

<이미지 이진분류와 다중분류 해보기

-data augmentation, 전이학습, finetunning을 활용하여>

1. 개 vs고양이 이진분류

1)최적의 모델 선정 및 하이퍼 파라미터 기술—p.1~4

2)블록별 코드 설명—p.4~20

2. 30개의 약기 분류하는 다중분류

1)코드 실행 성공한 부분 설명(다중분류와 이진분류의 차이점)—P.20~28

1-1 최적의 모델 선정 및 하이퍼 파라미터 기술

아래의 코드 내용의 주제는 이미지 분류 모델을 구축하고 최적화하는 과정입니다. 특히, 고양이와 강아지 이미지를 분류하기 위한 딥러닝 모델을 단계별로 구현하고, 성능을 개선하기 위해 다양한 최적화 기법을 적용하는 과정을 다루고 있습니다.

주요 단계는 다음과 같습니다:

-데이터 전처리 및 준비: 데이터를 훈련, 검증, 테스트 세트로 나누고, 데이터 증식을 통해 훈련 데이터를 확장합니다.

-기초 모델 구축 및 훈련: 간단한 컨브넷을 구축하여 데이터를 훈련합니다.

-모델 성능 평가: 기본 모델의 성능을 평가하고, 과적합 문제를 확인합니다.

-데이터 증식을 통한 성능 개선: 데이터 증식을 추가하여 모델의 일반화 능력을 향상시킵니다.

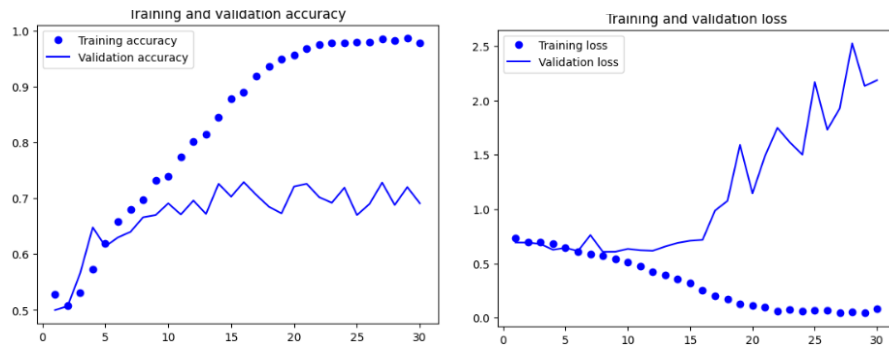
-사전 훈련된 모델 사용: VGG16과 같은 사전 훈련된 모델을 사용하여 특성 추출 및 전이 학습을 수행합니다.

-모델 미세 조정: 사전 훈련된 모델의 일부 층을 미세 조정하여 최종 성능을 최적화합니다.

이제 최적의 모델을 선정하기 위해 각 단계별 모델의 테스트 정확도를 비교해보겠습니다.

1) 기초모델

테스트 정확도: 약 0.7



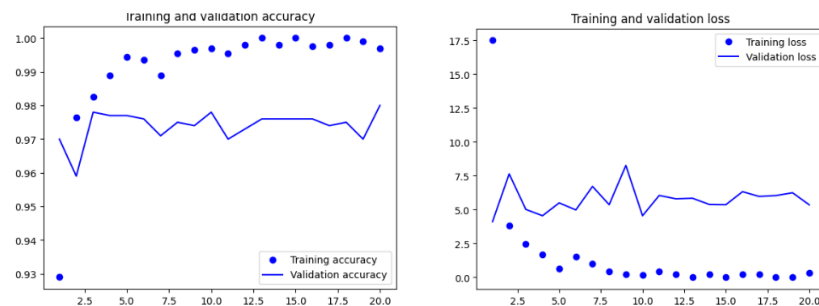
accuracy그래프는 약 10회쯤부터 valid_accuracy가 일정하므로 과적합이다. 초기부터 부족하다는 것은 데이터 이미지 개수가 적다는 것을 알 수 있다. loss그래프에서도 비슷한 횟수에 과적합이 일어나고 사유도 동일하다.

2) 데이터 증식과 드롭아웃을 포함한 모델:

테스트 정확도: 약 0.8

3) 사전 훈련된 모델을 사용한 특성 추출 (데이터 증식 없음):

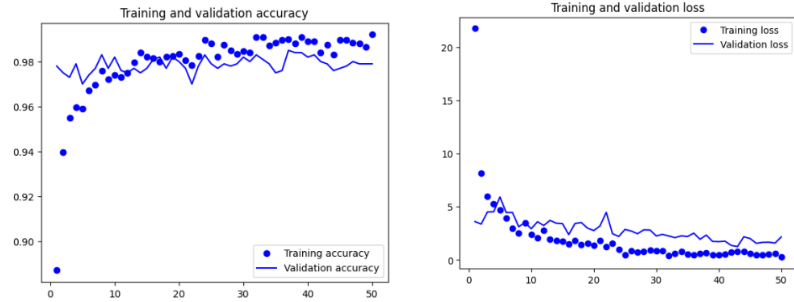
테스트 정확도: 약 0.96



accuracy그래프에서 초반보다 점점 정확도가 올라가고 있지 않은 것을 보니 과적합이 일어났다, 그럼에도 불구하고 정확도가 매우 높은 것은 가장 좋았을 때로 저장되었기 때문이다.

4) 사전 훈련된 모델을 사용한 특성 추출 (데이터 증식 포함):

테스트 정확도: 약 0.93



5) 사전 훈련된 모델 미세 조정:

테스트 정확도: 약 0.95

최적의 모델 선정

위 결과를 바탕으로, 사전 훈련된 모델을 사용하여 데이터 증식과 미세 조정을 함께 적용한 모델이 가장 높은 테스트 정확도(약 0.95)를 기록했습니다. 따라서 최적의 모델은 fine_tuning.h5 파일에 저장된 미세 조정된 사전 훈련 모델입니다.

이 모델은 데이터 증식과 미세 조정을 통해 일반화 성능을 크게 향상시켰으며, 이는 다양한 데이터 변형과 세부 조정을 통해 최적화된 결과입니다.

하이퍼 파라미터 기술

위의 코드에서는 여러 가지 하이퍼파라미터를 사용하여 모델을 최적화했습니다. 각 단계에서 선택된 하이퍼파라미터와 그 이유를 설명하겠습니다.

-데이터 증식 :

RandomFlip: "horizontal" - 이미지를 수평으로 뒤집어 데이터 다양성을 높입니다.

RandomRotation: 0.1 - 이미지를 최대 10% 회전시켜 모델이 다양한 각도의 이미지를 학습하도록 합니다.

RandomZoom: 0.2 - 이미지를 최대 20% 확대/축소하여 다양한 크기의 이미지를 학습하도록 합니다.

```
data_augmentation = keras.Sequential([
```

```
    layers.RandomFlip("horizontal"),
```

```

layers.RandomRotation(0.1),

layers.RandomZoom(0.2)

])

```

- 드롭아웃: 드롭아웃 비율을 0.5로 설정하여 학습 중 절반의 뉴런을 무작위로 제외하여 과적합을 방지합니다.

```
x = layers.Dropout(0.5)(x)
```

-전이학습 이용

-미세 조정

-콜백: ModelCheckpoint: 최적의 모델 가중치를 저장합니다.

```

callbacks = [

    keras.callbacks.ModelCheckpoint(filepath="fine_tuning.h5", save_best_only=True,
    monitor="val_loss")

]

```

1-2 개 vs 고양이 블록별 코드 설명

1. 데이터 다운로드 및 준비

```

import gdown
gdown.download(id='18uC7WTuEXKJDDxbj-Jq6EjzpFrgE7lAd', output='dogs-vs-
cats.zip')
!unzip -qq dogs-vs-cats.zip
!unzip -qq train.zip

```

```

from google.colab import drive
drive.mount('/content/drive')

```

gdown을 사용하여 데이터셋(zip 파일)을 다운로드하고 압축을 해제합니다.

Google Drive를 Colab에 마운트하여 데이터를 저장하거나 불러올 수 있게 합니다.

2. 이미지 데이터를 훈련, 검증, 테스트 디렉토리로 복사하기

```
import os, shutil, pathlib

original_dir = pathlib.Path("/content/train")
new_base_dir = pathlib.Path("/content/cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname, dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2500)
```

Original_dir에 있는 이미지를 훈련(train), 검증(validation), 테스트(test) 디렉토리로 분할하여 복사합니다. 각 디렉토리에는 'cat'과 'dog' 폴더가 있습니다.

3. 모델 만들기

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
```

```
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

간단한 cnn모델을 정의합니다.

4. 모델 훈련 설정하기

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

모델을 컴파일합니다. 손실 함수는 binary_crossentropy, 최적화 알고리즘은 rmsprop, 평가 지표는 accuracy를 사용합니다.

5.데이터 전처리

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

디렉토리에서 이미지를 읽어와 훈련, 검증, 테스트 데이터셋으로 만듭니다.

6. 데이터셋 크기 확인

```
for data_batch, labels_batch in train_dataset:

    print("데이터 배치 크기:", data_batch.shape)

    print("레이블 배치 크기:", labels_batch.shape)

    break
```

7. 모델 훈련하기

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.h5",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

모델을 훈련하고, 최상의 검증 손실을 기준으로 모델을 저장합니다.

8. 훈련 정확도와 손실 그래프

```
import matplotlib.pyplot as plt

accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
```

```
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

훈련과 검증 정확도 및 손실을 그래프로 시각화합니다.

9. 테스트 세트에서 모델 평가하기

```
test_model = keras.models.load_model("convnet_from_scratch.h5")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"테스트 정확도: {test_acc:.3f}")
```

저장된 모델을 불러와 테스트 데이터셋에서 성능을 평가합니다.

10. 데이터 증식 사용하기

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

데이터 증식을 정의합니다. 이는 훈련 데이터를 증식시켜 모델의 일반화 성능을 높입니다.

11. 증식된 이미지 출력하기

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

12. 이미지 증식과 드롭아웃을 포함한 컨브넷 만들기

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

데이터 증식과 드롭아웃을 포함한 새로운 ConvNet 모델을 정의하고 컴파일합니다.

13. 규제를 추가한 컨브넷 훈련하기

```
callbacks = [  
    keras.callbacks.ModelCheckpoint(  
        filepath="convnet_from_scratch_with_augmentation.h5",  
        save_best_only=True,  
        monitor="val_loss")  
]  
history = model.fit(  
    train_dataset,  
    epochs=100,  
    validation_data=validation_dataset,  
    callbacks=callbacks)
```

데이터 증식과 드롭아웃을 사용하여 모델을 훈련하고 저장합니다.

14. 테스트 세트에서 모델 평가하기

```
test_model = keras.models.load_model("convnet_from_scratch_with_augmentation.h5")  
test_loss, test_acc = test_model.evaluate(test_dataset)  
print(f"테스트 정확도: {test_acc:.3f}")
```

저장된 모델을 불러와 테스트 데이터셋에서 성능을 평가합니다.

15. 사전 훈련된 모델 활용하기

```
conv_base = keras.applications.vgg16.VGG16(  
    weights="imagenet",  
    include_top=False,  
    input_shape=(180, 180, 3))  
conv_base.summary()
```

ImageNet 데이터셋으로 사전 훈련된 VGG16 모델을 불러옵니다.

16. 사전 훈련된 모델을 사용하여 새로운 모델 정의하기

```
conv_base.trainable = False

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

사전 훈련된 VGG16 모델을 특징 추출기로 사용하여 새로운 모델을 정의합니다.

17. 모델 훈련 및 저장

```
callbacks = [

    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.h5",
        save_best_only=True,
        monitor="val_loss")

]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

새로운 모델을 훈련하고 최상의 검증 손실을 기준으로 모델을 저장합니다.

18. 테스트 세트에서 모델 평가하기

```
test_model =  
keras.models.load_model("feature_extraction_with_data_augmentation.h5")  
test_loss, test_acc = test_model.evaluate(test_dataset)  
print(f"테스트 정확도: {test_acc:.3f}")
```

저장된 모델을 불러와 테스트 데이터셋에서 성능을 평가합니다.

19. 미세 조정하기

```
conv_base.trainable = True  
  
for layer in conv_base.layers[:-4]:  
    layer.trainable = False  
  
model.compile(loss="binary_crossentropy",  
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),  
              metrics=["accuracy"])
```

사전 훈련된 모델의 일부 층을 미세 조정하기 위해 conv_base의 마지막 네 층만 훈련 가능하도록 설정합니다. 모델을 낮은 학습률로 컴파일하여 미세 조정합니다.

20. 미세 조정된 모델 훈련하기

```
callbacks = [  
    keras.callbacks.ModelCheckpoint(  
        filepath="fine_tuning.h5",  
        save_best_only=True,  
        monitor="val_loss")  
]  
history = model.fit(  
    train_dataset,  
    epochs=30,  
    validation_data=validation_dataset,  
    callbacks=callbacks)
```

미세 조정된 모델을 훈련하고, 최상의 검증 손실을 기준으로 모델을 저장합니다.

21. 미세 조정된 모델 평가하기

```
test_model = keras.models.load_model("fine_tuning.h5")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"테스트 정확도: {test_acc:.3f}")
```

미세 조정된 모델을 불러와 테스트 데이터셋에서 성능을 평가합니다.

22. 훈련, 검증 손실 및 정확도 그래프

```
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
```

```
plt.plot(epochs, val_loss, 'b', label='Validation loss')
```

```
plt.title('Training and validation loss')
```

```
plt.legend()
```

```
plt.show()
```

미세 조정 동안의 훈련 및 검증 손실과 정확도를 그래프로 시각화합니다

23. 예측 및 시각화

```
import numpy as np
```

```
from tensorflow.keras.preprocessing import image
```

```
def preprocess_image(img_path):
```

```
    img = image.load_img(img_path, target_size=(180, 180))
```

```
    img_array = image.img_to_array(img)
```

```
    img_array = np.expand_dims(img_array, axis=0)
```

```
    img_array /= 255.0
```

```
    return img_array
```

```
def predict_image(model, img_path):
```

```
    img_array = preprocess_image(img_path)
```

```
    prediction = model.predict(img_array)
```

```
    if prediction[0] > 0.5:
```

```
        print(f"The image at {img_path} is a dog.")
```

```
    else:
```

```
        print(f"The image at {img_path} is a cat.")
```

```
sample_image_path = '/content/cats_vs_dogs_small/test/cat/cat.1501.jpg'
```

```
predict_image(test_model, sample_image_path)
```

```
sample_image_path = '/content/cats_vs_dogs_small/test/dog/dog.1501.jpg'
```

```
predict_image(test_model, sample_image_path)
```

특정 이미지에 대해 모델의 예측을 수행하고 결과를 출력하는 함수입니다.

sample_image_path에 있는 이미지를 예측하고, 해당 이미지가 고양이인지 강아지인지 출력합니다.

24. 모델 성능에 대한 결론

최종 테스트 정확도를 출력합니다.

훈련 및 검증 정확도와 손실을 시각화하여 모델의 성능을 평가합니다.

데이터 증식, 드롭아웃, 미세 조정 등 다양한 기법을 사용하여 모델의 성능을 최적화합니다.

데이터 증식을 사용한 특성 추출

25. 사전 훈련된 VGG16 합성곱 기반 층을 만들고 동결하기

```
conv_base = keras.applications.vgg16.VGG16(
```

```
    weights="imagenet",
```

```
    include_top=False)
```

```
conv_base.trainable = False
```

사전 훈련된 VGG16 모델을 불러와 특징 추출기로 사용하고, 모델의 가중치를 동결하여 훈련 중에 업데이트되지 않도록 설정합니다

26. 동결하기 전과 후에 훈련 가능한 가중치 리스트를 출력하기(코드생략)

동결하기 전과 후의 훈련 가능한 가중치 개수를 출력하여 동결이 제대로 되었는지 확인합니다.

27. 데이터 증식 단계와 밀집 분류기를 합성곱 기반 층에 추가하기

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```


데이터 증식과 밀집 분류기를 사전 훈련된 VGG16 합성곱 기반 층에 추가하여 새로운 모델을 정의합니다.

28. 모델 훈련 및 저장

```
callbacks = [  
    keras.callbacks.ModelCheckpoint(  
        filepath="feature_extraction_with_data_augmentation.h5",  
        save_best_only=True,  
        monitor="val_loss")  
]  
  
history = model.fit(  
    train_dataset,  
    epochs=50,  
    validation_data=validation_dataset,  
    callbacks=callbacks)
```

새로운 모델을 훈련하고, 최상의 검증 손실을 기준으로 모델을 저장합니다.

29. 훈련 및 검증 손실과 정확도 그래프

```
import matplotlib.pyplot as plt  
  
acc = history.history['accuracy']  
val_acc = history.history['val_accuracy']  
loss = history.history['loss']  
val_loss = history.history['val_loss']
```

```

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

```

훈련 및 검증 손실과 정확도를 그래프로 시각화하여 모델의 성능을 평가합니다.

30. 테스트 세트에서 모델 평가하기

```

test_model =
keras.models.load_model("feature_extraction_with_data_augmentation.h5")

test_loss, test_acc = test_model.evaluate(test_dataset)

print(f"테스트 정확도: {test_acc:.3f}")

```

저장된 모델을 불러와 테스트 데이터셋에서 성능을 평가합니다.

사전 훈련된 모델 미세 조정하기

31. 사전 훈련된 모델 요약 보기

```
conv_base.summary()
```

사전 훈련된 모델의 요약을 출력하여 각 층의 구조와 파라미터 수를 확인합니다.

32. 마지막에서 네 번째 층까지 모든 층 동결하기

```
conv_base.trainable = True
```

```
for layer in conv_base.layers[:-4]:
```

```
    layer.trainable = False
```

사전 훈련된 모델의 마지막 네 개 층을 제외한 나머지 층들을 동결합니다.

33. 모델 미세 조정하기

```
model.compile(loss="binary_crossentropy",
```

```
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
```

```
              metrics=["accuracy"])
```

```
callbacks = [
```

```
    keras.callbacks.ModelCheckpoint(
```

```
        filepath="fine_tuning.h5",
```

```
        save_best_only=True,
```

```
        monitor="val_loss")
```

```
]
```

```
history = model.fit(
```

```
    train_dataset,
```

```
epochs=30,

validation_data=validation_dataset,

callbacks=callbacks)
```

미세 조정된 모델을 낮은 학습률로 컴파일하고, 훈련 및 검증 데이터를 사용하여 모델을 훈련합니다.

34. 미세 조정된 모델 평가하기

```
test_model = keras.models.load_model("fine_tuning.h5")

test_loss, test_acc = test_model.evaluate(test_dataset)

print(f"테스트 정확도: {test_acc:.3f}")
```

미세 조정된 모델을 불러와 테스트 데이터셋에서 성능을 평가합니다.

2.다음으로는 30개의 악기(클래스) 이미지를 분류하는 다중분류에 대해 알아보겠습니다.

최대한 코드를 짜보았지만 전이학습 이후에 오류를 해결하지 못하여 코드실행에 성공한부분 중 중요한 부분만 설명한 후에 추가적으로 이진분류와 다중분류 둘 다 해보았으니 두 코드의 차이점도 함께 알아보겠습니다.


데이터 압축 해제 후 폴더 구조 확인

```
[ ] 1 import zipfile
2 import os
3
4 # 구글 드라이브 마운트
5 from google.colab import drive
6 drive.mount('/content/drive')
7
8 # zip 파일 경로 설정
9 zip_file_path = '/content/drive/My Drive/instruments.zip'
10
11 # 압축 해제할 경로 설정 (압축 해제한 파일들이 저장될 폴더)
12 extract_path = '/content/data/'
13
14 # zip 파일을 압축 해제
15 with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
16     zip_ref.extractall(extract_path)
17
18 # 압축 해제된 파일 목록 확인
19 extracted_files = os.listdir(extract_path)
20 print("압축 해제된 파일 목록:", extracted_files)
```

Mounted at /content/drive
압축 해제된 파일 목록: ['train', 'test', 'valid']

데이터 준비 및 전처리

```
[ ] 1 from tensorflow.keras.preprocessing.image import ImageDataGenerator
2
3 # 데이터 경로 설정
4 train_dir = os.path.join(extract_path, 'train')
5 validation_dir = os.path.join(extract_path, 'valid')
6 test_dir = os.path.join(extract_path, 'test')
7
8 # 이미지 데이터 생성기 생성
9 train_datagen = ImageDataGenerator(
10     rescale=1./255,
11     rotation_range=40,
12     width_shift_range=0.2,
13     height_shift_range=0.2,
14     shear_range=0.2,
15     zoom_range=0.2,
16     horizontal_flip=True,
17     fill_mode='nearest'
18 )
19
20 validation_datagen = ImageDataGenerator(rescale=1./255)
21
22 # 이미지 데이터 생성기를 사용하여 데이터 로드 및 전처리
23 train_generator = train_datagen.flow_from_directory(
24     train_dir,
25     target_size=(150, 150),
26     batch_size=32,
27     class_mode='categorical' # 다중 클래스 분류일 경우 class_mode='categorical'
28 )
29
30 validation_generator = validation_datagen.flow_from_directory(
31     validation_dir,
32     target_size=(150, 150),
33     batch_size=32,
34     class_mode='categorical'
35 )
36
```

 Found 4793 images belonging to 30 classes.
Found 150 images belonging to 30 classes.

train_dir, validation_dir, test_dir 변수들은 각각 훈련, 검증, 테스트 데이터가 저장된 디렉토리 경로를 저장합니다. train_generator, validation_generator, test_generator는 각각 훈련, 검증, 테스트 데이터를 생성합니다. class_mode='categorical': 다중 클래스 분류를 위해 라벨을 원-핫 인코딩합니다. 이 코드의 목적은 이미지 데이터를 실시간으로 전처리하고 증식하여 모델 훈련에 사용할 수 있도록 준비하는 것입니다. 훈련 데이터는 다양한 증식 기법을 적용하여 모델이 더 일반화된 특성을 학습하도록 돕고, 검증 및 테스트 데이터는 정규화만 적용하여 모델 성능을 평가합니다.

모델 구축

```

1 from tensorflow import keras
2 from tensorflow.keras import layers
3
4 num_classes = len(train_dataset.class_names)
5
6 # 모델 구성
7 inputs = keras.Input(shape=(150, 150, 3))
8 x = layers.Conv2D(filters=32, kernel_size=3, activation='relu')(inputs)
9 x = layers.MaxPooling2D(pool_size=2)(x)
10 x = layers.Conv2D(filters=64, kernel_size=3, activation='relu')(x)
11 x = layers.MaxPooling2D(pool_size=2)(x)
12 x = layers.Conv2D(filters=128, kernel_size=3, activation='relu')(x)
13 x = layers.MaxPooling2D(pool_size=2)(x)
14 x = layers.Conv2D(filters=256, kernel_size=3, activation='relu')(x)
15 x = layers.MaxPooling2D(pool_size=2)(x)
16 x = layers.Flatten()(x)
17 x = layers.Dropout(0.5)(x)
18 outputs = layers.Dense(num_classes, activation='softmax')(x) # 클래스 수에 맞게 출력
19 model = keras.Model(inputs=inputs, outputs=outputs)
20
21 # 모델 컴파일
22 model.compile(loss='categorical_crossentropy',
23               optimizer='adam',
24               metrics=['accuracy'])
25
26 # 모델 훈련
27 model.summary()

```

Model: "model_1"

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 150, 150, 3)]	0
conv2d_8 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_9 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_9 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_10 (Conv2D)	(None, 34, 34, 128)	73536
max_pooling2d_10 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_11 (Conv2D)	(None, 16, 16, 256)	295168
max_pooling2d_11 (MaxPooling2D)	(None, 7, 7, 256)	0
flatten_2 (Flatten)	(None, 12844)	0
dropout_2 (Dropout)	(None, 12844)	0
dense_1 (Dense)	(None, 80)	87680

Total params: 784788 (2.92 MB)
Trainable params: 784788 (2.92 MB)

입력 이미지의 형태를 정의합니다. 여기서는 150x150 크기의 컬러 이미지(RGB)를 입력으로 받습니다. Conv2D 레이어는 3x3 크기의 필터 32개를 사용하여 입력 이미지에서 특징을 추출합니다. 활성화 함수로 ReLU를 사용합니다.

MaxPooling2D 레이어는 2x2 크기의 풀링을 적용하여 이미지를 축소합니다. 여기서는 50%의 뉴런을 드롭아웃합니다.

데이터 전처리 (image_dataset_from_directory 사용)

```

[ ] 1 from tensorflow.keras.utils import image_dataset_from_directory
2
3 train_dataset = image_dataset_from_directory(
4     train_dir,
5     image_size=(150, 150),
6     batch_size=32,
7     label_mode='categorical'
8 )
9 validation_dataset = image_dataset_from_directory(
10     validation_dir,
11     image_size=(150, 150),
12     batch_size=32,
13     label_mode='categorical'
14 )
15 test_dataset = image_dataset_from_directory(
16     test_dir,
17     image_size=(150, 150),
18     batch_size=32,
19     label_mode='categorical'
20 )
21

```

Found 4793 files belonging to 30 classes.
Found 150 files belonging to 30 classes.
Found 150 files belonging to 30 classes.

```

1 # 콜백 설정
2 callbacks = [
3     keras.callbacks.ModelCheckpoint(
4         filepath='model.h5',
5         save_best_only=True,
6         monitor='val_loss',
7     ),
8     keras.callbacks.EarlyStopping(
9         monitor='val_loss',
10        patience=10,
11        restore_best_weights=True
12    )
13 ]
14 # 모델 훈련
15 history = model.fit(
16     train_dataset,
17     validation_data=validation_dataset,
18     callbacks=callbacks,
19     epochs=20
20 )

```

Epoch 1/20
150/150 [=====] - 12s 43ms/step - loss: 4.9215 - accuracy: 0.1438 - val_loss: 2.3315 - val_accuracy: 0.4067
Epoch 2/20
150/150 [=====] - 12s 43ms/step - loss: 2.3383 - accuracy: 0.3591 - val_loss: 1.6043 - val_accuracy: 0.5067
Epoch 3/20
150/150 [=====] - 12s 43ms/step - loss: 1.8160 - accuracy: 0.4966 - val_loss: 1.4829 - val_accuracy: 0.5200
Epoch 4/20
150/150 [=====] - 12s 43ms/step - loss: 1.4750 - accuracy: 0.5767 - val_loss: 1.1290 - val_accuracy: 0.6867
Epoch 5/20
150/150 [=====] - 12s 43ms/step - loss: 1.1902 - accuracy: 0.6549 - val_loss: 1.2185 - val_accuracy: 0.6800
Epoch 6/20
150/150 [=====] - 12s 43ms/step - loss: 0.9397 - accuracy: 0.7213 - val_loss: 1.2471 - val_accuracy: 0.6267
Epoch 7/20
150/150 [=====] - 12s 43ms/step - loss: 0.7972 - accuracy: 0.7613 - val_loss: 1.1581 - val_accuracy: 0.7067
Epoch 8/20
150/150 [=====] - 12s 43ms/step - loss: 0.6763 - accuracy: 0.7903 - val_loss: 1.2632 - val_accuracy: 0.6467
Epoch 9/20
150/150 [=====] - 12s 43ms/step - loss: 0.5957 - accuracy: 0.8298 - val_loss: 1.4483 - val_accuracy: 0.6733
Epoch 10/20
150/150 [=====] - 12s 43ms/step - loss: 0.5361 - accuracy: 0.8329 - val_loss: 1.5097 - val_accuracy: 0.6800
Epoch 11/20
150/150 [=====] - 12s 43ms/step - loss: 0.5038 - accuracy: 0.8460 - val_loss: 1.4081 - val_accuracy: 0.7000
Epoch 12/20
150/150 [=====] - 12s 43ms/step - loss: 0.4613 - accuracy: 0.8625 - val_loss: 1.5874 - val_accuracy: 0.6733
Epoch 13/20
150/150 [=====] - 12s 43ms/step - loss: 0.3256 - accuracy: 0.8953 - val_loss: 1.7489 - val_accuracy: 0.6667
Epoch 14/20
150/150 [=====] - 12s 43ms/step - loss: 0.3472 - accuracy: 0.8985 - val_loss: 1.7976 - val_accuracy: 0.6867

훈련 정확도와 손실 그래프 그리기

```
[ ] 1 import matplotlib.pyplot as plt
2
3 # 훈련 및 검증 손실 그래프
4 plt.figure(figsize=(12, 4))
5 plt.subplot(1, 2, 1)
6 plt.plot(history.history['loss'], label='Train Loss')
7 plt.plot(history.history['val_loss'], label='Validation Loss')
8 plt.xlabel('Epochs')
9 plt.ylabel('Loss')
10 plt.legend()
11 plt.title('Training and Validation Loss')
12
13 # 훈련 및 검증 정확도 그래프
14 plt.subplot(1, 2, 2)
15 plt.plot(history.history['accuracy'], label='Train Accuracy')
16 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
17 plt.xlabel('Epochs')
18 plt.ylabel('Accuracy')
19 plt.legend()
20 plt.title('Training and Validation Accuracy')
21
22 plt.show()
23
```



테스트 세트에서 모델 평가

```
[ ] 1 test_loss, test_accuracy = model.evaluate(test_dataset)
2 print(f"Test Loss: {test_loss}")
3 print(f"Test Accuracy: {test_accuracy}")
4
```

```
5/5 [=====] - 0s 11ms/step - loss: 1.2889 - accuracy: 0.6467
Test Loss: 1.2889268398284912
Test Accuracy: 0.6466666460037231
```

기본 모델을 학습 했을 때에는 0.6의 정확도를 갖는다.

데이터 증식 정의

```
[ ] 1 from tensorflow.keras.preprocessing.image import ImageDataGenerator
2 import numpy as np
3 import matplotlib.pyplot as plt
4 datagen = ImageDataGenerator(
5     rotation_range=40,
6     width_shift_range=0.2,
7     height_shift_range=0.2,
8     shear_range=0.2,
9     zoom_range=0.2,
10    horizontal_flip=True,
11    fill_mode='nearest'
12 )
13
14 # 데이터 증식 예시 출력
15 sample_image = next(iter(train_dataset))[0][0].numpy().astype('uint8')
16 sample_image = np.expand_dims(sample_image, 0)
17 plt.figure(figsize=(12, 12))
18 for i in range(9):
19     plt.subplot(3, 3, i + 1)
20     augmented_image = datagen.flow(sample_image, batch_size=1).next()[0].astype('uint8')
21     plt.imshow(augmented_image)
22     plt.axis('off')
23 plt.show()
24
```



컨브넷 정의 및 훈련

```
1 # 모델 구성
2 inputs = keras.Input(shape=(150, 150, 3))
3 x = layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical")(inputs)
4 x = layers.experimental.preprocessing.RandomRotation(0.2)(x)
5 x = layers.experimental.preprocessing.RandomZoom(0.2)(x)
6 x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
7 x = layers.MaxPooling2D(pool_size=2)(x)
8 x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
9 x = layers.MaxPooling2D(pool_size=2)(x)
10 x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
11 x = layers.MaxPooling2D(pool_size=2)(x)
12 x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
13 x = layers.MaxPooling2D(pool_size=2)(x)
14 x = layers.Flatten()(x)
15 x = layers.Dropout(0.5)(x)
16 outputs = layers.Dense(num_classes, activation="softmax")(x)
17 model = keras.Model(inputs=inputs, outputs=outputs)
18
19 # 모델 컴파일
20 model.compile(loss="categorical_crossentropy",
21               optimizer="adam",
22               metrics=["accuracy"])
23
24 # 모델 요약
25 model.summary()
26
27 # 모델 훈련
28 history = model.fit(
29     train_dataset,
30     epochs=30,
31     validation_data=validation_dataset,
32     callbacks=callbacks,
33 )
34
```

출력층의 노드 수와 활성화 함수 변경: 클래스의 수에 맞게 출력 노드 수를 설정하고, sigmoid 대신 softmax를 사용합니다.

```
Epoch 9/30
150/150 [=====] - 5s 32ms/step - loss: 2.2786 - accuracy: 0.3507 - val_loss: 1.8555 - val_accuracy: 0.4800
Epoch 10/30
150/150 [=====] - 6s 39ms/step - loss: 2.2385 - accuracy: 0.3624 - val_loss: 1.7333 - val_accuracy: 0.5200
Epoch 11/30
150/150 [=====] - 6s 41ms/step - loss: 2.2228 - accuracy: 0.3745 - val_loss: 1.8668 - val_accuracy: 0.5200
Epoch 12/30
150/150 [=====] - 7s 43ms/step - loss: 2.2086 - accuracy: 0.3687 - val_loss: 1.8349 - val_accuracy: 0.4733
Epoch 13/30
150/150 [=====] - 5s 30ms/step - loss: 2.1158 - accuracy: 0.3922 - val_loss: 1.4297 - val_accuracy: 0.6267
Epoch 14/30
150/150 [=====] - 6s 40ms/step - loss: 2.0690 - accuracy: 0.4081 - val_loss: 1.7411 - val_accuracy: 0.5533
Epoch 15/30
150/150 [=====] - 4s 28ms/step - loss: 2.0583 - accuracy: 0.4146 - val_loss: 1.4597 - val_accuracy: 0.6000
Epoch 16/30
150/150 [=====] - 5s 31ms/step - loss: 2.0196 - accuracy: 0.4227 - val_loss: 1.6131 - val_accuracy: 0.5800
Epoch 17/30
150/150 [=====] - 6s 37ms/step - loss: 1.9597 - accuracy: 0.4413 - val_loss: 1.2079 - val_accuracy: 0.6733
Epoch 18/30
150/150 [=====] - 5s 23ms/step - loss: 1.9553 - accuracy: 0.4459 - val_loss: 1.4691 - val_accuracy: 0.6200
Epoch 19/30
150/150 [=====] - 6s 36ms/step - loss: 1.9256 - accuracy: 0.4482 - val_loss: 1.3335 - val_accuracy: 0.6533
Epoch 20/30
150/150 [=====] - 5s 33ms/step - loss: 1.9186 - accuracy: 0.4571 - val_loss: 1.4057 - val_accuracy: 0.6867
Epoch 21/30
150/150 [=====] - 5s 29ms/step - loss: 1.8781 - accuracy: 0.4548 - val_loss: 1.3388 - val_accuracy: 0.6533
Epoch 22/30
150/150 [=====] - 6s 41ms/step - loss: 1.8521 - accuracy: 0.4711 - val_loss: 1.4254 - val_accuracy: 0.6400
Epoch 23/30
150/150 [=====] - 4s 29ms/step - loss: 1.8435 - accuracy: 0.4628 - val_loss: 1.1346 - val_accuracy: 0.7400
Epoch 24/30
150/150 [=====] - 5s 23ms/step - loss: 1.7985 - accuracy: 0.4805 - val_loss: 1.3726 - val_accuracy: 0.5867
Epoch 25/30
150/150 [=====] - 6s 41ms/step - loss: 1.7872 - accuracy: 0.4838 - val_loss: 1.1456 - val_accuracy: 0.7267
Epoch 26/30
150/150 [=====] - 5s 23ms/step - loss: 1.7947 - accuracy: 0.4882 - val_loss: 1.2423 - val_accuracy: 0.6800
Epoch 27/30
150/150 [=====] - 5s 31ms/step - loss: 1.7723 - accuracy: 0.4888 - val_loss: 1.0881 - val_accuracy: 0.7067
Epoch 28/30
150/150 [=====] - 6s 36ms/step - loss: 1.7146 - accuracy: 0.5095 - val_loss: 1.3347 - val_accuracy: 0.6200
Epoch 29/30
150/150 [=====] - 5s 29ms/step - loss: 1.7044 - accuracy: 0.5091 - val_loss: 1.2085 - val_accuracy: 0.6933
Epoch 30/30
150/150 [=====] - 6s 38ms/step - loss: 1.6950 - accuracy: 0.5089 - val_loss: 1.0854 - val_accuracy: 0.7199
```

정확도가 0.6~0.7을 왔다 갔다 하는 것을 볼 수 있다.

테스트 세트에서 모델 평가

```
1 test_loss, test_accuracy = model.evaluate(test_dataset)
2 print(f"Test Loss: {test_loss}")
3 print(f"Test Accuracy: {test_accuracy}")
4
```

```
5/5 [=====] - 0s 20ms/step - loss: 1.3215 - accuracy: 0.6667
Test Loss: 1.3214656114578247
Test Accuracy: 0.66666666665348816
```

✓ 2.전이학습

** VGG16 합성곱 기반 층 만들기**

```
1 from tensorflow.keras.applications import VGG16
2 from tensorflow.keras.applications.vgg16 import preprocess_input
3
4 # VGG16 모델 불러오기
5 conv_base = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
6
```

** VGG16 특성과 해당 레이블 추출하기**

```
[ ] 1 import numpy as np
2
3 def get_features_and_labels(dataset):
4     all_features = []
5     all_labels = []
6     for images, labels in dataset:
7         # 이미지 크기 조정 및 전처리
8         resized_images = tf.image.resize(images, (224, 224))
9         preprocessed_images = preprocess_input(resized_images)
10
11         # 특성 추출
12         features = conv_base.predict(preprocessed_images)
13         all_features.append(features)
14         all_labels.append(labels)
15     return np.concatenate(all_features), np.concatenate(all_labels)
16
17 # 훈련 데이터셋
18 train_features, train_labels = get_features_and_labels(train_dataset)
19 # 검증 데이터셋
20 val_features, val_labels = get_features_and_labels(validation_dataset)
21 # 테스트 데이터셋
22 test_features, test_labels = get_features_and_labels(test_dataset)
23
24 print(train_features.shape)
```

**** 밀집 연결 분류기 정의하고 훈련하기****

```
1 from tensorflow.keras import layers, models, optimizers
2
3 # 밀집 연결 분류기 정의
4 inputs = layers.Input(shape=train_features.shape[1:])
5 x = layers.Flatten()(inputs)
6 x = layers.Dense(256, activation='relu')(x)
7 x = layers.Dropout(0.5)(x)
8 outputs = layers.Dense(num_classes, activation="softmax")(x)
9
10 # 모델 정의 및 컴파일
11 model = models.Model(inputs, outputs)
12 model.compile(loss="categorical_crossentropy",
13               optimizer=optimizers.RMSprop(),
14               metrics=["accuracy"])
15
16 # 콜백 정의
17 callbacks = [
18     keras.callbacks.ModelCheckpoint(
19         filepath="feature_extraction.h5",
20         save_best_only=True,
21         monitor="val_loss")
22 ]
23
24 # 모델 훈련
25 history = model.fit(
26     train_features, train_labels,
27     epochs=20,
28     validation_data=(val_features, val_labels),
29     callbacks=callbacks)
30
31
```

결과를 그래프로 나타내기

```
1 import matplotlib.pyplot as plt
2
3 # 정확도 그래프
4 plt.plot(history.history["accuracy"], "bo", label="Training accuracy")
5 plt.plot(history.history["val_accuracy"], "b", label="Validation accuracy")
6 plt.title("Training and validation accuracy")
7 plt.xlabel("Epochs")
8 plt.ylabel("Accuracy")
9 plt.legend()
10 plt.show()
11
12 # 손실 그래프
13 plt.plot(history.history["loss"], "bo", label="Training loss")
14 plt.plot(history.history["val_loss"], "b", label="Validation loss")
15 plt.title("Training and validation loss")
16 plt.xlabel("Epochs")
17 plt.ylabel("Loss")
18 plt.legend()
19 plt.show()
20
21 # 테스트 세트에서 모델 평가
22 test_model = keras.models.load_model("feature_extraction.h5")
23 test_loss, test_acc = test_model.evaluate(test_features, test_labels)
24 print(f"테스트 정확도: {test_acc:.3f}")
25
```

밀집 연결 분류기 정의: 이미지 분류 문제를 해결하기 위해 밀집 연결 신경망(dense neural network)을 정의합니다. 이 분류기는 사전에 추출된 이미지 특성(feature)을 입력으로 받아 각 이미지가 속할 클래스를 예측합니다.

모델 정의 및 컴파일: 정의된 밀집 연결 분류기를 포함하는 전체 모델을 구성하고, 이 모델을 컴파일합니다. 컴파일 과정에서는 손실 함수로 `categorical_crossentropy`를, 옵티마이저로 `RMSprop`을 선택하여 모델을 최적화합니다. 또한 모델의 성능을 평가할 지표로 정확도를 설정합니다.

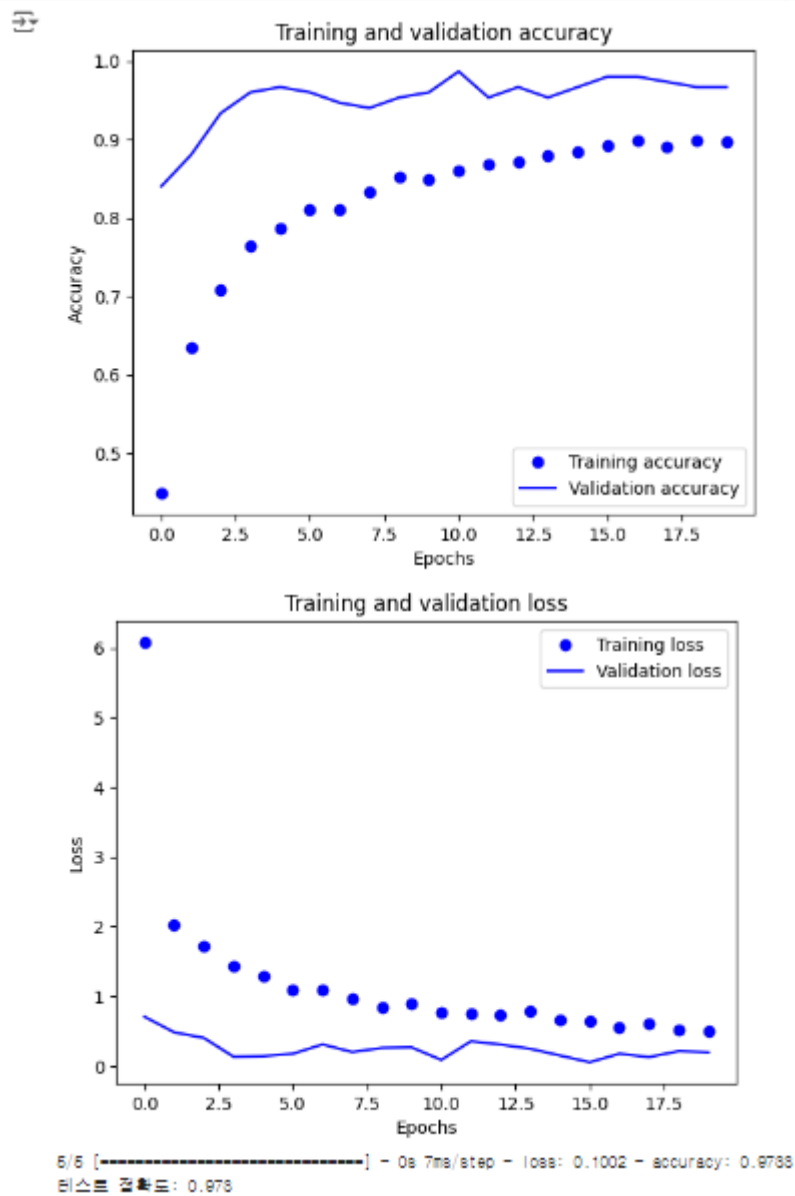
손실 함수 변경: `binary_crossentropy` 대신 `categorical_crossentropy`를 사용합니다.

콜백 정의: 모델 훈련 중에 필요한 콜백(callback)을 정의합니다. 여기서는 검증 손실(`val_loss`)을 모니터링하고, 검증 손실이 가장 낮을 때 모델을 저장하는 `ModelCheckpoint` 콜백을 사용합니다.

모델 훈련: 정의된 모델을 훈련 데이터셋으로 훈련합니다. 훈련 데이터셋을 사용하여 모델이 이미지를 올바르게 분류할 수 있도록 학습시키고, 검증 데이터셋을 사용하여 모델의 성능을 평가합니다. 훈련 과정에서 정의한 콜백이 활용됩니다.

결과 시각화: 훈련 과정에서 얻은 결과를 시각화하여 모델의 훈련과 검증 세트에 대한 정확도와 손실의 변화를 관찰할 수 있습니다. 이를 통해 모델이 훈련 중에 어떻게 성능을 향상시키는지 시각적으로 확인할 수 있습니다.

테스트 세트 평가: 최종적으로 테스트 데이터셋을 사용하여 훈련된 모델의 성능을 평가합니다. 이를 통해 모델이 새로운 데이터에 대해 얼마나 잘 일반화되는지를 확인할 수 있습니다.



전이학습을 한 이후에는 0.6-7에 이르던 것이 0.975까지 오른 것을 알 수 있다.