[13]

```python
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.15,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # reserve 20% for validation
)

val_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

print("Defined train_datagen and val_datagen with validation_split=0.2")
```

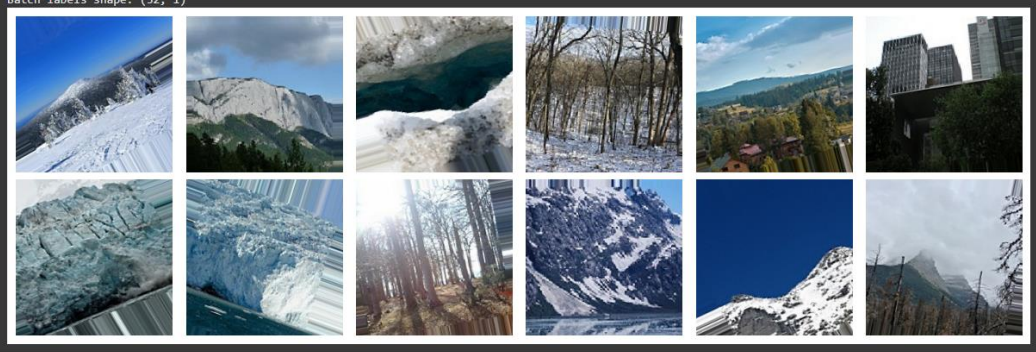
```
Defined train_datagen and val_datagen with validation_split=0.2
```

[14]

```python
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True,
    subset='training',
    seed=SEED
)

validation_generator = val_datagen.flow_from_directory(
    data_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False,
```

```
Batch images shape: (32, 224, 224, 3)
Batch labels shape: (32, 1)
```



Why data augmentation matters

- Prevents overfitting
  Augmentation makes many slightly different versions of each image. This stops the model from memorizing the exact training images and helps it learn real patterns.
- Helps the model work on new images
  Seeing many variations during training teaches the model to handle small changes it will meet in the real world.
- Cheap way to get more data
  You don't need to label more images,augmentation creates extra useful examples from what you already have.

Augmentation is an easy, low-cost trick that makes a fine-tuned model more reliable on small datasets.

```
model.summary()
```

Model: "resnet50_transfer_6cls"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 6)	1,542

Total params: 24,113,798 (91.99 MB)
Trainable params: 526,086 (2.01 MB)
Non-trainable params: 23,587,712 (89.98 MB)

Why we freeze early convolutional layers

- Early layers learn basic visual patterns like edges, colors and simple textures.
- These basic features are useful across many image tasks, so we keep them fixed to avoid destroying that knowledge.
- Freezing speeds up training and reduces overfitting when we only have a small dataset.
- We train only the top (new) layers so the model quickly learns task-specific patterns for the 6 classes.

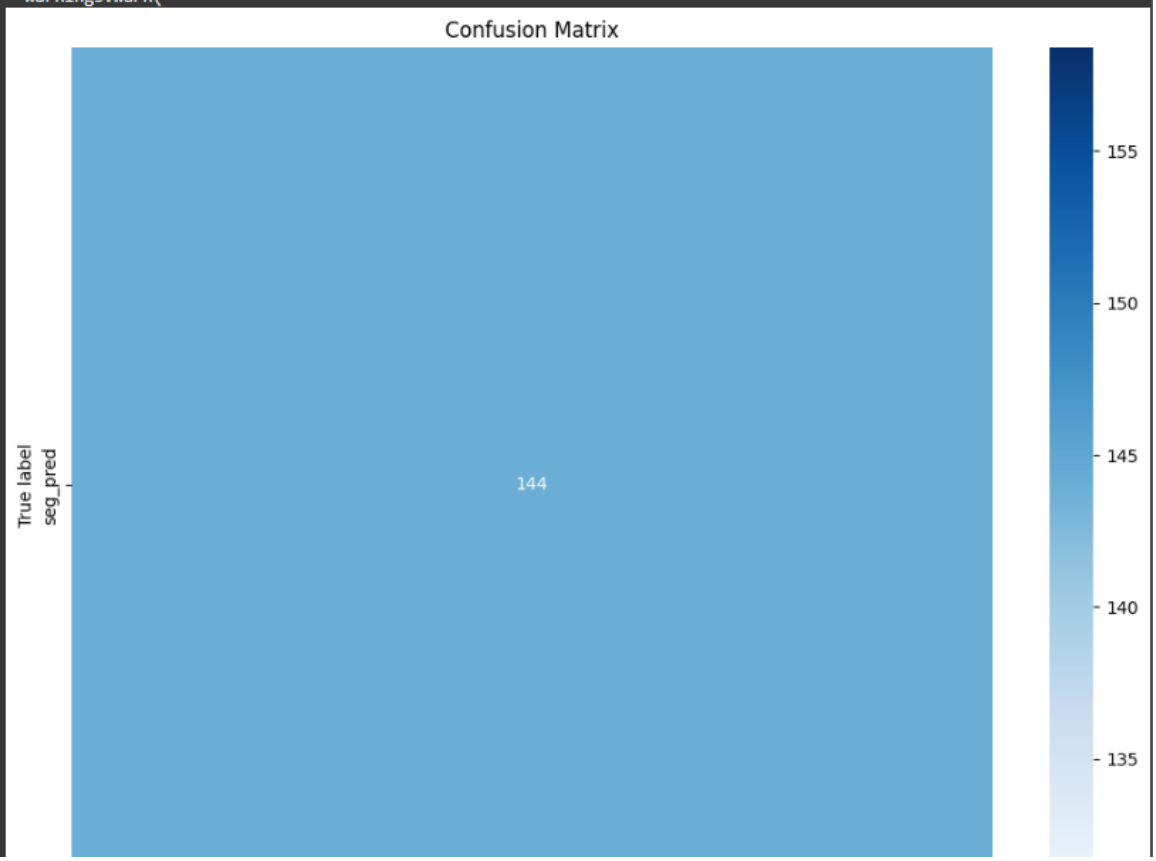
freeze early layers because they already know useful low-level features and freezing them makes fine-tuning faster and more stable on small datasets.

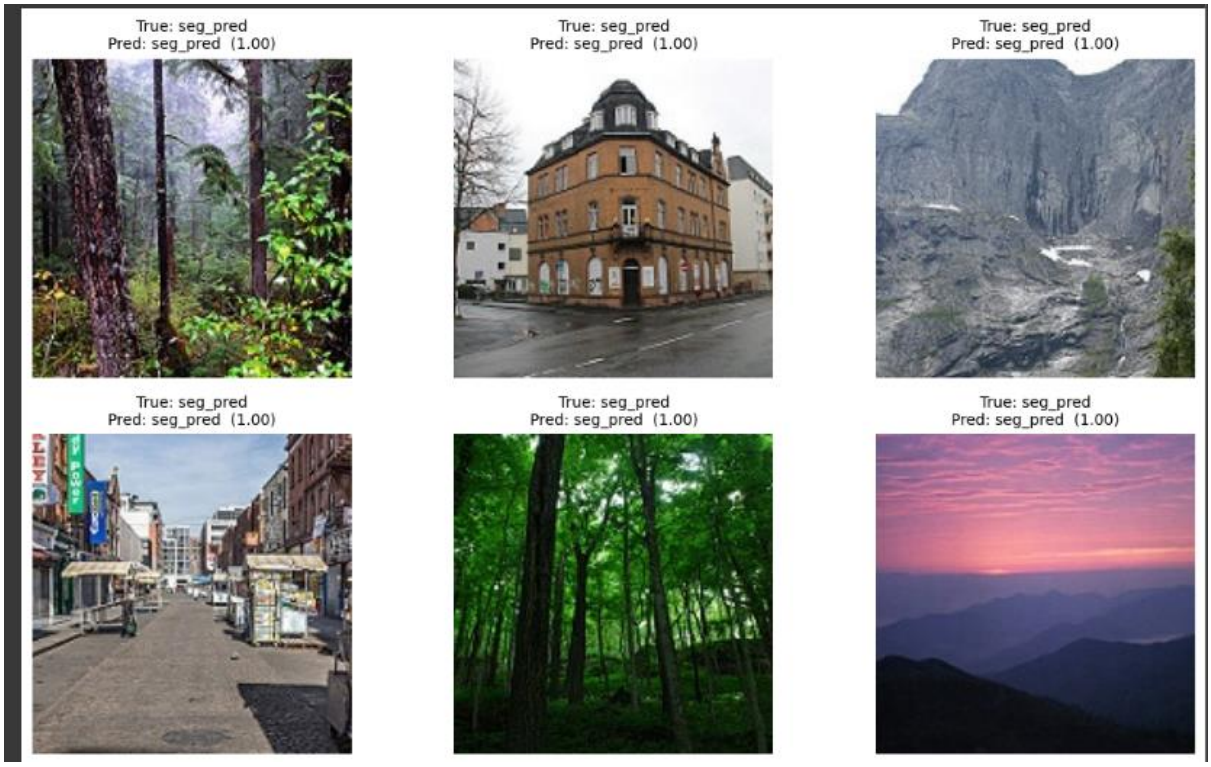
```
              precision    recall  f1-score   support

    seg_pred     1.0000    1.0000    1.0000       144

    accuracy                         1.0000       144
   macro avg     1.0000    1.0000    1.0000       144
weighted avg     1.0000    1.0000    1.0000       144

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: A single label was found
  warnings.warn(
```





Uploaded image

Prediction

This looks like: **seg_pred**

Confidence: **1.0000** (100.0%)

Top predictions

- seg_pred: 1.0000

Drag and drop files here
Limit 200MB per file • JPG, JPEG, PNG, BMP

Browse files

927.jpg 13.1KB

926.jpg 10.6KB

99.jpg 17.9KB

Showing page 1 of 2

4 file(s) uploaded — performing batch prediction...

The `use_column_width` parameter has been deprecated and will be removed in a future release. Please utilize the `use_container_width` parameter instead.

Prediction

Class: seg_pred

Confidence: 1.0000 (100.0%)

Top predictions

- seg_pred: 1.0000