In [1]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator


from keras.utils import to_categorical
```

In [2]:
```python
# Hyperparameters
EPOCHS = 20
LEARNING_RATE = 0.0051
BATCH_SIZE = 128
IMG_SIZE = (128, 128)
LOSS_FUNCTION = 'categorical_crossentropy'
```

In [3]:
```python
# Train and Validation Data Generator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    validation_split=0.2  # 20%를 validation으로 분리
)

# Test Data Generator (no augmentation)
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [4]:
```python
# Train Data (80% of train folder)
train_generator = train_datagen.flow_from_directory(
    '/Users/User/Experiment_6/dataset_directory/train',  # Train 데이터 경로
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'  # Train용 데이터
)

# Validation Data (20% of train folder)
validation_generator = train_datagen.flow_from_directory(
    '/Users/User/Experiment_6/dataset_directory/train',  # Train 데이터 경로
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'  # Validation용 데이터
)

# Test Data (Separate folder)
test_generator = test_datagen.flow_from_directory(
    '/Users/User/Experiment_6/dataset_directory/test',  # Test 데이터 경로
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False  # 테스트 데이터는 순서를 유지
)
```

```python
# CNN Model
model = Sequential([
    # First Convolutional Layer
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)
    MaxPooling2D(pool_size=(2, 2)),

    # Second Convolutional Layer
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Third Convolutional Layer
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Flattening and Fully Connected Layer
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')  # Output layer with 3 classes
])

# Compile Model
model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE),
    loss=LOSS_FUNCTION,
    metrics=['accuracy']
)

print("Original class indices:", train_generator.class_indices)
```

```
Found 5120 images belonging to 4 classes.
Found 1280 images belonging to 4 classes.
Found 640 images belonging to 4 classes.
Original class indices: {'clear': 0, 'cloudy': 1, 'rain': 2, 'sunrise': 3}
```

```
C:\Users\User\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.p
y:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [5]:

```python
print(train_generator.class_indices)
```

```
{'clear': 0, 'cloudy': 1, 'rain': 2, 'sunrise': 3}
```

In [6]:

```python
# Model Training
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=EPOCHS,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_steps=validation_generator.samples // BATCH_SIZE
)
"""
history = model.fit(train_generator,
    validation_data=validation_generator,
```

```
              epochs = 50, batch_size = 256,
              validation_split=0.2)


validation_split=0.2



"""



# Model Evaluation
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Epoch 1/20
C:\Users\User\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
40/40 ━━━━━━━━━━━━━━━━━━━━ 60s 1s/step - accuracy: 0.4969 - loss: 2.0756 - val_accuracy: 0.8547 - val_loss: 0.4245
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 3/20
C:\Users\User\anaconda3\Lib\contextlib.py:158: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self.gen.throw(value)
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 36s 763ms/step - accuracy: 0.8411 - loss: 0.4336 - val_ac
curacy: 0.8687 - val_loss: 0.3380
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 520ms/step - accuracy: 0.8949 - loss: 0.3174 - val_ac
curacy: 0.9281 - val_loss: 0.2126
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 600us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 524ms/step - accuracy: 0.9165 - loss: 0.2757 - val_ac
curacy: 0.9406 - val_loss: 0.1609
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 535ms/step - accuracy: 0.9247 - loss: 0.2200 - val_ac
curacy: 0.9516 - val_loss: 0.1389
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 850us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 24s 563ms/step - accuracy: 0.9419 - loss: 0.1753 - val_ac
curacy: 0.9477 - val_loss: 0.1318
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 24s 556ms/step - accuracy: 0.9518 - loss: 0.1446 - val_ac
curacy: 0.9258 - val_loss: 0.2104
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 650us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 527ms/step - accuracy: 0.9361 - loss: 0.1859 - val_ac
curacy: 0.9625 - val_loss: 0.1070
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 675us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 540ms/step - accuracy: 0.9538 - loss: 0.1394 - val_ac
curacy: 0.9414 - val_loss: 0.1482
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 668us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 526ms/step - accuracy: 0.9368 - loss: 0.1891 - val_ac
curacy: 0.9594 - val_loss: 0.1279
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 243ms/step - accuracy: 0.9323 - loss: 0.2450
Test Loss: 0.2439723014831543
Test Accuracy: 0.9375
```

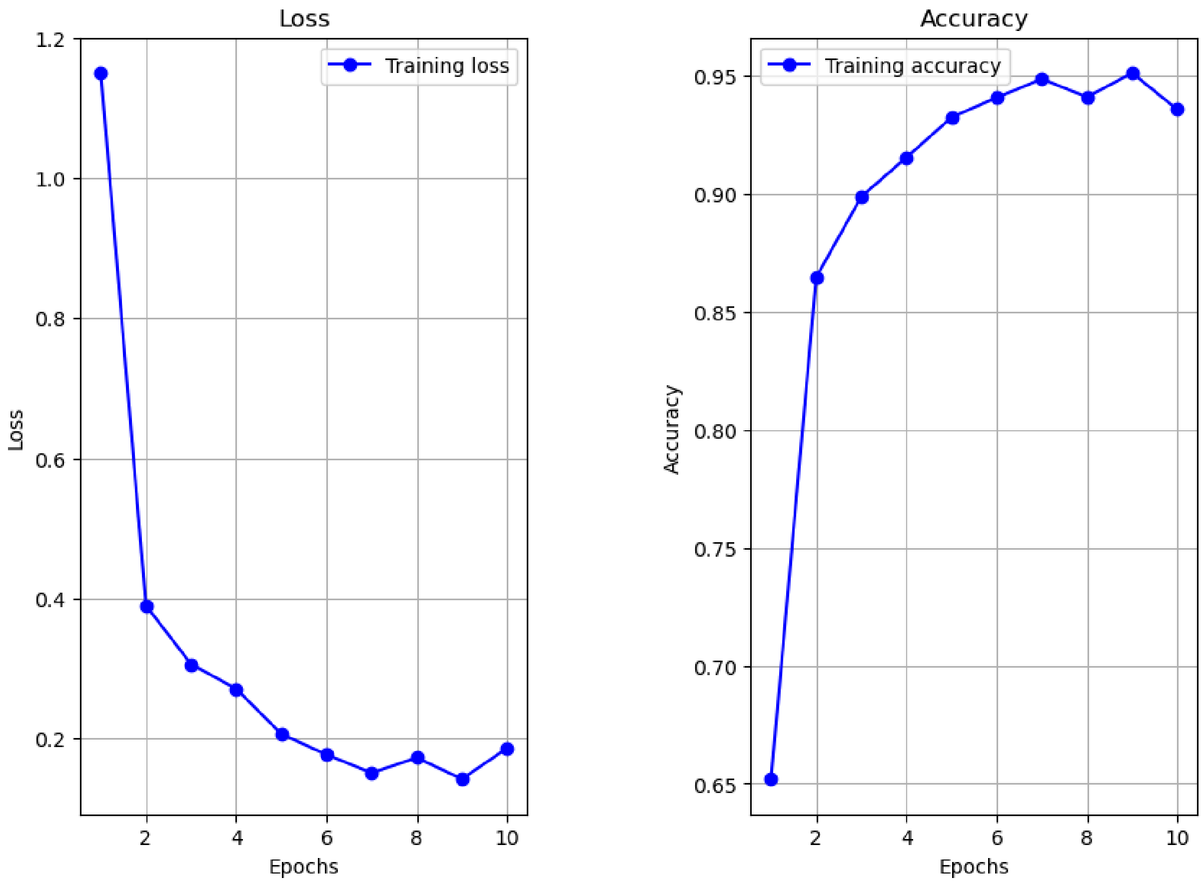
In [7]:

```
print(train_generator.class_indices)
print(validation_generator.class_indices)
print(test_generator.class_indices)
```

```
{'clear': 0, 'cloudy': 1, 'rain': 2, 'sunrise': 3}
{'clear': 0, 'cloudy': 1, 'rain': 2, 'sunrise': 3}
{'clear': 0, 'cloudy': 1, 'rain': 2, 'sunrise': 3}
```

In [8]:

```python
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()
```



In [9]:
```
history = model.fit(train_generator,
    validation_data=validation_generator,
                epochs = 50, batch_size = 128,
                validation_split=0.2)
```

```
Epoch 1/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 548ms/step - accuracy: 0.9482 - loss: 0.1609 - val_ac
curacy: 0.9672 - val_loss: 0.0884
Epoch 2/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9590 - loss: 0.1167 - val_accur
acy: 0.9711 - val_loss: 0.0803
Epoch 3/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9628 - loss: 0.1179 - val_accur
acy: 0.9680 - val_loss: 0.0861
Epoch 4/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 45s 977ms/step - accuracy: 0.9658 - loss: 0.1059 - val_ac
curacy: 0.9820 - val_loss: 0.0556
Epoch 5/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 531ms/step - accuracy: 0.9704 - loss: 0.0901 - val_ac
curacy: 0.9789 - val_loss: 0.0612
Epoch 6/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 517ms/step - accuracy: 0.9732 - loss: 0.0856 - val_ac
curacy: 0.9891 - val_loss: 0.0380
Epoch 7/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 542ms/step - accuracy: 0.9758 - loss: 0.0728 - val_ac
curacy: 0.9758 - val_loss: 0.0697
Epoch 8/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 537ms/step - accuracy: 0.9823 - loss: 0.0629 - val_ac
curacy: 0.9797 - val_loss: 0.0540
Epoch 9/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 536ms/step - accuracy: 0.9724 - loss: 0.0770 - val_ac
curacy: 0.9867 - val_loss: 0.0383
Epoch 10/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 516ms/step - accuracy: 0.9818 - loss: 0.0572 - val_ac
curacy: 0.9914 - val_loss: 0.0330
Epoch 11/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 539ms/step - accuracy: 0.9802 - loss: 0.0638 - val_ac
curacy: 0.9875 - val_loss: 0.0417
Epoch 12/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 24s 537ms/step - accuracy: 0.9740 - loss: 0.0810 - val_ac
curacy: 0.9883 - val_loss: 0.0491
Epoch 13/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 25s 565ms/step - accuracy: 0.9736 - loss: 0.0734 - val_ac
curacy: 0.9805 - val_loss: 0.0581
Epoch 14/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9722 - loss: 0.0940 - val_accur
acy: 0.9945 - val_loss: 0.0217
Epoch 15/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - accuracy: 0.9814 - loss: 0.0534 - val_accur
acy: 0.9820 - val_loss: 0.0423
Epoch 16/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step - accuracy: 0.9825 - loss: 0.0572 - val_accur
acy: 0.9945 - val_loss: 0.0124
Epoch 17/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step - accuracy: 0.9844 - loss: 0.0367 - val_accur
acy: 0.9922 - val_loss: 0.0163
Epoch 18/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - accuracy: 0.9804 - loss: 0.0521 - val_accur
acy: 0.9945 - val_loss: 0.0205
Epoch 19/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9858 - loss: 0.0357 - val_accur
```

```
acy: 0.9953 - val_loss: 0.0135
Epoch 20/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9902 - loss: 0.0292 - val_accur
acy: 0.9937 - val_loss: 0.0127
Epoch 21/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.9897 - loss: 0.0311 - val_accur
acy: 0.9961 - val_loss: 0.0132
Epoch 22/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9899 - loss: 0.0313 - val_accur
acy: 0.9922 - val_loss: 0.0189
Epoch 23/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9845 - loss: 0.0604 - val_accur
acy: 0.9828 - val_loss: 0.0412
Epoch 24/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.9862 - loss: 0.0494 - val_accur
acy: 0.9969 - val_loss: 0.0095
Epoch 25/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.9896 - loss: 0.0302 - val_accur
acy: 0.9930 - val_loss: 0.0169
Epoch 26/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 51s 1s/step - accuracy: 0.9837 - loss: 0.0482 - val_accur
acy: 0.9945 - val_loss: 0.0199
Epoch 27/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - accuracy: 0.9903 - loss: 0.0246 - val_accur
acy: 0.9953 - val_loss: 0.0140
Epoch 28/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 52s 1s/step - accuracy: 0.9863 - loss: 0.0400 - val_accur
acy: 0.9930 - val_loss: 0.0215
Epoch 29/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.9879 - loss: 0.0465 - val_accur
acy: 0.9953 - val_loss: 0.0111
Epoch 30/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 51s 1s/step - accuracy: 0.9867 - loss: 0.0358 - val_accur
acy: 0.9953 - val_loss: 0.0095
Epoch 31/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 27s 565ms/step - accuracy: 0.9871 - loss: 0.0314 - val_ac
curacy: 0.9766 - val_loss: 0.0669
Epoch 32/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 45s 1s/step - accuracy: 0.9786 - loss: 0.0757 - val_accur
acy: 0.9883 - val_loss: 0.0428
Epoch 33/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - accuracy: 0.9818 - loss: 0.0548 - val_accur
acy: 0.9898 - val_loss: 0.0327
Epoch 34/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - accuracy: 0.9778 - loss: 0.0763 - val_accur
acy: 0.9836 - val_loss: 0.0339
Epoch 35/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - accuracy: 0.9842 - loss: 0.0519 - val_accur
acy: 0.9984 - val_loss: 0.0071
Epoch 36/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.9878 - loss: 0.0382 - val_accur
acy: 0.9969 - val_loss: 0.0128
Epoch 37/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 56s 1s/step - accuracy: 0.9878 - loss: 0.0338 - val_accur
acy: 0.9961 - val_loss: 0.0086
Epoch 38/50
```

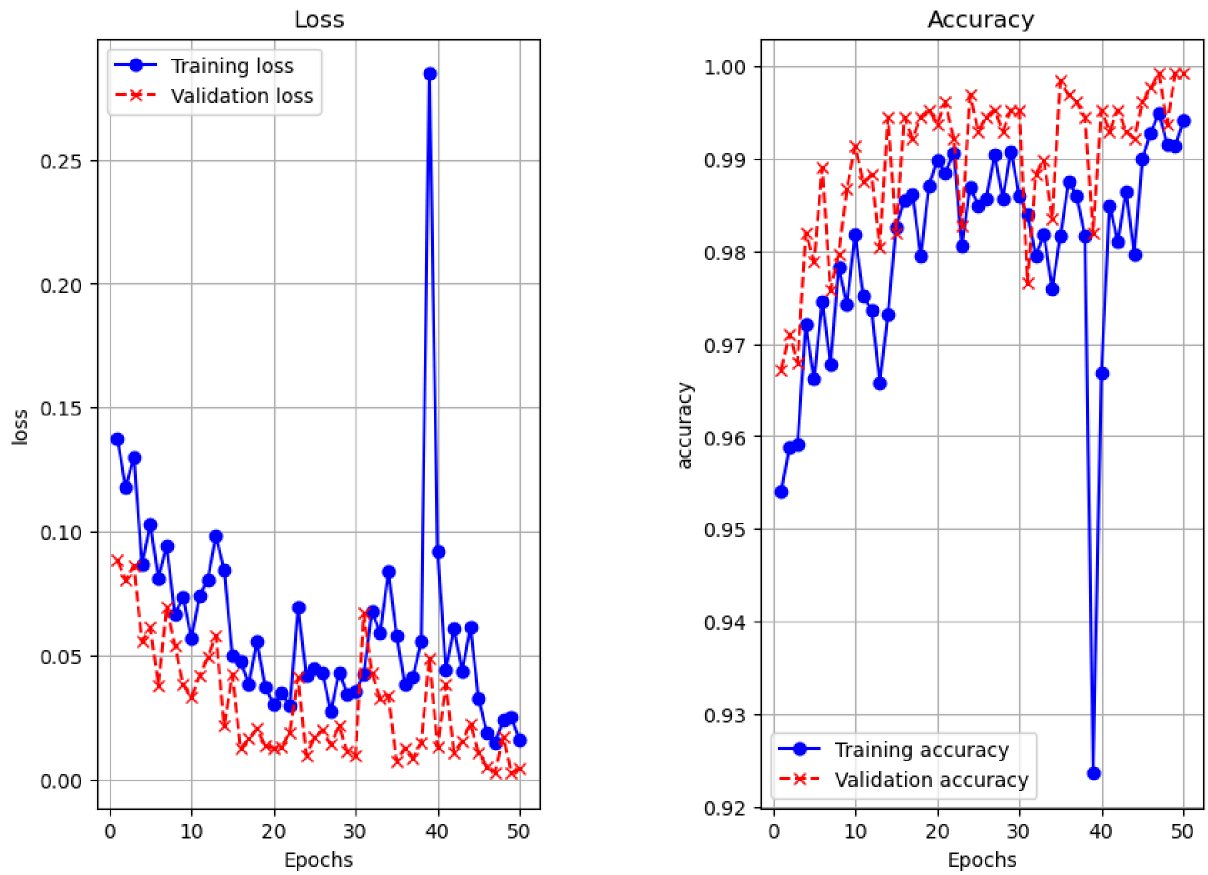
```
40/40 ━━━━━━━━━━━━━━━━━━━━ 55s 1s/step - accuracy: 0.9884 - loss: 0.0395 - val_accur
acy: 0.9945 - val_loss: 0.0147
Epoch 39/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 57s 1s/step - accuracy: 0.9238 - loss: 0.2992 - val_accur
acy: 0.9820 - val_loss: 0.0488
Epoch 40/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9656 - loss: 0.0933 - val_accur
acy: 0.9953 - val_loss: 0.0132
Epoch 41/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9874 - loss: 0.0369 - val_accur
acy: 0.9930 - val_loss: 0.0382
Epoch 42/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9772 - loss: 0.0803 - val_accur
acy: 0.9953 - val_loss: 0.0109
Epoch 43/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 51s 1s/step - accuracy: 0.9895 - loss: 0.0328 - val_accur
acy: 0.9930 - val_loss: 0.0153
Epoch 44/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9786 - loss: 0.0687 - val_accur
acy: 0.9922 - val_loss: 0.0224
Epoch 45/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9904 - loss: 0.0336 - val_accur
acy: 0.9961 - val_loss: 0.0107
Epoch 46/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.9908 - loss: 0.0245 - val_accur
acy: 0.9977 - val_loss: 0.0052
Epoch 47/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.9932 - loss: 0.0168 - val_accur
acy: 0.9992 - val_loss: 0.0030
Epoch 48/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 45s 979ms/step - accuracy: 0.9931 - loss: 0.0189 - val_ac
curacy: 0.9937 - val_loss: 0.0172
Epoch 49/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 47s 1s/step - accuracy: 0.9906 - loss: 0.0258 - val_accur
acy: 0.9992 - val_loss: 0.0026
Epoch 50/50
40/40 ━━━━━━━━━━━━━━━━━━━━ 45s 1s/step - accuracy: 0.9933 - loss: 0.0147 - val_accur
acy: 0.9992 - val_loss: 0.0042
```

In [10]:

```python
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')
```

```python
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()
```

Out[10]: <matplotlib.legend.Legend at 0x1311bb53f50>



In [11]:
```python
model.save('weather_CNN1.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [ ]: