬 갯수를 의미
 - 아래 그림의 경우 4x4행렬로 $z=16$ 으로 나누어 Global Average Pooling 수행
 - 계산된 α 는 예측한 클래스에 대한 피쳐 맵의 중요도 정보를 포함

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

$$L_{Grad-CAM}^c = ReLU \sum_k \alpha_k^c A^k$$

c : 예측 클래스

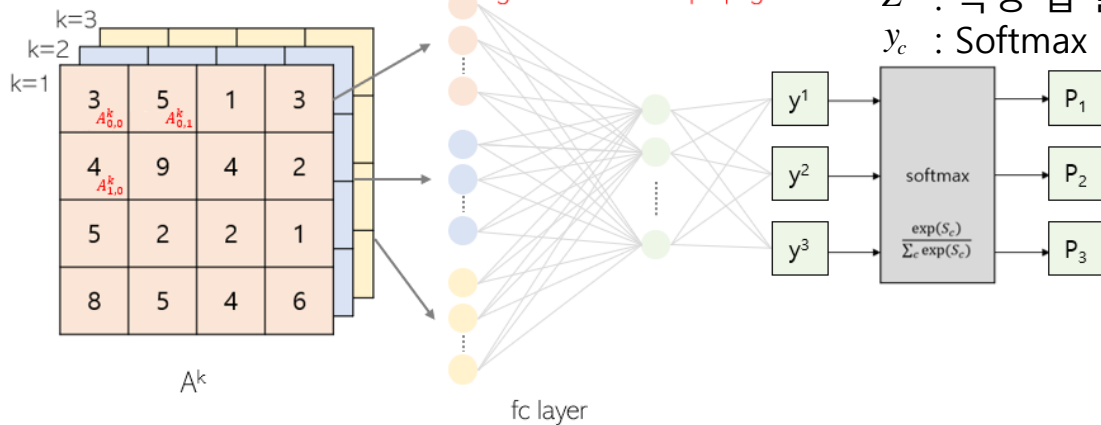
α_k^c : C 클래스를 예측하는 K번째 특징 맵 가중치

A^k : k번째 특징 맵

$A_{i,j}$: 특징 맵 내 ij 위치

Z : 특징 맵 별 요소의 숫자

y_c : Softmax 층을 통과하기 전 클래스 별 결과값

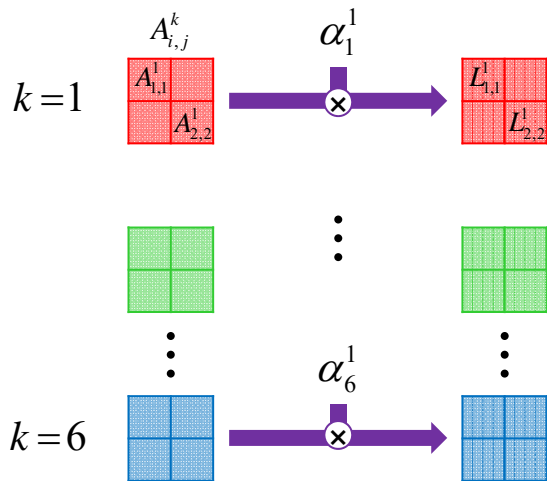


Grad-CAM 점수 계산

- 가중치 α 와 피쳐 맵 A_k 의 선형 결합 연산을 통해 Grad-CAM 점수 계산
 - CAM 방법과 동일하게 피쳐 맵의 각 행렬값은 가중치 α 를 곱하여 연산
 - 클래스를 분류하는 데 있어 긍정적인 영향을 나타내는 특성인자만 표현하기 위해 ReLU 함수를 적용하여 점수 산출

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

$$L_{Grad-CAM}^c = ReLU \sum_k \alpha_k^c A^k$$



c : 예측 클래스

α_k^c : C 클래스를 예측하는 K번째 특징 맵 가중치

A^k : k번째 특징 맵

$A_{i,j}$: 특징 맵 내 ij 위치

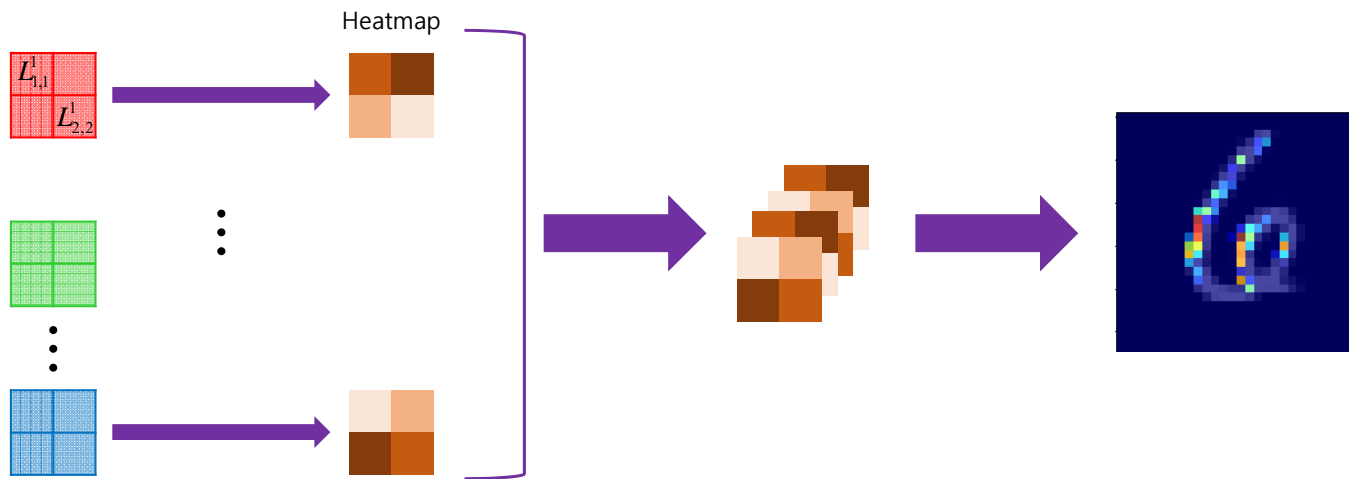
Z : 특징 맵 별 요소의 숫자

y_c : Softmax 층을 통과하기 전 클래스 별 결과값

시각화

Grad-CAM 점수 시각화

- 계산된 점수를 바탕으로 클래스를 분류하는데 가장 주요한 부분을 시각화
 - 계산된 Grad-CAM 점수를 정규화하여 Heatmap으로 표현
 - 아래 예시를 통해 6을 예측하는 데 있어 주요하게 사용된 부분 확인



CAM and Grad-CAM

- CAM

- GAP Layer 학습을 통한 가중치 w 를 활용
- 마지막 Layer에 대해서만 적용 가능함

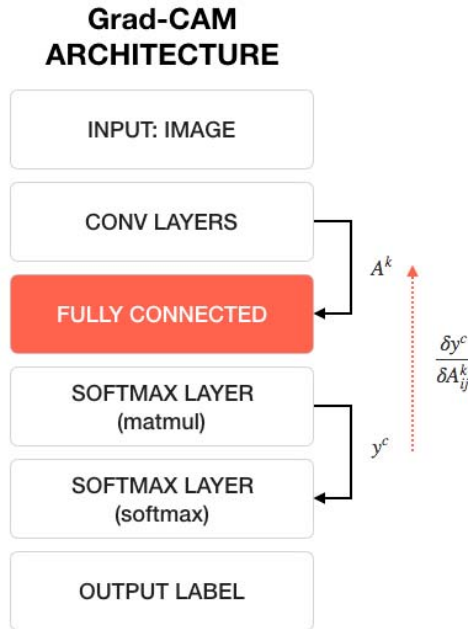
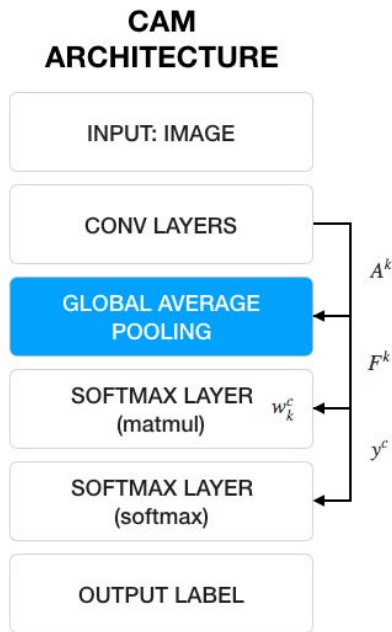
$$L_{CAM}^c = \sum_k w_k^c A^k$$

- Grad-CAM

- 모델 변경 없이 Gradient값을 활용하여 가중치 α 산출
- 모든 Layer에 대해 적용 가능함

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

$$L_{Grad-CAM}^c = ReLU \sum_k \alpha_k^c A^k$$



CAM and Grad-CAM

- Grad-CAM은 CAM의 일반화 증명

- GAP Layer를 통해 출력된 값 F^k 로 정의하면, CAM은 학습된 가중치와 이를 곱하여 CAM 점수 산출

$$F^k = \frac{1}{Z} \sum_i \sum_j A_{i,j}^k \rightarrow y_c = \sum_k w_k^c F^k$$

- 산출된 점수에 대한 GAP Layer의 Gradient를 계산
- 앞서 정의한 F^k 를 A^k 로 편미분을 수행하면 $1/Z$ 만 남으며, CAM 점수를 F^k 로 편미분 시 w 만 남게 됨

$$\frac{\partial L_{cam}^c}{\partial F^k} = \frac{\partial L_{cam}^c}{\partial A_{i,j}^k} \frac{\partial A_{i,j}^k}{\partial F^k} \text{ (by chain rule)} \rightarrow w_c^k = \frac{\partial y_c}{\partial A_{i,j}^k} \times Z$$

- 양측에 모든 요소에 대한 합을 산출시 다음과 같이 정리됨

$$\sum_i \sum_j w_k^c = \sum_i \sum_j \frac{\partial L_{cam}^c}{\partial A_{i,j}^k} \times Z \text{ rewritten as } w_k^c = \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k} (\sum_i \sum_j 1 = Z)$$

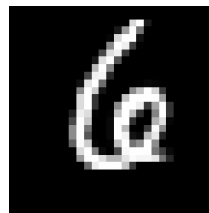
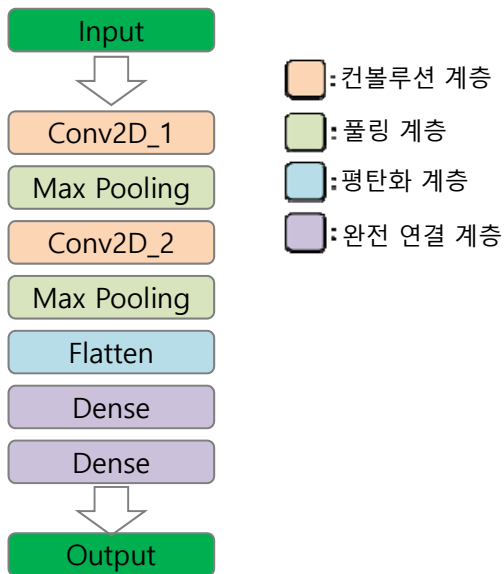
Grad-CAM
점수 계산 식

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

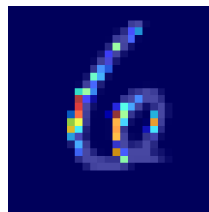
- 즉 CAM의 가중치에 대해 일반화 시킨 값은 Grad CAM에서 수행된 가중치와 동일함

Grad-CAM 구현 결과

- MNIST 데이터 셋에 대한 코드 구현 결과
 - 기존 CNN 구조에 대해 모든 합성곱 층별 결과를 가시화 가능



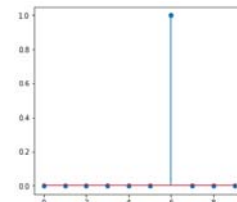
Input



Conv2D_1



Conv2D_2

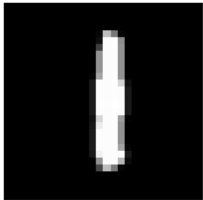
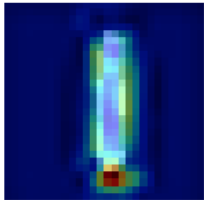
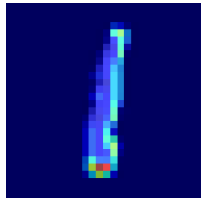
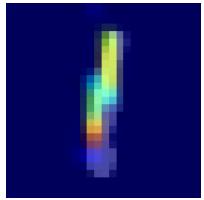

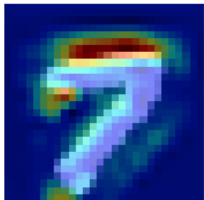

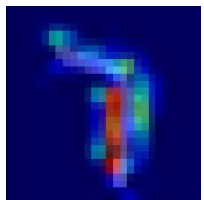

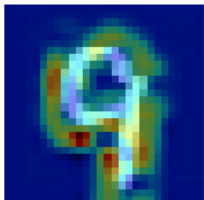
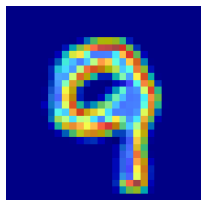
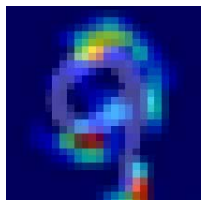


Output

판단 근거 시각화

- 유사한 이미지에 대해 가중치를 시각화하여 판단 근거 확인이 가능함

- 1과 7, 9의 구분은 가로 방향의 특질에 대해서 판단함
- 7, 9의 구분은 가로 방향의 갯수가 다름을 보임
- 유사한 모양에 대해 판단 근거를 확인 가능함
- Grad-CAM을 통해 각 층별 판단 근거까지 확인 가능

	Input	CAM	Grad-CAM Layer 1	Grad-CAM Layer 2
Class '1'				
Class '7'				
Class '9'				

Demo

MNIST Example

- 라이브러리 호출

```
1 import os
2 import tensorflow as tf
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from tensorflow import keras
6 from tensorflow.keras import models
7 from tensorflow.keras import backend as K
8 from tensorflow.keras.preprocessing import image
9 from PIL import Image
10 import matplotlib.cm as cm
```

executed in 1.15s, finished 23:11:01 2021-12-16

- MNIST 데이터 로드

```
1 mnist = tf.keras.datasets.mnist
2
3 (train_x, train_y), (test_x, test_y) = mnist.load_data()
4
5 train_x, test_x = train_x/255.0, test_x/255.0
6
7 train_x = train_x.reshape((train_x.shape[0], 28, 28, 1))
8 test_x = test_x.reshape((test_x.shape[0], 28, 28, 1))
9
10 n_train = train_x.shape[0]
11 n_test = test_x.shape[0]
12
13 print ("The number of training images : {}, shape : {}".format(n_train, train_x.shape))
14 print ("The number of testing images : {}, shape : {}".format(n_test, test_x.shape))
```

executed in 260ms, finished 16:10:50 2021-12-16

MNIST Example

• CNN 아키텍처 구성

```

1 model = tf.keras.models.Sequential([
2     tf.keras.layers.Conv2D(filters = 32,
3                             kernel_size = (3, 3),
4                             activation = 'relu',
5                             padding = 'SAME',
6                             input_shape = (28, 28, 1)),
7     tf.keras.layers.MaxPool2D((2, 2)),
8
9     tf.keras.layers.Conv2D(filters = 64,
10                             kernel_size = (3, 3),
11                             activation = 'relu',
12                             padding = 'SAME',
13                             input_shape = (14, 14, 32)),
14
15     tf.keras.layers.MaxPool2D((2, 2)), —————→ Max pooling layer
16
17     tf.keras.layers.Flatten(),
18     tf.keras.layers.Dense(units = 32, activation = 'relu'),
19     tf.keras.layers.Dense(units = 10, activation = 'softmax')
20 ])
  
```

executed in 53ms, finished 14:40:30 2021-12-30

Input



Conv2D_1

Max Pooling

Conv2D_2

Max Pooling

Flatten

Dense

Dense



Output

-  : 컨볼루션 계층
-  : 풀링 계층
-  : 평탄화 계층
-  : 완전 연결 계층

MNIST Example

- 인공지능 모델 세팅 및 훈련

```
1 model.compile(optimizer = 'adam',  
2               loss = 'sparse_categorical_crossentropy',  
3               metrics = 'accuracy')
```

executed in 11ms, finished 23:11:01 2021-12-16

```
1 model.fit(train_x, train_y, batch_size = 50)
```

executed in 6.33s, finished 23:11:08 2021-12-16

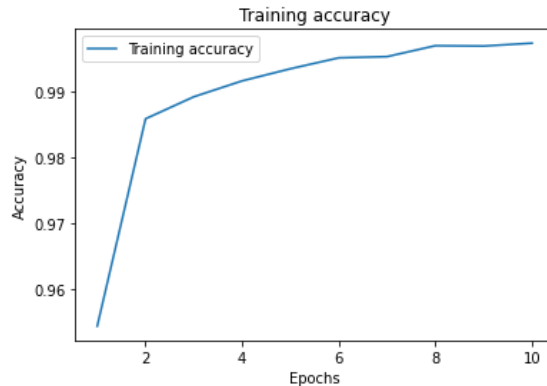
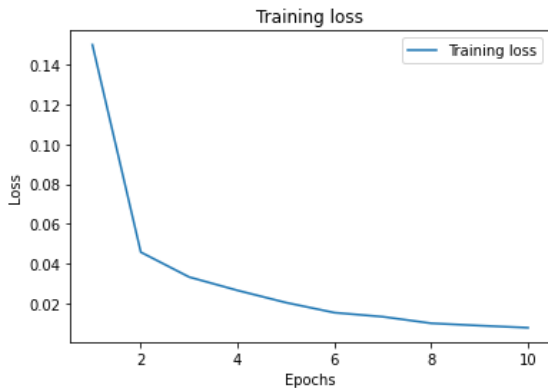
- 모델 평가

```
1 test_img = test_x[[5555]]  
2  
3 predict = model.predict(test_img)  
4 mypred = np.argmax(predict, axis = 1)  
5  
6 plt.figure(figsize = (12, 5))  
7  
8 plt.subplot(1, 2, 1)  
9 plt.imshow(test_img.reshape(28, 28), 'gray')  
10 plt.axis('off')  
11 plt.subplot(1, 2, 2)  
12 plt.stem(predict[0])  
13 plt.show()  
14  
15 print('Prediction : {}'.format(mypred[0]))
```

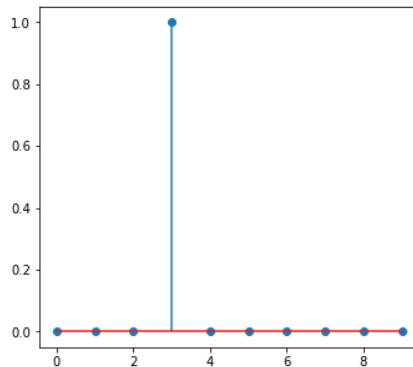
executed in 131ms, finished 23:53:37 2021-12-16

MNIST Example

- 학습 결과 및 모델 평가 예시



Prediction : 3



MNIST Example

- 학습 모델 요약

1	model.summary()
executed in 13ms, finished 23:30:32 2021-12-16	
Model: "sequential"	

Layer (type)	Output Shape Param #

conv2d (Conv2D)	(None, 28, 28, 32) 320

max_pooling2d (MaxPooling2D)	(None, 14, 14, 32) 0

conv2d_1 (Conv2D)	(None, 14, 14, 64) 18496

max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64) 0

conv2d_2 (Conv2D)	(None, 7, 7, 64) 36928

max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64) 0

flatten (Flatten)	(None, 576) 0

dense (Dense)	(None, 128) 73856

dense_1 (Dense)	(None, 10) 1290
=====	
Total params: 130,890	
Trainable params: 130,890	
Non-trainable params: 0	

MNIST Example

• Gradient를 통한 중요도 계산 함수 구성

```

1 def make_gradcam_heatmap(img_array, model, conv_layer_name, pred_index=None):
2     # 입력에 따른 피쳐 맵 모델 생성
3     grad_model = tf.keras.models.Model(
4         [model.inputs], [model.get_layer(conv_layer_name).output, model.output]
5     )
6     # 출력에 따른 Gradient 계산 함수
7     with tf.GradientTape() as tape:
8         conv_layer_output, preds = grad_model(img_array)
9         if pred_index is None:
10             pred_index = tf.argmax(preds[0])
11             class_channel = preds[:, pred_index]
12
13     # 예측된 값에 대한 Gradient 호출
14     grads = tape.gradient(class_channel, conv_layer_output)
15
16     # 각 피쳐 맵에 대한 가중치 산출
17     pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
18
19     # 계산된 가중치와 해당 레이어를 곱하여 얼마나 중요도를 가지는지 히트맵 계산
20     conv_layer_output = conv_layer_output[0]
21     heatmap = conv_layer_output @ pooled_grads[..., tf.newaxis]
22     heatmap = tf.squeeze(heatmap)
23
24     # 시각화를 위해 0-1 범위로 정규화 및 ReLU 수행
25     heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
26     return heatmap.numpy()
27
  
```

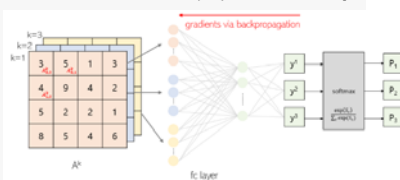
executed in 12ms, finished 14:43:39 2021-12-27

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

출력값에 대한 Gradient 계산 및 호출

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_{i,j}^k}$$

Gradient 합 및 평균 계산 후 가중치 산출



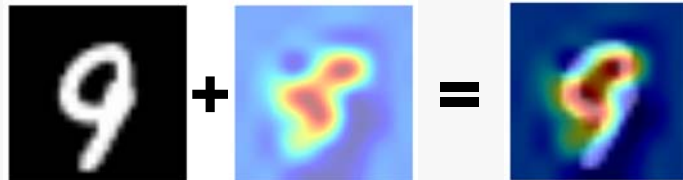
$$L_{Grad-CAM}^c = ReLU \sum_k \alpha_k^c A^k$$

중요도와 Feature Map을 곱하여 Heatmap 계산

MNIST Example

- Heatmap 결과와 Input Data 가시화 함수

```
1 def save_and_display_gradcam(img_input, heatmap, cam_path = "cam.jpg", alpha=0.01):
2     # Load the original image
3     img = img_input.reshape(28,28)
4     img = keras.preprocessing.image.img_to_array(img)
5
6     # 정규화된 Heatmap 이미지를 0-255 범위로 변환
7     heatmap = np.uint8(255 * heatmap)
8     jet = cm.get_cmap("jet")
9
10    # 계산된 값을 RGB 값으로 변경 및 이미지 변환
11    jet_colors = jet(np.arange(256))[:, :3]
12    jet_heatmap = jet_colors[heatmap]
13    jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
14    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
15    jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)
16
17    # 계산한 Heatmap과 입력 이미지 곱함
18    # alpha 값을 통해 입력 이미지의 투명도 계산
19    superimposed_img = jet_heatmap * alpha + img
20    superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)
21
22    # 생성된 이미지 저장
23    superimposed_img.save(cam_path)
24
25    # 가시화
26    plt.imshow(superimposed_img)
```



executed in 10ms, finished 15:49:45 2021-12-27

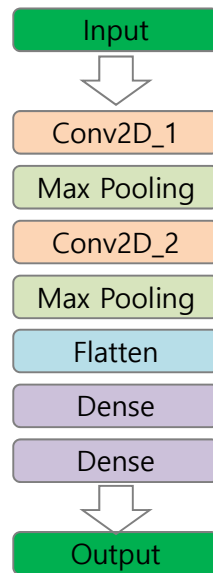
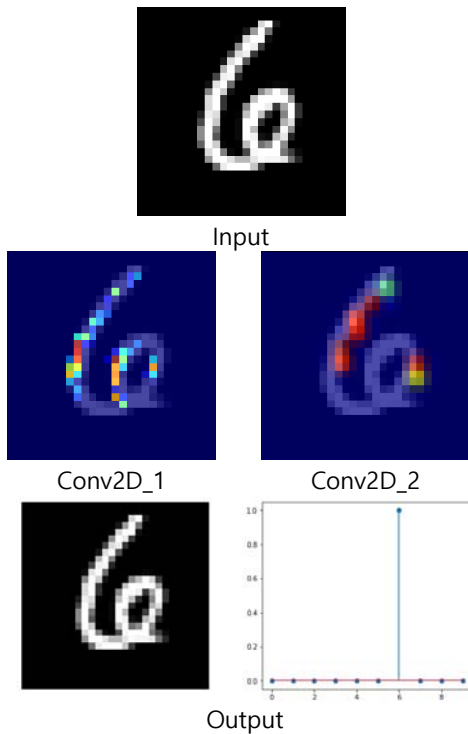
MNIST Example

• 결과

```

1 input_image = test_x[[123]]
2 make_heatmap = make_gradcam_heatmap(input_image, model, 'conv2d_2')
3 save_and_display_gradcam(input_image, make_heatmap)

```



- : 컨볼루션 계층
- : 풀링 계층
- : 평탄화 계층
- : 완전 연결 계층

Grad CAM Using Pretrained Model

- VGG16 모델 및 입력에 쓰일 이미지 불러오기

```
1 model_builder = tf.keras.applications.vgg16.VGG16
2
3 preprocess_input = keras.applications.vgg16.preprocess_input
4 decode_predictions = keras.applications.vgg16.decode_predictions
5
6 # 이미지 경로
7 image_ = load_img('./images/cat.1.jpg', target_size=(224, 224))
8 plt.figure(figsize=(10,10))
9 plt.imshow(image_)
10
```

executed in 223ms, finished 14:08:15 2022-01-18



Grad CAM Using Pretrained Model

- 학습된 Imagenet의 가중치 확인 및 모델 구성 확인

1	model = model_builder(weights="imagenet")	Layer (type)	Output Shape	Param #
2	model.summary()	input_2 (InputLayer)	[(None, 224, 224, 3)]	0
executed in 1.30s, finished 14:05:32 2021-12-30		block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
		block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
		block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
		block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
		block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
		block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
		block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
		block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
		block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
		block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
		block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
		block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
		block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
		block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
		block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
		block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
		block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
		block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
		flatten (Flatten)	(None, 25088)	0
		fc1 (Dense)	(None, 4096)	102764544
		fc2 (Dense)	(None, 4096)	16781312
		predictions (Dense)	(None, 1000)	4097000
		Total params: 138,357,544		
		Trainable params: 138,357,544		
		Non-trainable params: 0		

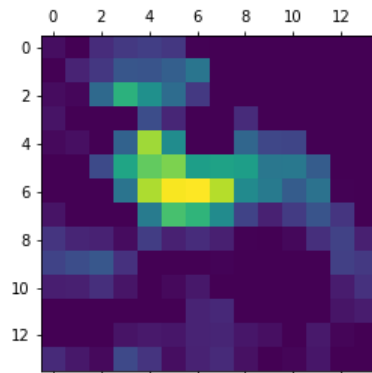
Grad CAM Using Pretrained Model

- 원하는 Layer를 선정하여 Heatmap 산출

```
1 last_conv_layer_name = "block5_conv3"
2
3 # Prepare image
4 img_array = preprocess_input(get_img_array(img_path, size=img_size))
5
6 # Make model
7 model = model_builder(weights="imagenet")
8
9
10 # Print what the top predicted class is
11 preds = model.predict(img_array)
12 print("Predicted:", decode_predictions(preds, top=1)[0])
13
14 # Generate class activation heatmap
15 heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
16
17 # Display heatmap
18 plt.matshow(heatmap)
19 plt.show()
```

executed in 1.64s, finished 14:07:30 2021-12-30

Predicted: [('n02123045', 'tabby', 0.09234099)]



Grad CAM Using Pretrained Model

- 얼룩고양이(Tabby) 판단한 근거 가시화

