```python
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB3
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
import os
from google.colab import drive
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Mount Google Drive
drive.mount('/content/drive')

# Set Paths and Constants
folder_path = '/content/drive/MyDrive/climate'
IMAGE_SIZE = 256
BATCH_SIZE = 8
EPOCHS = 10

# Load Dataset
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    folder_path,
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

# Split into training and validation sets
val_size = int(0.2 * len(dataset))
val_ds = dataset.take(val_size)
```

```
train_ds = dataset.skip(val_size)

# Prefetch for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)

# Build the Model with EfficientNetB3
base_model = EfficientNetB3(weights='imagenet', include_top=False, input_shape=(IMAGE_SIZE,
IMAGE_SIZE, 3))
base_model.trainable = False  # Freeze base model layers

# Create Model
inputs = Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
x = base_model(inputs, training=False)
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
outputs = Dense(4, activation='softmax')(x)  # 4 classes: cloudy, rain, shine, sunrise

model = Model(inputs, outputs)

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

# Model Summary
model.summary()
```

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

# Get true labels and predicted labels
y_true = np.concatenate([y for x, y in val_ds], axis=0)
y_pred_probs = model.predict(val_ds)
y_pred = np.argmax(y_pred_probs, axis=1)

# Classification report
print(classification_report(y_true, y_pred, target_names=dataset.class_names))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=dataset.class_names,
            yticklabels=dataset.class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Plot Accuracy and Loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Accuracy')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss')
plt.legend()

plt.show()

# Predict and visualize images
  class_names = dataset.class_names
    for images, labels in val_ds.take(1):
     predictions = model.predict(images)
      predicted_classes = np.argmax(predictions, axis
        plt.figure(figsize=(15, 10))
        for i in range(images.shape[0]):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        true_label = class_names[labels[i]]
        pred_label = class_names[predicted_classes[i]]
        color = "green" if true_label == pred_label else "red"
        plt.title(f"True: {true_label}\nPred: {pred_label}", color=color)
        plt.axis("off")
       plt.show()
```