```
!pip install -q tensorflow matplotlib
```

```
# Import necessary libraries
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt


# Set the base directory for the dataset
base_dir = '/content/Dataset/'  # Adjust if needed


# Set image size and batch size
img_size = 180
batch_size = 32


# Load dataset with training and validation split
train_ds = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size
)
```

```
Found 250 files belonging to 5 classes.
Using 200 files for training.
Found 250 files belonging to 5 classes.
Using 50 files for validation.
```

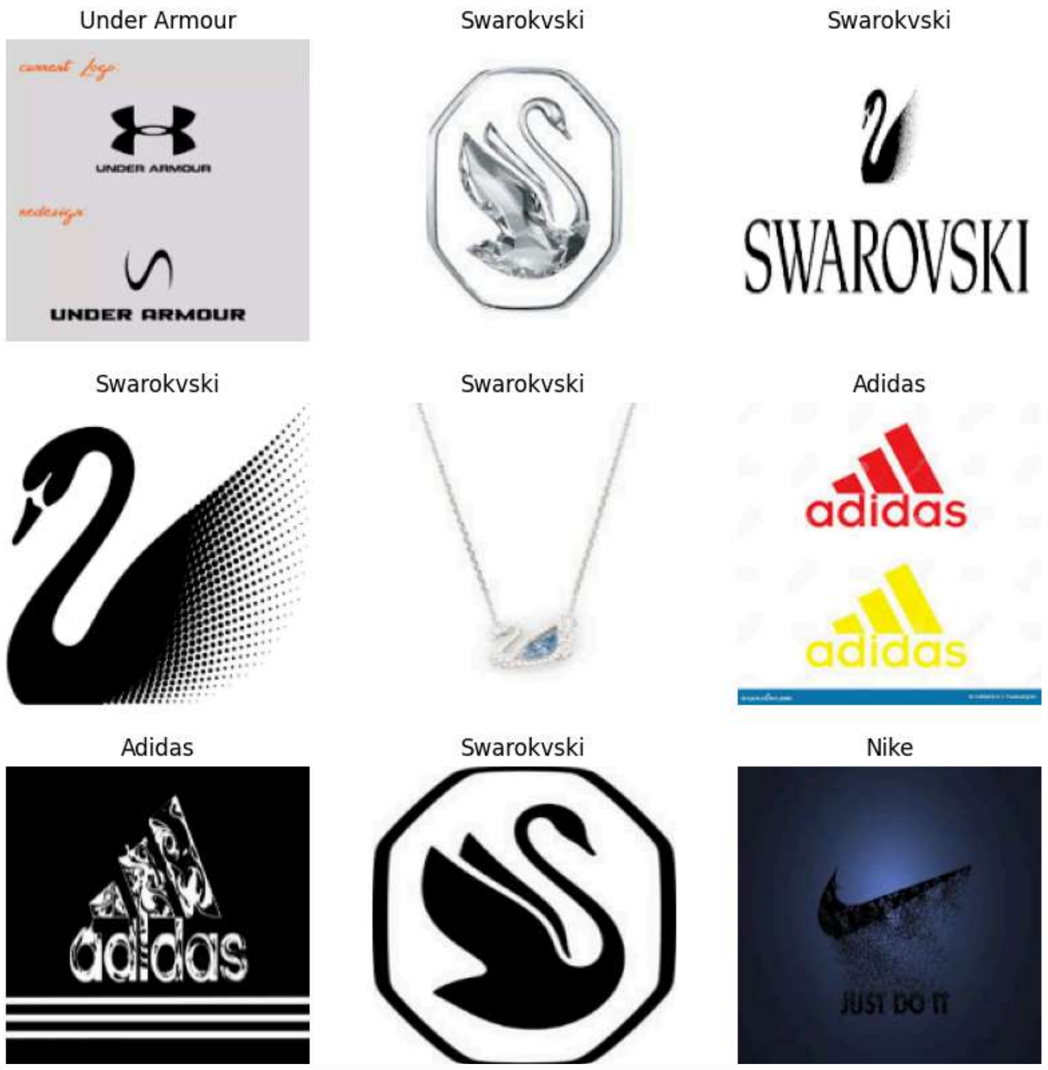
```
# Check class names to confirm they load correctly
brand_names = train_ds.class_names
print("Brand Names:", brand_names)
```

```
Brand Names: ['Adidas', 'Apple', 'Nike', 'Swarokvski', 'Under Armour']
```

```
# Data Augmentation
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
])


# Optimize dataset loading with caching and prefetching
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)


# Visualize sample images (optional)
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(brand_names[labels[i]])
        plt.axis("off")
plt.show()
```



```
# Reload the dataset (if needed)
train_ds = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size
)

# Check class names right after loading the dataset
print("Class names:", train_ds.class_names)

# Continue with the caching and prefetching
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
Found 250 files belonging to 5 classes.
Using 200 files for training.
Class names: ['Adidas', 'Apple', 'Nike', 'Swarokvski', 'Under Armour']
```

```python
from tensorflow.keras import layers, models

# Define a minimal CNN model to test if it builds correctly
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(180, 180, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(5, activation='softmax')
])

# Display the model's summary
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/preprocessing/tf_data_layer.py:19: UserWarning: Do not pass an `input_shape
  super().__init__(**kwargs)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ rescaling (Rescaling)                │ (None, 180, 180, 3)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d (Conv2D)                      │ (None, 178, 178, 32)        │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 89, 89, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 253472)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 5)                   │       1,267,365 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 1,268,261 (4.84 MB)
 Trainable params: 1,268,261 (4.84 MB)
```

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(180, 180, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(5, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Display the model's architecture
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
conv2d_1 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_2 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_3 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_4 (Conv2D)	(None, 18, 18, 128)	147,584
max_pooling2d_4 (MaxPooling2D)	(None, 9, 9, 128)	0
flatten_1 (Flatten)	(None, 10368)	0
dense_1 (Dense)	(None, 128)	1,327,232
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645

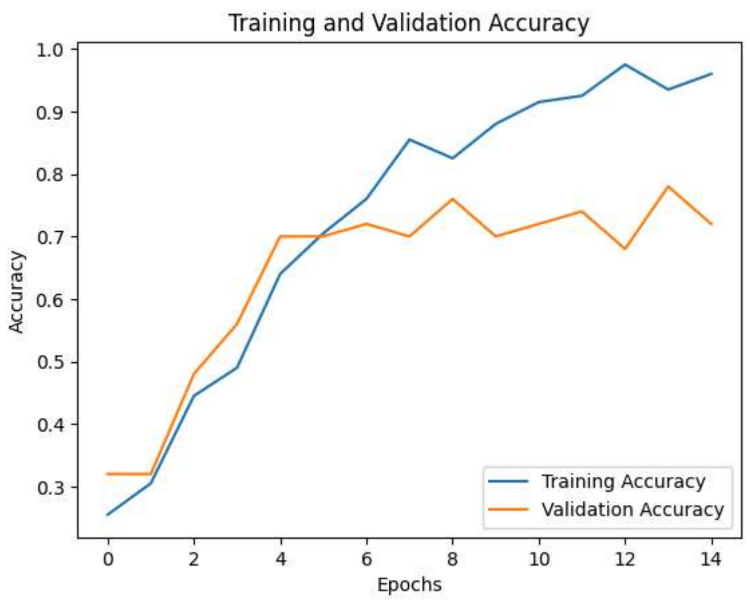
```
# Train the model
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=15  # You can adjust the number of epochs if needed
)
```

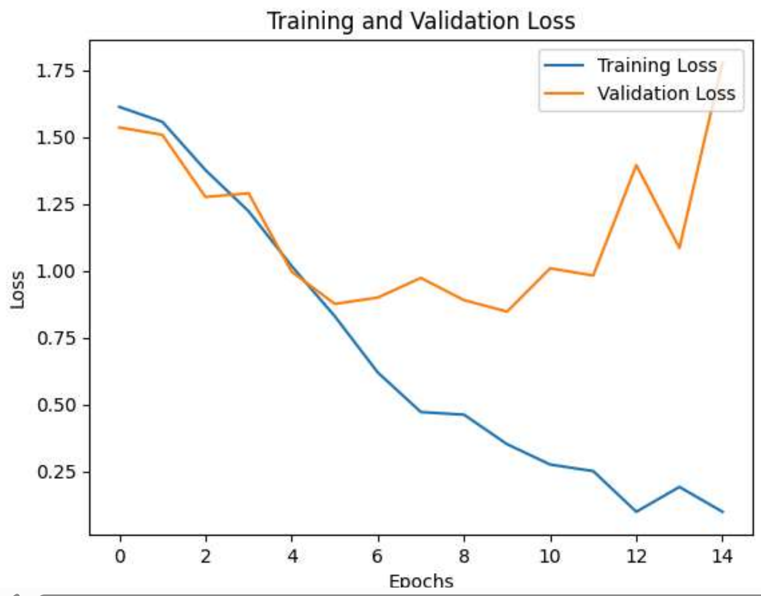
```
Epoch 1/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 17s 2s/step - accuracy: 0.2473 - loss: 1.6210 - val_accuracy: 0.3200 - val_loss: 1.5351
Epoch 2/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3054 - loss: 1.5593 - val_accuracy: 0.3200 - val_loss: 1.5076
Epoch 3/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.4276 - loss: 1.4272 - val_accuracy: 0.4800 - val_loss: 1.2752
Epoch 4/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4863 - loss: 1.2169 - val_accuracy: 0.5600 - val_loss: 1.2890
Epoch 5/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.6600 - loss: 1.0322 - val_accuracy: 0.7000 - val_loss: 0.9957
Epoch 6/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step - accuracy: 0.6880 - loss: 0.8898 - val_accuracy: 0.7000 - val_loss: 0.8754
Epoch 7/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.7718 - loss: 0.6478 - val_accuracy: 0.7200 - val_loss: 0.8991
Epoch 8/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.8458 - loss: 0.5133 - val_accuracy: 0.7000 - val_loss: 0.9726
Epoch 9/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.8314 - loss: 0.4351 - val_accuracy: 0.7600 - val_loss: 0.8894
Epoch 10/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 14s 2s/step - accuracy: 0.8840 - loss: 0.3858 - val_accuracy: 0.7000 - val_loss: 0.8467
Epoch 11/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 14s 2s/step - accuracy: 0.9239 - loss: 0.2495 - val_accuracy: 0.7200 - val_loss: 1.0084
Epoch 12/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 14s 2s/step - accuracy: 0.9312 - loss: 0.2531 - val_accuracy: 0.7400 - val_loss: 0.9816
Epoch 13/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.9526 - loss: 0.1443 - val_accuracy: 0.6800 - val_loss: 1.3941
Epoch 14/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9010 - loss: 0.2992 - val_accuracy: 0.7800 - val_loss: 1.0843
Epoch 15/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 16s 2s/step - accuracy: 0.9524 - loss: 0.1005 - val_accuracy: 0.7200 - val_loss: 1.7781
```

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()

# Plot training & validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





```
val_loss, val_accuracy = model.evaluate(val_ds)
print(f"Validation Loss: {val_loss}")
print(f"Validation Accuracy: {val_accuracy}")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 671ms/step - accuracy: 0.7196 - loss: 1.9221
Validation Loss: 1.7781223058700562
Validation Accuracy: 0.7200000286102295
```