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

```
Mounted at /content/gdrive/
```

```
import zipfile

# 파일 경로 입력
zip_file_name = '/content/drive/MyDrive/Experiment_6/dataset_directory.zip'

# 압축 해제할 경로 입력
extraction_dir = '/content/dataset'

# 압축 해제
with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
  zip_ref.extractall(extraction_dir)
```

```


import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# ResNet50 모델 불러오기 (ImageNet 가중치 사용, 최상위 레이어 제외)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

# 기본 모델의 가중치 동결 (Feature Extraction)
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# 출력층 추가
x = base_model.output
x = GlobalAveragePooling2D()(x)  # Global Average Pooling
x = Dropout(0.5)(x)  # 과적합 방지
x = Dense(128, activation='relu')(x)  # Fully Connected Layer
predictions = Dense(4, activation='softmax')(x)  # 날씨 클래스: 비, 일출, 맑음, 먹구름

# 최종 모델 정의
model = Model(inputs=base_model.input, outputs=predictions)
```

```
# 모델 컴파일
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000012),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 데이터 증강 (Train/Test Split)
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    validation_split=0.2  # train 데이터의 20%를 validation으로 사용
)

test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

# Train 데이터 생성 (subset='training')
train_generator = train_datagen.flow_from_directory(
    '/content/dataset/train',
    target_size=(128, 128),
    batch_size=128,
    class_mode='categorical',
    subset='training',  # Train 데이터 서브셋
    shuffle=True
)

# Validation 데이터 생성 (subset='validation')
validation_generator = train_datagen.flow_from_directory(
    '/content/dataset/train',
    target_size=(128, 128),
    batch_size=128,
    class_mode='categorical',
    subset='validation',  # Validation 데이터 서브셋
    shuffle=False
)
```

```
# Test 데이터 생성
test_generator = test_datagen.flow_from_directory(
    '/content/dataset/test',
    target_size=(128, 128),
    batch_size=128,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 5120 images belonging to 4 classes.
Found 1280 images belonging to 4 classes.
Found 640 images belonging to 4 classes.
```

```
# 모델 학습
model.fit(
    train_generator,
    epochs=10,
    batch_size=128
)
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should ca
  self._warn_if_super_not_called()
40/40 ━━━━━━━━━━━━━━━━━━━━ 25s 212ms/step - accuracy: 0.2550 - loss: 2.7681
Epoch 2/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 210ms/step - accuracy: 0.3269 - loss: 2.1334
Epoch 3/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 238ms/step - accuracy: 0.4218 - loss: 1.6667
Epoch 4/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 240ms/step - accuracy: 0.4876 - loss: 1.4081
Epoch 5/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 10s 209ms/step - accuracy: 0.5434 - loss: 1.1727
Epoch 6/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 211ms/step - accuracy: 0.5884 - loss: 1.0256
Epoch 7/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 244ms/step - accuracy: 0.6701 - loss: 0.8426
Epoch 8/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 10s 222ms/step - accuracy: 0.7048 - loss: 0.7608
Epoch 9/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 10s 200ms/step - accuracy: 0.7361 - loss: 0.6679
Epoch 10/10
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 238ms/step - accuracy: 0.7580 - loss: 0.6070
<keras.src.callbacks.history.History at 0x79b82006ebc0>
```

```
# 상위 몇 층을 학습 가능하도록 설정
base_model.trainable = True
for layer in base_model.layers[:-50]:  # 마지막 50개 층만 학습
    layer.trainable = False

# 재컴파일
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000012),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fine-Tuning 학습
history = model.fit(
    train_generator,
    epochs=9,
    batch_size=128
)
```

```
Epoch 1/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 39s 261ms/step - accuracy: 0.7168 - loss: 0.7164
Epoch 2/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 233ms/step - accuracy: 0.9164 - loss: 0.2840
Epoch 3/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 22s 265ms/step - accuracy: 0.9605 - loss: 0.1455
Epoch 4/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 12s 263ms/step - accuracy: 0.9854 - loss: 0.0786
Epoch 5/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 12s 271ms/step - accuracy: 0.9847 - loss: 0.0643
Epoch 6/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 248ms/step - accuracy: 0.9934 - loss: 0.0405
Epoch 7/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 270ms/step - accuracy: 0.9972 - loss: 0.0298
Epoch 8/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 12s 258ms/step - accuracy: 0.9966 - loss: 0.0232
Epoch 9/9
40/40 ━━━━━━━━━━━━━━━━━━━━ 11s 242ms/step - accuracy: 0.9987 - loss: 0.0162
```

```
# Performance Evaluate
test_loss, test_acc = model.evaluate(test_generator)
```

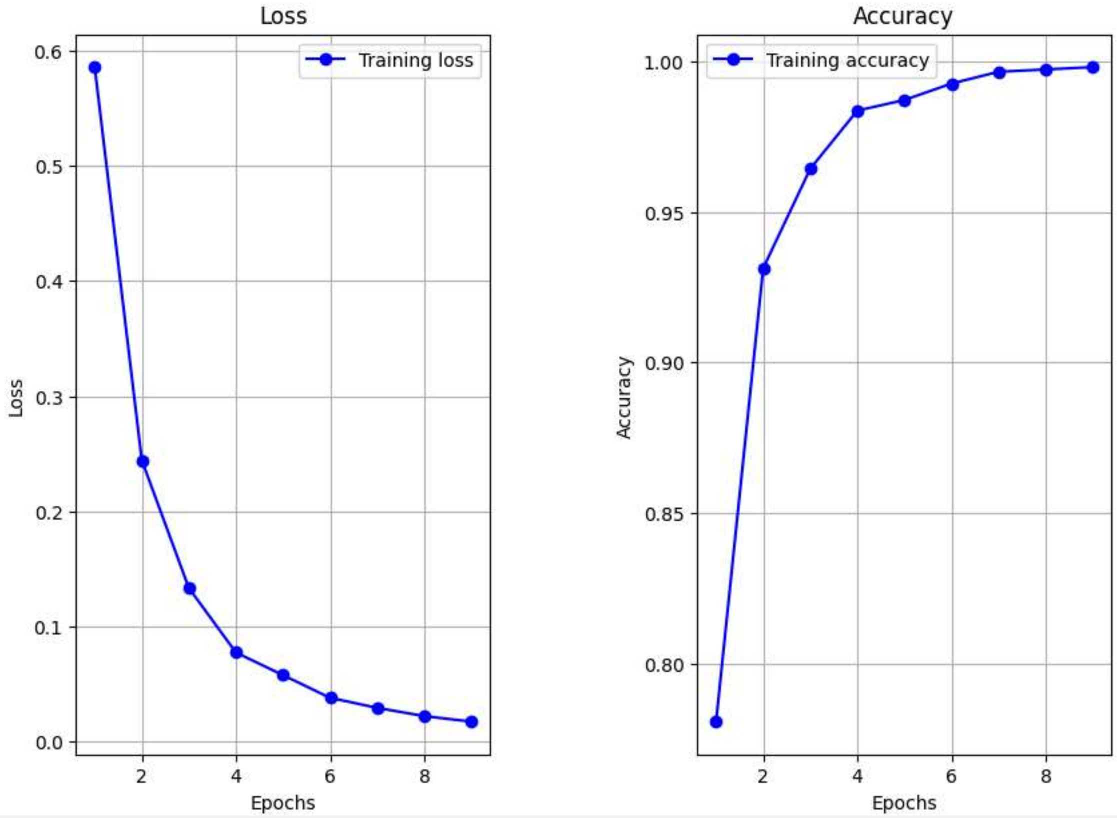
```
5/5 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 165ms/step - accuracy: 0.9619 - loss: 0.1899
```

```
print('test_loss:            ', test_loss)
print('test_accuracy:          ', test_acc)
```

```
test_loss:            0.16169561445713043
test_accuracy:          0.9609375
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
import matplotlib.pyplot as plt

# Example history object for demonstration purposes
# history.history = {'loss': [0.5, 0.4, 0, 0.3], 'accuracy': [0.7, 0.8, 0, 0.85]}

# 원본 데이터
loss = history.history['loss']
acc = history.history['accuracy']

# 0 값을 제외한 데이터 필터링
filtered_loss = [l for l in loss if l != 0]
filtered_acc = [a for a in acc if a != 0]

# 필터링된 에포크 계산
epochs = range(1, len(filtered_loss) + 1)

# 그래프 그리기
plt.figure(figsize=(10, 7))
plt.subplots_adjust(wspace=0.5)

plt.subplot(1, 2, 1)
plt.plot(epochs, filtered_loss, 'bo-', label='Training loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid()
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, filtered_acc, 'bo-', label='Training accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.grid()
plt.legend()

plt.show()
```



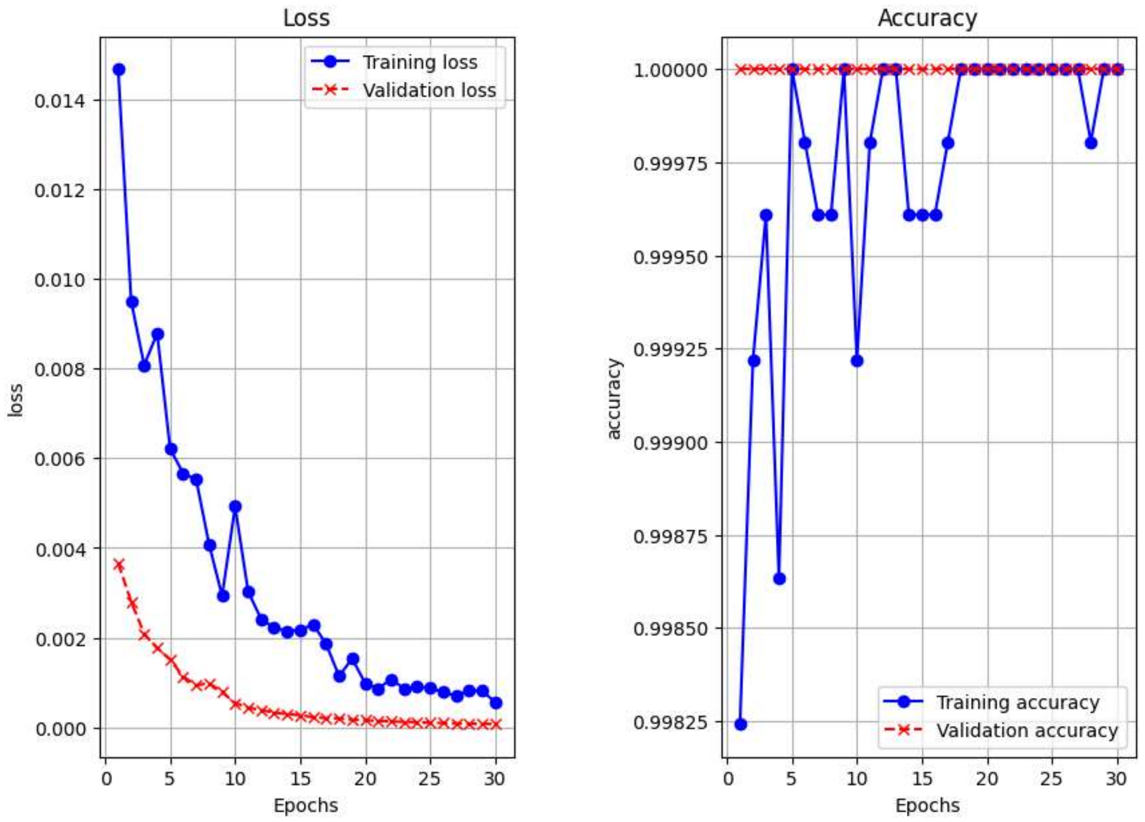
```
history= model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=30,
    batch_size=128
)
```

```
Epoch 2/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 15s 332ms/step - accuracy: 0.9997 - loss: 0.0098 - val_accuracy: 1.0000 - val_loss: 0.0028
Epoch 3/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.9997 - loss: 0.0082 - val_accuracy: 1.0000 - val_loss: 0.0021
Epoch 4/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 317ms/step - accuracy: 0.9988 - loss: 0.0081 - val_accuracy: 1.0000 - val_loss: 0.0018
Epoch 5/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 315ms/step - accuracy: 1.0000 - loss: 0.0063 - val_accuracy: 1.0000 - val_loss: 0.0015
Epoch 6/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 323ms/step - accuracy: 0.9997 - loss: 0.0063 - val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 7/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 314ms/step - accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 1.0000 - val_loss: 9.3860e-04
Epoch 8/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 314ms/step - accuracy: 0.9994 - loss: 0.0045 - val_accuracy: 1.0000 - val_loss: 9.8573e-04
Epoch 9/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 317ms/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 1.0000 - val_loss: 7.8270e-04
Epoch 10/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 305ms/step - accuracy: 0.9994 - loss: 0.0057 - val_accuracy: 1.0000 - val_loss: 5.4141e-04
Epoch 11/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 311ms/step - accuracy: 0.9998 - loss: 0.0029 - val_accuracy: 1.0000 - val_loss: 4.4075e-04
Epoch 12/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 317ms/step - accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 1.0000 - val_loss: 3.8750e-04
Epoch 13/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 327ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 3.2611e-04
Epoch 14/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 307ms/step - accuracy: 0.9997 - loss: 0.0020 - val_accuracy: 1.0000 - val_loss: 2.9734e-04
Epoch 15/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 326ms/step - accuracy: 0.9994 - loss: 0.0026 - val_accuracy: 1.0000 - val_loss: 2.6756e-04
Epoch 16/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 313ms/step - accuracy: 0.9996 - loss: 0.0024 - val_accuracy: 1.0000 - val_loss: 2.2725e-04
Epoch 17/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 315ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 1.0000 - val_loss: 2.0485e-04
Epoch 18/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 311ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 2.0061e-04
Epoch 19/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 309ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 1.0000 - val_loss: 1.7289e-04
Epoch 20/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 309ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 1.5699e-04
Epoch 21/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 313ms/step - accuracy: 1.0000 - loss: 8.7791e-04 - val_accuracy: 1.0000 - val_loss: 1.461
Epoch 22/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 316ms/step - accuracy: 1.0000 - loss: 0.0010 - val_accuracy: 1.0000 - val_loss: 1.3210e-04
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 320ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 1.1288e-04
Epoch 25/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 327ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 1.0271e-04
Epoch 26/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 15s 328ms/step - accuracy: 1.0000 - loss: 7.9092e-04 - val_accuracy: 1.0000 - val_loss: 9.4868
Epoch 27/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 310ms/step - accuracy: 1.0000 - loss: 7.1026e-04 - val_accuracy: 1.0000 - val_loss: 8.7077
Epoch 28/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 323ms/step - accuracy: 0.9996 - loss: 8.9843e-04 - val_accuracy: 1.0000 - val_loss: 8.2989
Epoch 29/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 14s 326ms/step - accuracy: 1.0000 - loss: 6.2353e-04 - val_accuracy: 1.0000 - val_loss: 7.7494
Epoch 30/30
40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 309ms/step - accuracy: 1.0000 - loss: 4.8563e-04 - val_accuracy: 1.0000 - val_loss: 7.2935
```

```
import matplotlib.pyplot as plt

#plots
loss = history.history['loss']
acc = history.history['accuracy']

val_loss = history.history['val_loss']
val_acc = history.history['val_accuracy']


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

plt.figure(figsize=(10,7))
plt.subplots_adjust(wspace=0.5)

plt.subplot(1,2,1)
plt.plot(epochs, loss, 'bo-', label = 'Training loss')
plt.plot(epochs, val_loss, 'rx--', label = 'Validation loss')   # 검증 부분
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.grid()
plt.legend()

plt.subplot(1,2,2)
plt.plot(epochs, acc, 'bo-', label='Training accuracy')
plt.plot(epochs, val_acc, 'rx--', label = 'Validation accuracy')  # 검증 부분
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('accuracy')
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x79b78056b010>
```



```
# 모델 평가
```

```
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {loss}")
print(f"Validation Accuracy: {accuracy}")

# 모델 저장
```