CNN과 ViT 모델의 제작 및 유사 클래스 분석 프로젝트

강유진

1. 실험기간

2023년 11월 28일 ~ 2023년 12월 17일

2. 사용한 기술

- Python
- Tensorflow
- Pandas
- Numpy
- Matplotlib
- Google Colab

3. 목적 및 필요성

데이터셋이 적을 경우 비전 트랜스포머(ViT)에 비해 합성곱신경망(CNN)이 성능이 뛰어나다고 알려져있다. 하지만, 이는 검증 데이터셋에 대한 경우이며 실제 사용의 경우에는 어떤 결과가 나타나는지 알지 못한다. ViT와 CNN의 실생활 데이터의 추측 결과와 차이에 대해 분석했다.

4. 결과 요약

- ViT보다 정확도가 뛰어난 CNN에서 실제 데이터에 대해 ViT보다 못한 성능을 보였다.
- 이는 유사 클래스에서 특히 두드러지게 나타났다.
- 검증 데이터셋의 확률 분포상 CNN보다 ViT가 유사한 클래스를 더 잘 구분한다.

5. 학습 데이터 정보

- 60,000개의 예제로 구성된 훈련 세트와 10,000개의 예제로 구성된 테스트 세트로 구성
- 28x28 픽셀 회색조 (784픽셀)
- 10개 클래스 중 하나에 할당

0	1	2	3	4	5	6	7	8	9
T-shirt/ top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot

6. 주요 연구 내용

※ 데이터 확인 등 사전 작업은 제외하고 기술한다.

가. CNN 구현

1) 구현 코드

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

from sklearn.model_selection import train_test_split
```

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels,
test_size=0.2, random_state=13)

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
val_images = val_images.reshape(val_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)

train_images = train_images /255.0
val_images = val_images /255.0
test_images = test_images /255.0
```

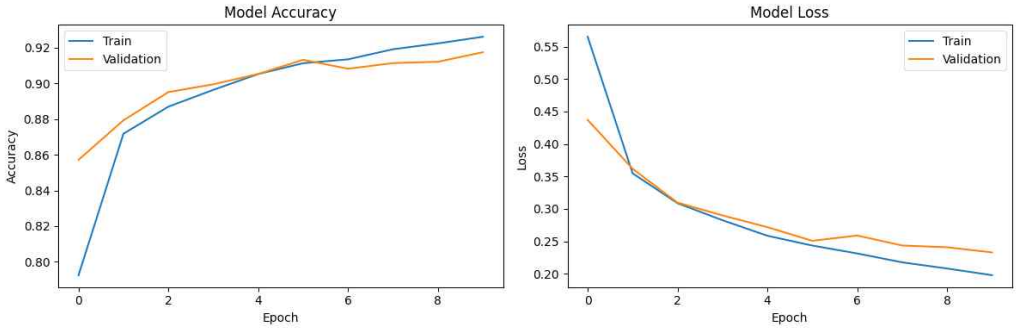
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense

model = Sequential()
input_shape = (28, 28, 1)
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Dropout(0.2))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64,(3,3), padding='same', activation='relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64,(3,3), padding='valid', activation='relu'))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(10))

model.summary()
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, batch_size=64,
validation_data=(val_images, val_labels))

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

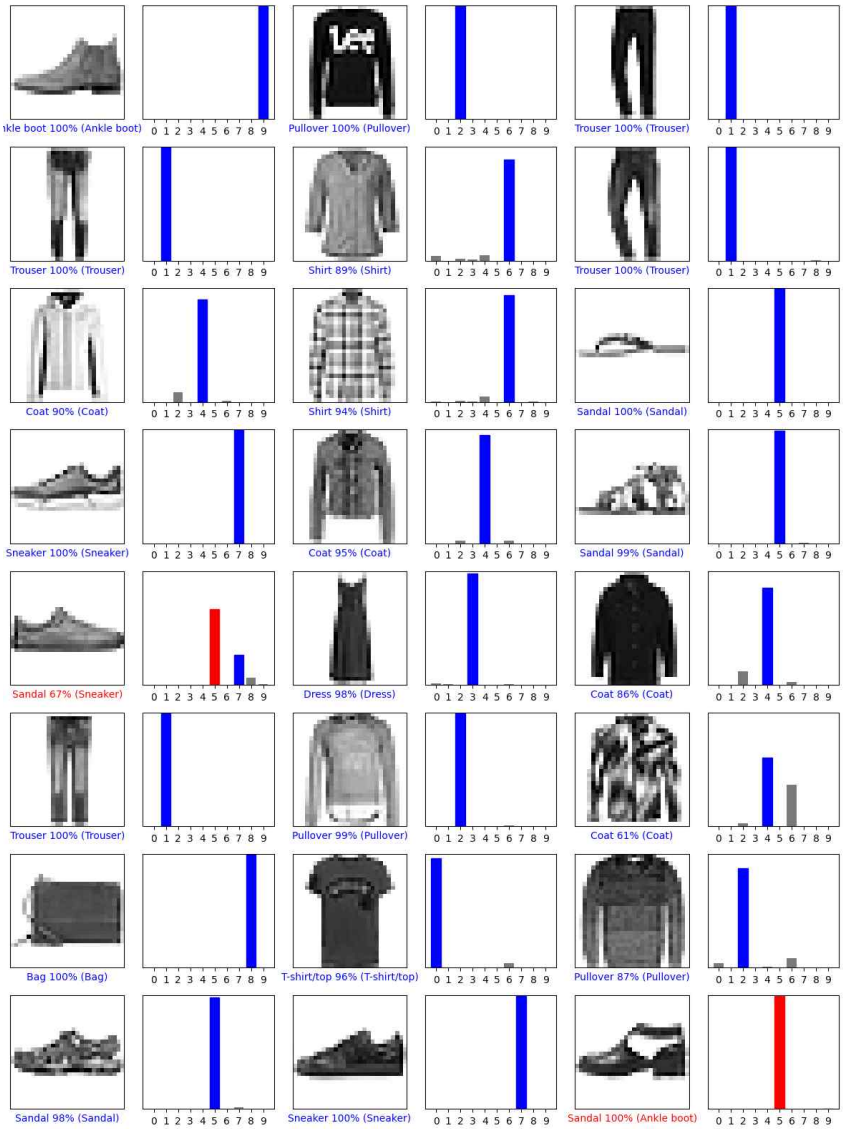
2) 모델 성능

(가) 정확도: 91.11%

(나) 정확도와 손실함수 변화



(다) 샘플 24개의 확률 분포



[그림 1]

나. ViT 구현

1) 구현 코드

```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models
import numpy as np
```

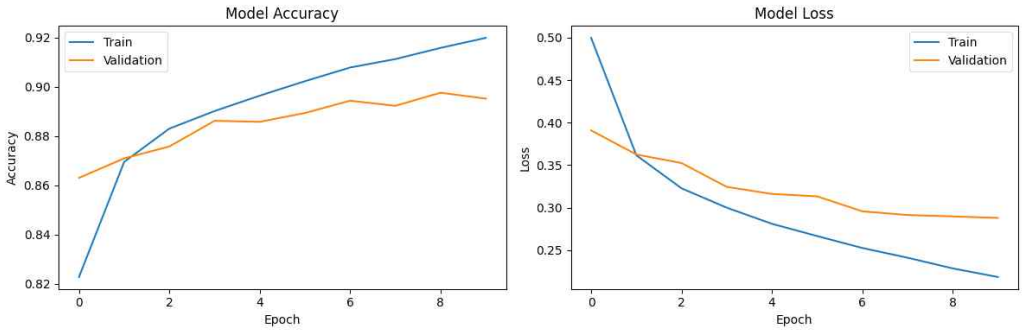
```
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels,
test_size=0.2, random_state=13)

train_images = train_images.astype('float32') /255.0
val_images = val_images.astype('float32') /255.0
test_images = test_images.astype('float32') /255.0

train_images = tf.image.resize(train_images[..., tf.newaxis], (32, 32))
val_images = tf.image.resize(val_images[..., tf.newaxis], (32, 32))
test_images = tf.image.resize(test_images[..., tf.newaxis], (32, 32))

train_labels = tf.keras.utils.to_categorical(train_labels, num_classes=10)
val_labels = tf.keras.utils.to_categorical(val_labels, num_classes=10)
test_labels = tf.keras.utils.to_categorical(test_labels, num_classes=10)
```

```
def create_vit_model():
    patch_size =4
    num_patches = (32 // patch_size) **2
    patch_dim =3 * patch_size **2
    inputs = layers.Input(shape=(32, 32, 1))
    patches = layers.Conv2D(patch_dim, kernel_size=patch_size, strides=patch_size,
          padding="valid")(inputs)
    reshaped_patches = layers.Reshape((num_patches, patch_dim))(patches)
    position_embeddings = layers.Embedding(input_dim=num_patches,
          output_dim=patch_dim)(np.arange(num_patches))
    embeddings = reshaped_patches + position_embeddings
    num_transformer_blocks =6

    for _ in range(num_transformer_blocks):
        x = layers.MultiHeadAttention(num_heads=8, key_dim=64)(embeddings,
         embeddings)
        x = layers.LayerNormalization()(x + embeddings)
        x = layers.Dense(256, activation="relu")(x)
        x = layers.Dense(patch_dim)(x)
        embeddings = layers.LayerNormalization()(x + embeddings)
    outputs = layers.Flatten()(embeddings)
    outputs = layers.Dense(10, activation="softmax")(outputs)

    model = models.Model(inputs=inputs, outputs=outputs)
    return model

vit_model = create_vit_model()
vit_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
history = vit_model.fit(train_images, train_labels, epochs=10, batch_size=64,
validation_data=(val_images, val_labels))
test_loss, test_acc = vit_model.evaluate(test_images, test_labels)
```

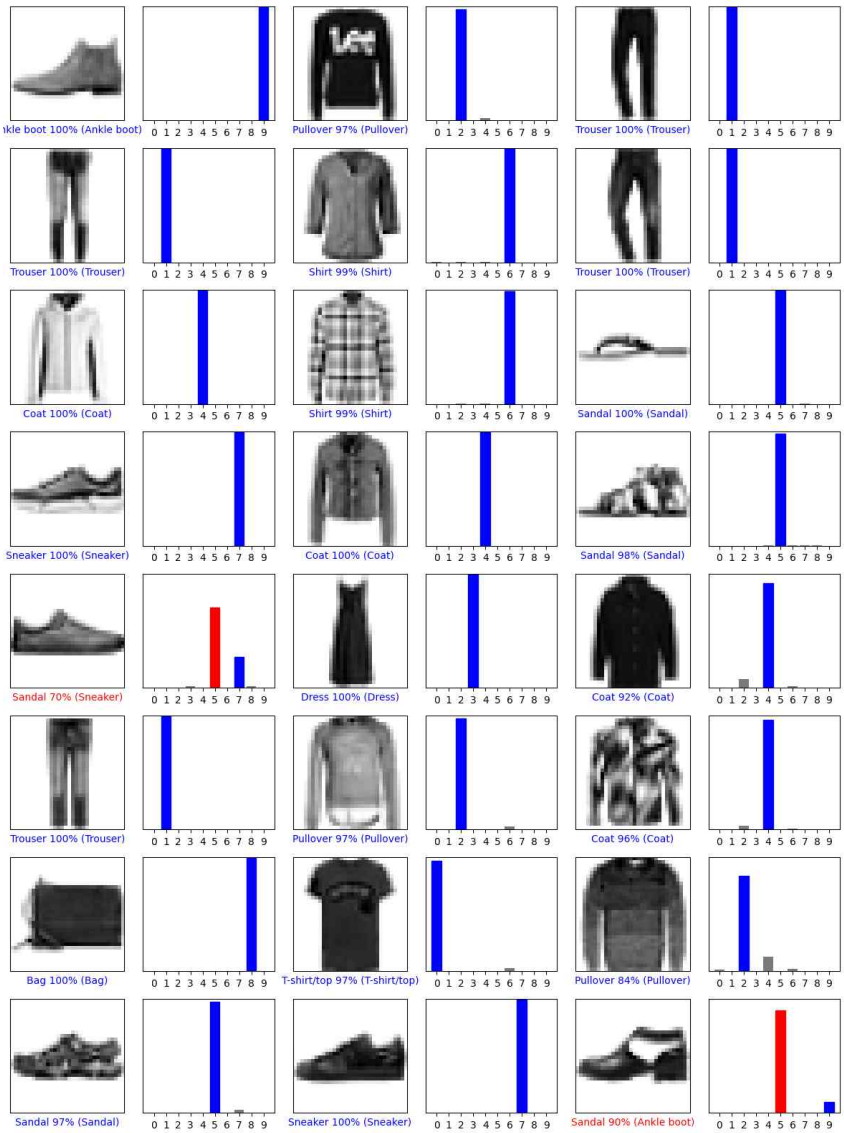
2) 모델 성능

(가) 정확도: 89.26%

(나) 정확도와 손실함수 변화



(다) 샘플 24개의 확률 분포



[그림 2]

다. 결과 비교 분석

※ 모델 간 비교를 위해 에폭, 손실함수, 옵티마이저, 학습률, 성능 측정 지표(정확도), 배치 사이즈를 동일하게 했다.

1) 정확도 결과 비교

모델	확률	Google Colab 캡처 화면
CNN	91.11% (% 환산 시)	
ViT	89.29%	

측정된 정확도는 학습에 사용되지 않은 Fashion MNIST 데이터를 사용했다. 각 모델의 코드 중 데이터 선정 부분에 작성된 'random_state=13'을 통해 무작위 선별한다. 하지만, 두 모델이 학습하는 데이터셋과 검증하는 데이터셋은 서로 같아야 하므로 같은 값(13)을 넣어서 조정했다.

측정 결과, CNN은 91.11%의 정확도를, ViT는 89.26%의 정확도를 보였다. 즉, 정확도를 기준으로 검증 데이터셋에서 CNN이 ViT보다 뛰어나다고 할 수 있다.

2) 데이터셋 외 사진 입력 결과

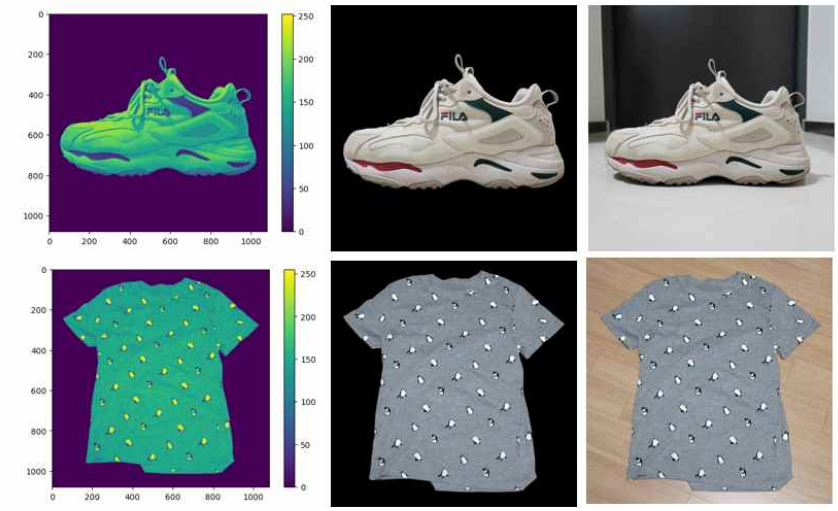
현실에서 유의미한 결과를 내기 위해서는 이런 유사한 형태의 클래스도 구분할 줄 알아야 할 것이다. 데이터셋 내에 서로 비슷한 클래스가 있으므로 그에 해당하는 의상 사진을 직접 촬영하여 각 모델에 입력값으로 넣었다. 그 결과를 비교, 대조하여 실험했다.

우리가 흔히 쇼핑몰에서 대분류와 소분류로 나누는 식으로 생각하면 비슷한 종류를 구분하기 쉽다. T-shirt와 Shirt, 등은 상의, Sandal, Ankle Boot는 신발에 해당한다. 같은 대분류에 속하는 클래스를 유사 클래스라고 명명했다.

최종적으로 유사 클래스는 다음과 같이 분류했다.

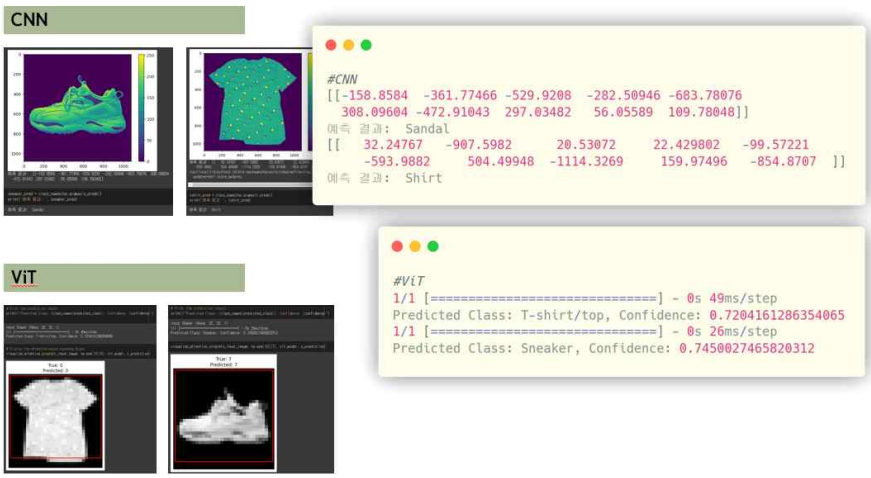
대분류	소분류 (번호)
상의	T-shirt/top(0), Pullover(2), Coat(4), Shirt(6)
하의	Trouser(1)
드레스(전신)	Dress(3)
신발	Sandal(5), Sneaker(7), Ankle Boot(9)
잡화	Bag(8)

유사 클래스 중 상의와 신발에 해당하는 실제 사진을 가지고 추측하도록 실험해보았다. 우선 사진으로 선택한 것은, Sneaker(7)에 해당하는 신발과 T-shirt/top(5)에 해당하는 티셔츠다. 두 의상은 실제 내가 가진 물건들로, 신발은 진한 색과 흰색이 섞여 있어 회색조로 변경 시 배경과 구분되지 않는 부분이 생긴다. 티셔츠는 흰색의 작은 무늬가 전체적으로 배치되어 회색조 변경 시 객체 내 상대적으로 큰 값이 규칙적이게 나타난다. 이런 요소가 이상치로 작용할 것으로 기대하여 두 의상을 골랐다.



실제 모델에 입력되는 사진은 Fashion MNIST와 유사하게 변형 과정을 거쳤다. 유사 클래스 간 구분 성능을 알아보는 것이 목적이기에 사진을 찍을 때 방향과 배치 형태 등을 조정했다. 이후 배경은 값을 0으로 맞추기 위해 검정으로, 객체는 회색조로 변경했다. 회색조는 이미지로 명확하게 보이지 않아 확인 시에만 히트맵 형태로 바꾸었다.

그리고 모델의 추측 결과를 확인해보았다.



CNN은 Sandal(5)와 Shirt(6)이라고 잘못 추측했고 ViT는 놀랍게도 둘 다 맞췄다. 어째서일까? 분명 정확도는 CNN이 높았다. 겨우 두 장이기에 사진의 개수가 많았다면 결과가 달랐을 수도 있다. 하지만 91%의 정확도로 하나도 못 맞춘 CNN과 정확도가 더 낮으면서도 둘 다 맞춘 ViT 사이에는 무슨 차이가 있는지 관심이 생겼고, 이를 추론하리라 결정했다.

나는 확률 분포로 이 차이를 추측하고자 했다. 이유는 Sandal과 Shirt 모두 참값과 대분류가 같은 유사 클래스에 해당하기 때문이다. 즉, 추측 결과에서 유사 클래스의 결과도 함께 파악함으로써 결과의 원인을 파악하고자 했다.

3) 샘플 데이터 결과 비교 분석

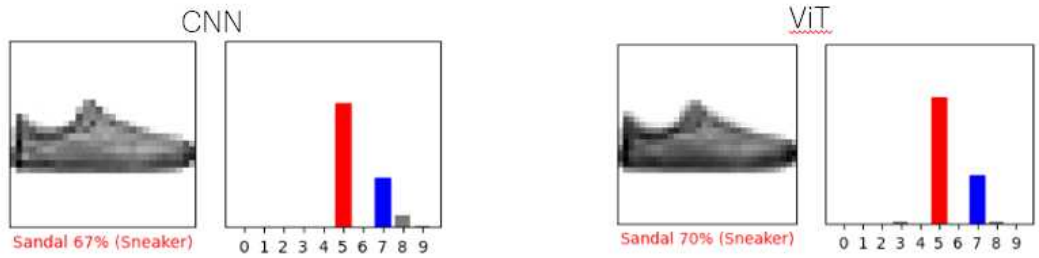
유사 클래스의 결과를 파악하기 위해 위의 24가지의 샘플 중, 확률 분포에 차이가 큰 세 가지 항목을 살펴보겠다. 세 가지 항목은 CNN과 ViT에서 추측한 값이 같다. 그러나, 이 분포에 차이를 지니고 있다. 이런 결과로 선정한 이유는 다음과 같다.

- 서로 같은 클래스로 추측했으므로 정확도 차이에 영향을 준 샘플이 아니다.
- 확률 분포에 차이가 있다는 것은 다른 입력값에 다른 출력을 내놓을 수 있다는 뜻이다.
- 실제 촬영한 사진의 결과가 정확도와 완전히 다른 값을 내놓은 이유가 정확도에 영향을 주지 않으나 예측값에 영향을 끼치는 요인 때문일 수 있다.

(가) 특징적이지 않은 결과들

[그림 1]과 [그림 2]를 참고하면, 샘플 내 특징적이지 않은 값들은 두 모델 전부 유사한 형태의 추측 확률 분포를 가진다. 대게 두 모델 모두에서 신뢰도가 높으며 정답 값이다. 그렇기에 주요한 차이를 보인 결과에 대해서만 알아보겠다.

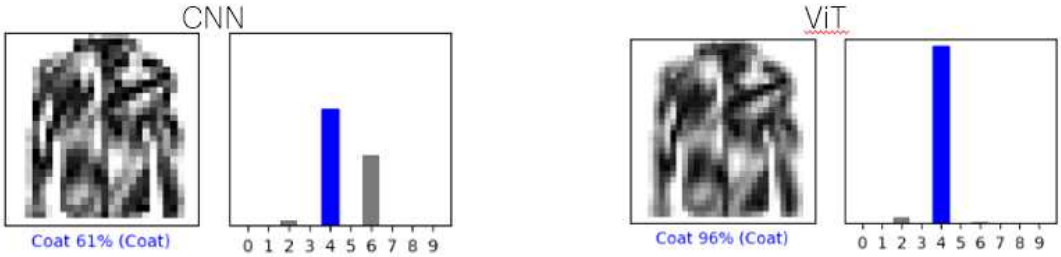
(나) Sandal (Sneaker)



Sandal(5)이라 추측한 Sneaker(7)는 CNN과 ViT 모두 알맞은 클래스를 찾아내는 데 실패했다. 하지만, 유사 클래스의 확률에선 정확도와 다른 부분이 있다. 두 모델의 예측 확률을 보면, 아예 대분류가 다른 Bag(8)이란 항목으로 분류될 가능성이 있다. 그런데, ViT와 CNN은 서로 다른 확률을 보인다. ViT는 Bag(8)일 확률이 눈에 띄게 낮다. 오히려 정답 값인 Sneaker(7)에 속할 확률은 CNN보다 높다.

즉, ViT는 Sandal이란 유사 클래스로 추측할 확률이 더 높지만, Bag이라는 아예 다른 분류로 추측할 확률은 낮게 나타났다. 그리고 알맞은 값인 Sneaker에 대한 확률도 ViT가 더 높다.

(다) Coat (Coat)

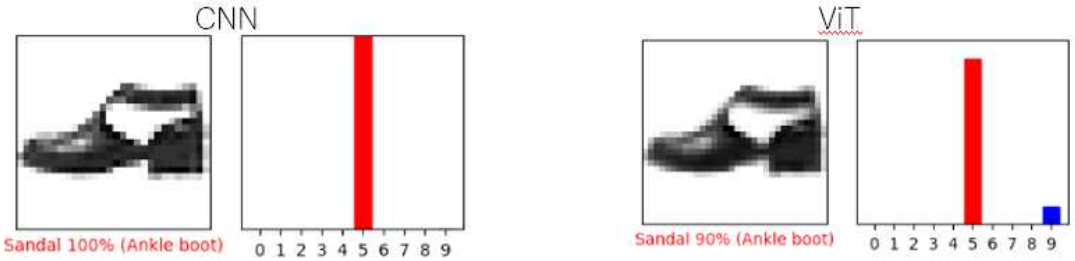


Coat(4)는 둘 다 정답값을 맞췄지만, 그 확률 분포에 큰 차이를 보인다. 예측값인 Coat(4)를 빼고 나머지를 보면, 차이가 명확하게 보인다. CNN은 Shirt(6)이라고 추측할 확률이 높지만, ViT는 Shirt(6)이라고 추측할 확률이 거의 없다. 긴 팔 상의에 해당하는 Pullover(2)는 둘 다 약간의 확률을 가진다.

또한, 정답 추측의 확신 정도로 보면 무려 35% 차이가 난다. 이 결과를 통해 정확도는 확신한 정도를 뜻하지 않는다는 것을 알 수 있다. 즉, CNN이 ViT보다 정확도가 높다고 해서 더 확실하게 추측했다고 말할 수는 없다.

이 결과는 유사 클래스인 Shirt와 Coat 간 분류에서 CNN보다 ViT가 더 잘 구분한 예시다.

(라) Sandal(Ankle Boot)



Sandal(5)이라 추측한 Ankle boot(9)의 결과는 명확한 차이를 지닌다. 결론적으로 선택한 값이 가장 높은 확률인 Sandal(5)이라는 건 같다. 그러나 CNN은 완전히 오답을 선택했고, ViT는 정답일 가능성이 있다.

단 10%라도 'B일 수 있지만, A일 것이다'와 100%로 'A이다'는 완전히 다른 이야기다. 반대로 말하면 정답일 확률이 CNN에서는 0%이고, ViT에서는 10%이다.

10%의 차이가 정답 값이라는 것에 초점을 맞춰보자. 이 모델들은 분류 모델로 쓰이지만, 만약 예측한 확률 비율에 따라 각 카테고리의 인기 제품을 10건 보여준다고 하자. 어떤 사람이 Ankle boots를 사려고 한다면, CNN은 10건 전부 Sandal을 보여줄 것이고 ViT는 9건은 Sandal이지만, 1건이라도 Ankle Boot를 보여줄 것이다.

위의 극단적인 예시가 아니더라도, 이런 클래스 차이는 실제 모델의 예측에 영향을 줄 수 있다. 이 모델이 실제 사용되는 모델이고, 성능 개선을 위해 전이 학습한다고 가정해보자. 그러면 유사 클래스를 구분할 수 있는 능력은 전이 학습 시에도 영향을 끼칠 것이다.

위의 3가지 특징적인 샘플의 결과를 바탕으로 알아낸 사실은 다음과 같다.

- 정확도에 반영되지 않은 성능이 학습되지 않은 데이터에 대한 신뢰도를 떨어뜨릴 수 있다.
- 정확도가 확신도를 의미하지 않는다.

7. 활용 가능성

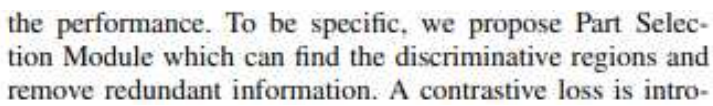
- 인공지능의 '성능이 뛰어나다'에 대한 다른 관점을 제시할 수 있다. 물론 기존에도 Recall, F1 Score 등의 방법이 있지만, 해당 방법들은 유사 클래스의 분포에 대한 부분까지 다루지는 않는다.
- 실제 비교한 CNN과 ViT의 모델이 다양성이 부족하나, 그 두 모델로 정확도가 실제로도 뛰어남을 증명하는지, 소량 데이터셋에서 CNN이 ViT보다 뛰어난 것을 의미하는지 살펴보았다.
- 유사 클래스 간의 연결성이 높아지면 임베딩과 같이 각 클래스의 연관성에 목적이 있는 경우 ViT를 유용하게 사용할 수 있다.
- ViT를 기반으로 전이 학습 시 더 뛰어난 효과를 낼 것이라 기대할 수 있다.

8. 추후 연구 방향

- 미세 분류의 영향력에 대한 실험: 미세 분류가 적은 데이터셋에서 유의미한 분류 차이를 불러오는가?
- 유사 클래스 간의 차이를 모델의 결과 산출에 도움이 되도록 만드는 방법
- 다양한 객체에 대한 ViT와 다른 모델 간의 성능 실험
- ViT의 예측한 이유 (가중치)를 수치화하거나 시각화할 방법

9. 추가 연구

[그림 3]은 논문 <TransFG: A Transformer Architecture for Fine-Grained Recognition>에 나오는 내용이다. 미세 분류라고 불리는 ViT의 특징을 '판별 영역을 찾을 수 있는 부분 선택 모듈'이라고 설명하고 있다. 이 특징으로 인해 ViT가 비슷한 종류를 더 잘 구분할 수 있는 것이라면, 위의 결과의 근본적 원인을 알 수 있을 것이다.


the performance. To be specific, we propose Part Selection Module which can find the discriminative regions and remove redundant information. A contrastive loss is intro-

[그림 3]

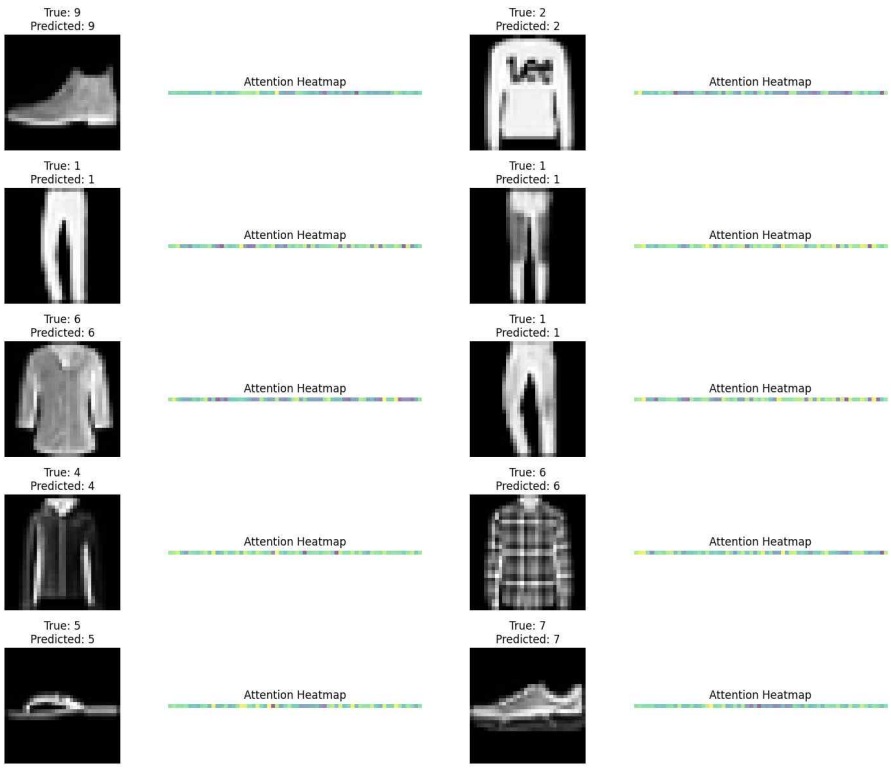
(Ref: He, 853p, Vol. 36 No. 1: AAAI-22 Technical Tracks 1)

미세 분류가 진짜 영향을 끼치는 게 맞는지 어떻게 알아볼까? CNN에서 자주 사용하던 방법인 Grad-CAM처럼 어느 부분에 가중치를 두었는지 찾아내는 것을 고려했다. 인터넷에 ViT의 가중치를 이미지에 띄우는 방법을 찾아보았으나 ViT가 최근 연구인 탓인지 아직 이에 관련된 코드가 공유된 것은 없었다. 그래서 수업 시간에 배운 Grad-CAM 방식을 응용하여 Attention Heatmap 코드를 작성했다.

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
```

```
def generate_attention_heatmap(image, model):
        intermediate_layer_model = tf.keras.Model(inputs = model.input,
                outputs = model.layers[3].output)
        attention_outputs = intermediate_layer_model(image)
        attention_weights = tf.reduce_mean(attention_outputs,axis=-1)
        attention_weights = tf.squeeze(attention_weights)
        returnattention_weights.numpy()

def visualize_predictions_with_attention_heatmap(images, labels, predictions, model):
        plt.figure(figsize=(15,12))
    for in range(10):
        image = images[i].numpy()
        true_label = np.argmax(labels[i])
        predicted_label = np.argmax(predictions[i])

        attention_map = generate_attention_heatmap(np.expand_dims(image,axis=0),model)
        plt.subplot(5,4,2* i + 1)
        plt.imshow(np.squeeze(image),cmap='gray')
        plt.title(f"True: {true_label}₩nPredicted: {predicted_label}")
        plt.axis('off')

        plt.subplot(5,4,2* i + 2)
        plt.imshow(np.expand_dims(attention_map,axis=0),cmap='viridis',alpha=0.6)
        plt.title('Attention Heatmap')
        plt.axis('off')
        plt.tight_layout()
        plt.show()

visualize_predictions_with_attention_heatmap(test_images,test_labels,predictions,vit_model)
```

이 코드의 결과는 다음과 같이 나온다.

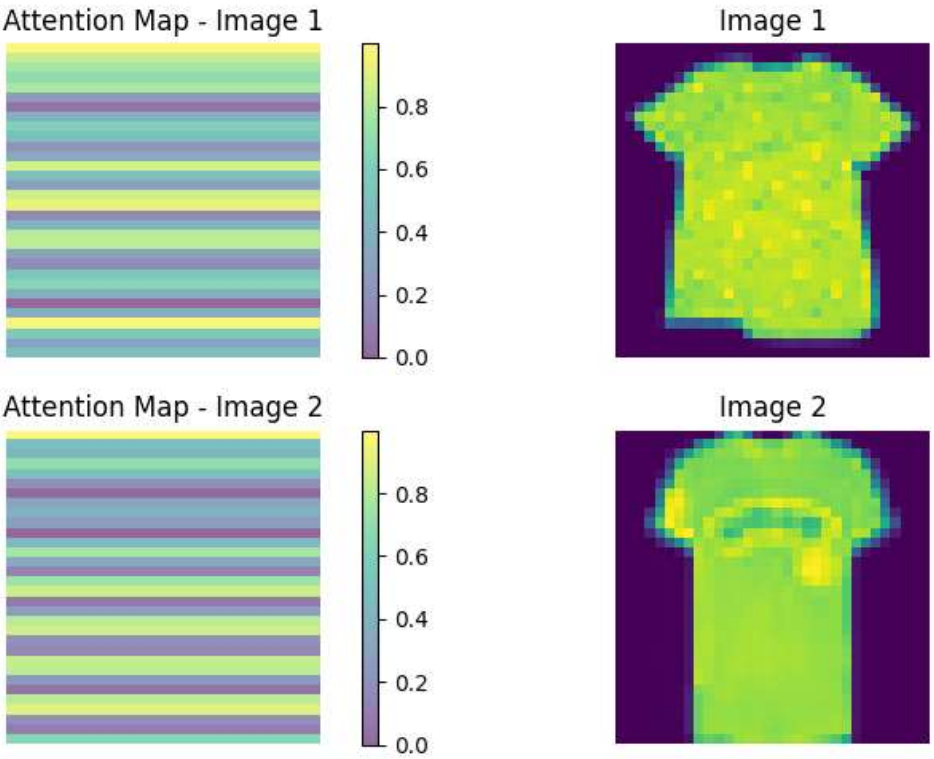


출력된 결과물이 얇은 줄 형태로 보이게 되는데, 구분이 쉽지 않아 이미지를 늘려 확인해보았다. 그리고 구분된 클래스가 같은 것이 실제로 유사한 히트맵을 가지는지 알아보기 위해 두 이미지와 히트맵을 동시에 띄운다.

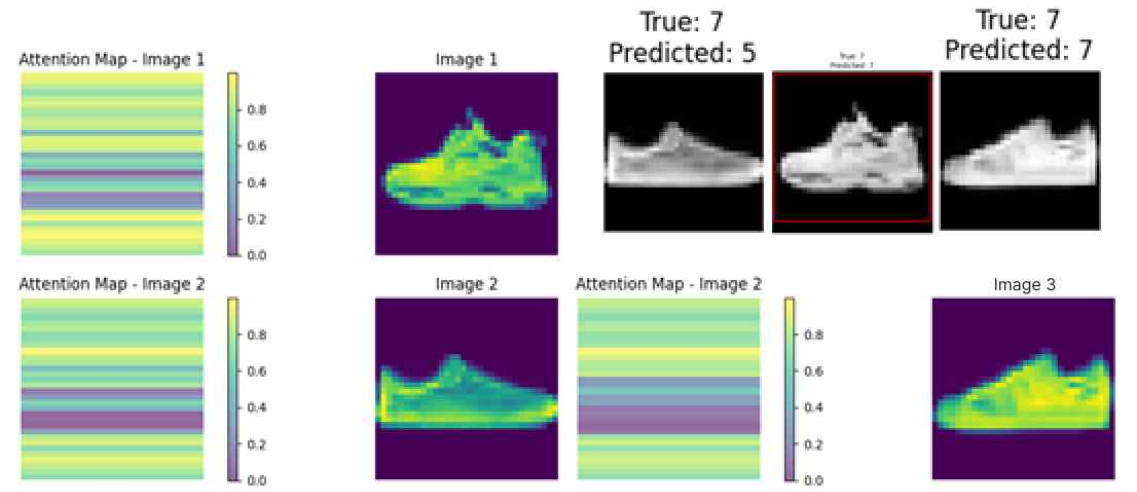
```
def visualize_attention_comparison(image1, image2, prediction1, prediction2, model,
        threshold=0.9):
        // 코드 생략

visualize_attention_comparison(t_input_image, test_images[19], t_prediction,predictions[19],
        vit_model)
```

직접 찍은 사진을 넣어서 실험했다. 19번 이미지가 티셔츠로 분류되었고 TP (참, 긍정)이기 때문에 19번 이미지와 직접 찍은 사진을 비교한다.

두 히트맵은 실제로 유사한 분포를 띄우고 있었다. 가중치가 극도로 높은 부분과 가중치가 극도로 작은 부분의 전체적인 패턴이 유사한 위치임을 확인할 수 있다.



하지만, 이 두 가지가 비슷하다고 해서 미세 분류 특징이 드러난다고 할 수는 없다. 다른 예시인 신발에 대해서도 보겠다. 신발은 세 가지를 비교해보았다. ViT가 오답을 낸 것도 비교하는 것이다.



일단, Sandal(5)이라 추측한 Sneaker(7)인 Image 2와 제대로 Sneaker로 추측한 Image 1과 Image 3을 비교해보겠다. 전체적인 패턴 중 유사한 부분이 눈에 띈다. 세 신발의 히트맵이 각각 다르지만, 위의 티셔츠의 히트맵과 대조하면 확연하게 다르고 비교적 신발들끼리는 비슷하다.

그런데, Image 1은 제대로 추측했으나 Image 2를 제대로 추측한 이유가 이해 안 될 정도로 Image 2와 Image 3이 유사한 히트맵을 보였다.

어째서일까? 이 부분도 미세 분류 특성이 작용했기 때문이라 추측한다. 전체적인 패턴이 비슷하게 보이지만, Sandal과 Sneaker를 구분 짓는 미세한 차이로 인해 다른 클래스로 판단한 것이다. 그리고 이 모델에서는 그 미세한 차이를 잘못 학습했으리라 추측할 수 있다. 이는 시간의 문제로 추가적인 실험을 해보지 못했다.

결과적으로 얻어낸 사실은 같은 클래스에 속한 개체가 서로 유사한 히트맵을 보인다는 것이다. 하지만, 유사한 클래스와 FP (거짓, 긍정) 추측에 대해 시각화한 결과만으로 그 이유를 추측해내기 어려웠으므로 수치적인 결과를 포함한 후속 연구가 필요하다. 수치 결과는 각 어탠션 당 값 차이를 알아내는 방법 등을 고려할 수 있다.