9 - d 컨브넷 필터 시각화

필터 시각화

빈 이미지를 넣어서 필터의 값을 확인
다른 점은 loss function을 늘리는 방식을 채택 해야함 (경사 상승법)
손실 값을 늘려서 패턴을 보이도록 유도한다. → 필터를 보기 위해서

page29

벽돌이 주어졌을 때 반응 하는 필터에 빈 이미지를 주어도 벽돌로 반응하게 하기 위함?

즉, 쌩으로 있는 필터가 어느 이미지에 반응하는지 확인하기 위한 작업?
이를 위해서 필터의 응답이 보기 위해 필터의 응답을 최대화하는 경사상승법을 적용한다.
필터가 최대로 응답하는 이미지가 최종적으로 나온다.

page 35

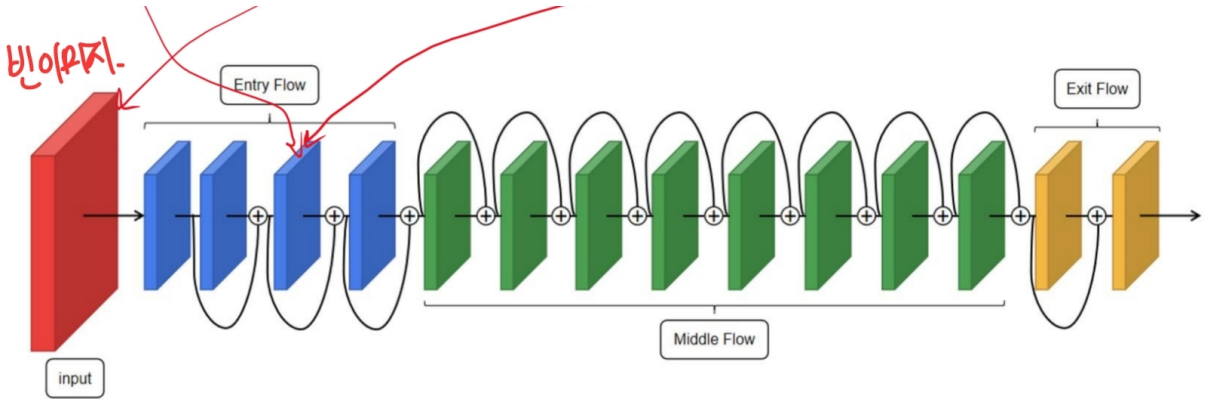
필터에 대한 활성화 평균 값을 손실로 보고(실제 손실은 아님) 이 평균 값을 미분 값을 사용해 점점 늘려가는 방식

page 36

특성 추출 모델 만들기

```
layer_name = "block3_sepconv1" # Xception 합성곱 기반에 있는 다른 층의 이름
layer = model.get_layer(name=layer_name) # 관심 대상인 층의 객체
feature_extractor = keras.Model(inputs=model.input, outputs=layer.output)
```

model.input과 layer.output을 사용해서 입력 이미지가 주어졌을 때 해당 층의 출력을 반환하는 모델



데이터 전처리 (전처리 한번 필요하다)

```
activation = feature_extractor(
        keras.applications.xception.preprocess_input(img_tensor)
)
```

이미지를 [-1,1] 범위로 스케일링하고 RGB 채널을 BGR 채널 순서로 변환하는 전처리 과정

손실 함수: 특성 추출 모델을 사용해서 입력 이미지가 층의 필터를 얼마나 활성화 하는지

경사 상승법: 필터가 어느 이미지에 반응하는지 확인하기 위한 작업, 이를 위해서 필터의 응답이 보기 위해 필터의 응답을 최대화하는 경사상승법을 적용한다.

```
import tensorflow as tf

def compute_loss(image, filter_index):
        activation = feature_extractor(image)
```

```
        filter_activation = activation[:, 2:-2, 2:-2, filter_index] # 1.
        return tf.reduce_mean(filter_activation)                    # 2.
```

1. 활성화 테두리를 따라 두 픽셀을 제외합니다. 경계에 나타나는 부수 효과를 제외한다.
2. 이 필터에 대한 활성화 값의 평균을 반환한다.

미분값을 더하면 경사 상승법이된다.

---

predict - 마지막 레이어에 대한 predict 값이 나온다.

model(x) - 각각의 모든 레이어 별로 predict하도록 만듦 (1~9까지의 각각 레이어별의 predict)

확률적 경사 상승법을 사용한 손실 최대화

```
@tf.function
def gradient_ascent_step(image, filter_index, learning_rate):
        with tf.GradientTape() as tape:
                tape.watch(image) # 이미지 텐서는 텐서플로 변수가 아니기 때문에 명시적으로 지정한다.
                loss = compute_loss(image, filter_index) # 1.

    grads = tape.gradient(loss, image) # 2.
    grads = tf.mathl2_normalize(grads) # 3.
    image += learning_rate * grads # 4.
    return image # 5.
```

1. image 가 필터를 얼마나 활성화 하는지 손실을 계산한다.
2. 이미지에 대한 손실의 미분값을 계산한다.
3. 그래이디언트 정규화 트릭을 적용한다.
4. 필터가 더 강하게 반응할 수 있는 (더 강하게 활성화시킬 수 있는) 방향으로 이동한다.
5. 업데이트된 이미지를 반환한다.

필터 시각화 생성 함수

```
img_width = 200
img_height = 200

def generate_filter_pattern(filter_index):
        iterations 30 # 1.
        learning_Rate = 10. # 학습률
        image = tf.random.uniform( # 2.
                minval = 0.4,
                maxval = 0.6,
                shape = (1, img_width, img_height, 3))
        for i in range(iterations):
                image = gradient_ascent_step(image, filter_index, learning_rate) # 3.
        return image[0].numpy()
```

1. 경사 상승법 단계 적용 횟수
2. 3개의 채널을 가진 무작위한 이미지를 생성한다. Xception 모델은 [-1, 1] 범위의 입력값을 기대하고 여기서는 0.5를 중심으로 범위를 선택했다.
   0.4~0.6으로 최소 최대를 설정한 이유는 특정 패턴에 집중하도록 하기 위함이다.
3. 경사 상승법을 적용해 손실 함수를 최대화 하도록 반복적 업데이트한다.

텐서를 이미지로 변환하기 위한 유틸리티 함수

```
# 텐서를 이미지로 변환해주는 후처리 함수
def deprocess_image(image):
    # [0,255] 범위로 이미지 텐서를 정규화한다.
    image -= image.mean()
    image /= image.std()
    image *= 64
    image += 128
    image = np.clip(image, 0, 255).astype("uint8")

    # 부수 효과를 피하기 위해 경계 픽셀을 제외시킨다.
    image = image[25:-25, 25:-25, :]

    return image
```

```
plt.axis("off")
plt.imshow(deprocess_image(generate_filter_pattern(filter_index=2)))
plt.show()
```

block3_sepconv1 층에 있는 세번째 필터(채널, filter_index=2)이 최대로 반응하는 패턴을 출력한다.

```
all_images = [] # 층에 있는 처음 64개의 필터를 시각화하여 저장한다.
for filter_index in range(64):
    print(f"{filter_index}번 필터 처리중")
    image = deprocess_image(
        generate_filter_pattern(filter_index)
    )
    all_images.append(image)

# all_images 라는 리스트에 후처리를 끝낸 image들을 아래 과정들을 통해 시각화한다.

# 필터 시각화를 출력할 빈 이미지를 준비한다.
margin = 5  # 그림 구분 짓기 위한 margin
n = 8
cropped_width = img_width - 25 * 2
cropped_height = img_height - 25 * 2
width = n * cropped_width + (n - 1) * margin
height = n * cropped_height + (n - 1) * margin
stitched_filters = np.zeros((width, height, 3))

for i in range(n): # 저장된 필터로 이미지를 채운다.
    for j in range(n):
        image = all_images[i * n + j]
        stitched_filters[
            (cropped_width + margin) * i : (cropped_width + margin) * i + cropped_width,
            (cropped_height + margin) * j : (cropped_height + margin) * j
            + cropped_height,
            :,
        ] = image

keras.utils.save_img( # 이미지를 디스크에 저장한다.
    f"filters_for_layer_{layer_name}.png", stitched_filters)
```