```
import pandas as pd
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Parameters
IMAGE_SIZE = (128, 128)
BATCH_SIZE = 32
EPOCHS = 30

# Load CSVs
train_csv = pd.read_csv("Training_set.csv")
test_csv = pd.read_csv("Testing_set.csv")

train_csv.head()
```

	filename	label
0	Image_1.jpg	SOUTHERN DOGFACE
1	Image_2.jpg	ADONIS
2	Image_3.jpg	BROWN SIPROETA
3	Image_4.jpg	MONARCH
4	Image_5.jpg	GREEN CELLED CATTLEHEART

```
test_csv.head()
```

	filename
0	Image_1.jpg
1	Image_2.jpg
2	Image_3.jpg
3	Image_4.jpg
4	Image_5.jpg

```
# Add image paths
train_csv['filepath'] = 'train/' + train_csv['filename']
test_csv['filepath'] = 'test/' + test_csv['filename']

# Encode labels for training set only
label_encoder = LabelEncoder()
train_csv['label_enc'] =
label_encoder.fit_transform(train_csv['label'])

# Load and preprocess training images
def load_images(filepaths, labels):
    images = []
```

```
    for fp in filepaths:
        img = load_img(fp, target_size=IMAGE_SIZE)
        img = img_to_array(img) / 255.0  # normalize
        images.append(img)
    return np.array(images), tf.keras.utils.to_categorical(labels)

# Load test images (no labels)
def load_images_test(filepaths):
    images = []
    for fp in filepaths:
        img = load_img(fp, target_size=IMAGE_SIZE)
        img = img_to_array(img) / 255.0
        images.append(img)
    return np.array(images)

# Load train data
X_train, y_train = load_images(train_csv['filepath'],
train_csv['label_enc'])

# Load test data (optional usage — predictions only)
X_test = load_images_test(test_csv['filepath'])

# Train-validation split
X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train, y_train, test_size=0.3, random_state=42
)

# Build CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
input_shape=(*IMAGE_SIZE, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(label_encoder.classes_),
activation='softmax')
])

c:\Users\sajee\anaconda3\Lib\site-packages\keras\src\layers\
convolutional\base_conv.py:107: UserWarning: Do not pass an
```

```
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ conv2d (Conv2D)                      │ (None, 126, 126, 32)        │
896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization                  │ (None, 126, 126, 32)        │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d (MaxPooling2D)         │ (None, 63, 63, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_1 (Conv2D)                    │ (None, 61, 61, 64)          │
18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization_1                │ (None, 61, 61, 64)          │
256 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 30, 30, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_2 (Conv2D)                    │ (None, 28, 28, 128)         │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
```

```
| batch_normalization_2            | (None, 28, 28, 128)    |
512 |
| (BatchNormalization)             |                        |
|
├──────────────────────────────────┼────────────────────────┼──────────
──────┤
| max_pooling2d_2 (MaxPooling2D)   | (None, 14, 14, 128)    |
0 |
├──────────────────────────────────┼────────────────────────┼──────────
──────┤
| flatten (Flatten)                | (None, 25088)          |
0 |
├──────────────────────────────────┼────────────────────────┼──────────
──────┤
| dense (Dense)                    | (None, 256)            |
6,422,784 |
├──────────────────────────────────┼────────────────────────┼──────────
──────┤
| dropout (Dropout)                | (None, 256)            |
0 |
├──────────────────────────────────┼────────────────────────┼──────────
──────┤
| dense_1 (Dense)                  | (None, 75)             |
19,275 |
└──────────────────────────────────┴────────────────────────┴──────────
──────┘

 Total params: 6,536,203 (24.93 MB)

 Trainable params: 6,535,755 (24.93 MB)

 Non-trainable params: 448 (1.75 KB)
```

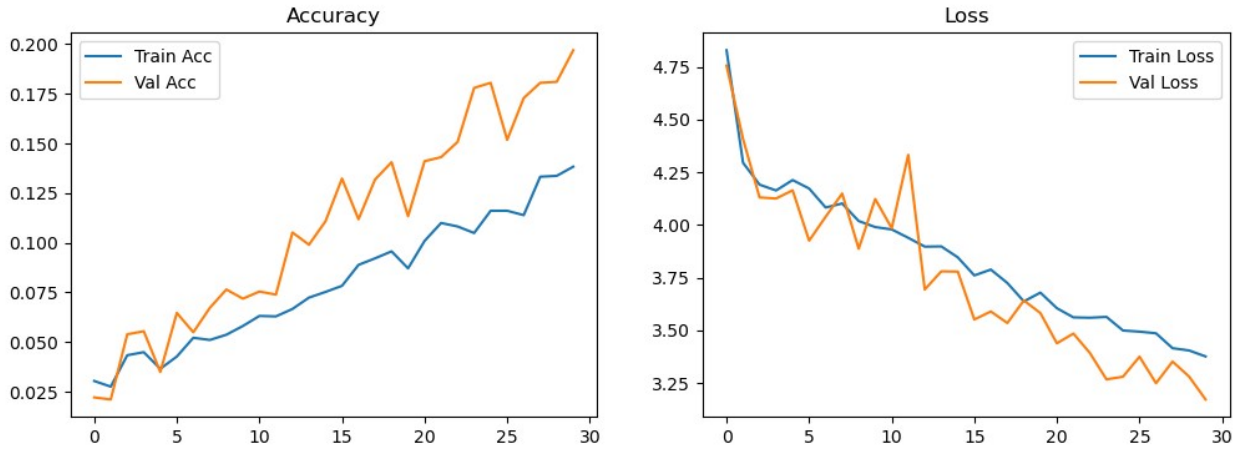
```
# Train model
history = model.fit(
    X_train_final, y_train_final,
    validation_data=(X_val, y_val),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE
)

Epoch 1/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 58s 370ms/step - accuracy: 0.0284 - loss:
6.0858 - val_accuracy: 0.0221 - val_loss: 4.7553
Epoch 2/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 52s 364ms/step - accuracy: 0.0295 - loss:
4.2965 - val_accuracy: 0.0210 - val_loss: 4.4069
Epoch 3/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 52s 364ms/step - accuracy: 0.0369 - loss:
4.2269 - val_accuracy: 0.0538 - val_loss: 4.1302
```

```
Epoch 4/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 52s 364ms/step - accuracy: 0.0423 - loss:
4.1732 - val_accuracy: 0.0554 - val_loss: 4.1250
Epoch 5/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 69s 481ms/step - accuracy: 0.0439 - loss:
4.1995 - val_accuracy: 0.0349 - val_loss: 4.1640
Epoch 6/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 68s 472ms/step - accuracy: 0.0377 - loss:
4.1907 - val_accuracy: 0.0646 - val_loss: 3.9253
Epoch 7/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 66s 459ms/step - accuracy: 0.0505 - loss:
4.0975 - val_accuracy: 0.0549 - val_loss: 4.0386
Epoch 8/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 59s 411ms/step - accuracy: 0.0448 - loss:
4.1493 - val_accuracy: 0.0672 - val_loss: 4.1489
Epoch 9/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 52s 365ms/step - accuracy: 0.0558 - loss:
4.0457 - val_accuracy: 0.0764 - val_loss: 3.8866
Epoch 10/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 53s 370ms/step - accuracy: 0.0575 - loss:
3.9886 - val_accuracy: 0.0718 - val_loss: 4.1225
Epoch 11/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 53s 369ms/step - accuracy: 0.0681 - loss:
3.9717 - val_accuracy: 0.0754 - val_loss: 3.9841
Epoch 12/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 60s 416ms/step - accuracy: 0.0588 - loss:
3.9459 - val_accuracy: 0.0738 - val_loss: 4.3321
Epoch 13/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 58s 409ms/step - accuracy: 0.0766 - loss:
3.8656 - val_accuracy: 0.1051 - val_loss: 3.6934
Epoch 14/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 60s 420ms/step - accuracy: 0.0692 - loss:
3.9028 - val_accuracy: 0.0990 - val_loss: 3.7794
Epoch 15/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 61s 425ms/step - accuracy: 0.0836 - loss:
3.8351 - val_accuracy: 0.1108 - val_loss: 3.7783
Epoch 16/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 65s 454ms/step - accuracy: 0.0788 - loss:
3.7734 - val_accuracy: 0.1323 - val_loss: 3.5511
Epoch 17/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 66s 460ms/step - accuracy: 0.0890 - loss:
3.7990 - val_accuracy: 0.1118 - val_loss: 3.5893
Epoch 18/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 66s 461ms/step - accuracy: 0.0936 - loss:
3.7366 - val_accuracy: 0.1318 - val_loss: 3.5345
Epoch 19/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 67s 470ms/step - accuracy: 0.0975 - loss:
3.6278 - val_accuracy: 0.1405 - val_loss: 3.6415
Epoch 20/30
```

```
143/143 ━━━━━━━━━━━━━━━━━━━━ 68s 476ms/step - accuracy: 0.1026 - loss:
3.6568 - val_accuracy: 0.1133 - val_loss: 3.5817
Epoch 21/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 69s 481ms/step - accuracy: 0.1077 - loss:
3.6025 - val_accuracy: 0.1410 - val_loss: 3.4383
Epoch 22/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 70s 488ms/step - accuracy: 0.1092 - loss:
3.5683 - val_accuracy: 0.1431 - val_loss: 3.4845
Epoch 23/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 69s 483ms/step - accuracy: 0.1104 - loss:
3.5478 - val_accuracy: 0.1508 - val_loss: 3.3919
Epoch 24/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 67s 469ms/step - accuracy: 0.1060 - loss:
3.5517 - val_accuracy: 0.1779 - val_loss: 3.2672
Epoch 25/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 71s 497ms/step - accuracy: 0.1184 - loss:
3.4943 - val_accuracy: 0.1805 - val_loss: 3.2798
Epoch 26/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 67s 470ms/step - accuracy: 0.1204 - loss:
3.4632 - val_accuracy: 0.1518 - val_loss: 3.3749
Epoch 27/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 74s 516ms/step - accuracy: 0.1076 - loss:
3.4994 - val_accuracy: 0.1728 - val_loss: 3.2490
Epoch 28/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 76s 471ms/step - accuracy: 0.1361 - loss:
3.3874 - val_accuracy: 0.1805 - val_loss: 3.3517
Epoch 29/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 68s 477ms/step - accuracy: 0.1305 - loss:
3.4014 - val_accuracy: 0.1810 - val_loss: 3.2807
Epoch 30/30
143/143 ━━━━━━━━━━━━━━━━━━━━ 67s 472ms/step - accuracy: 0.1285 - loss:
3.4119 - val_accuracy: 0.1969 - val_loss: 3.1719
```

```python
# Plot accuracy and loss
def plot_history(hist):
    plt.figure(figsize=(12, 4))

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

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

```
plot_history(history)
```




```
# Evaluate on validation set
val_preds = model.predict(X_val)
y_val_pred = np.argmax(val_preds, axis=1)
y_val_true = np.argmax(y_val, axis=1)
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step
```

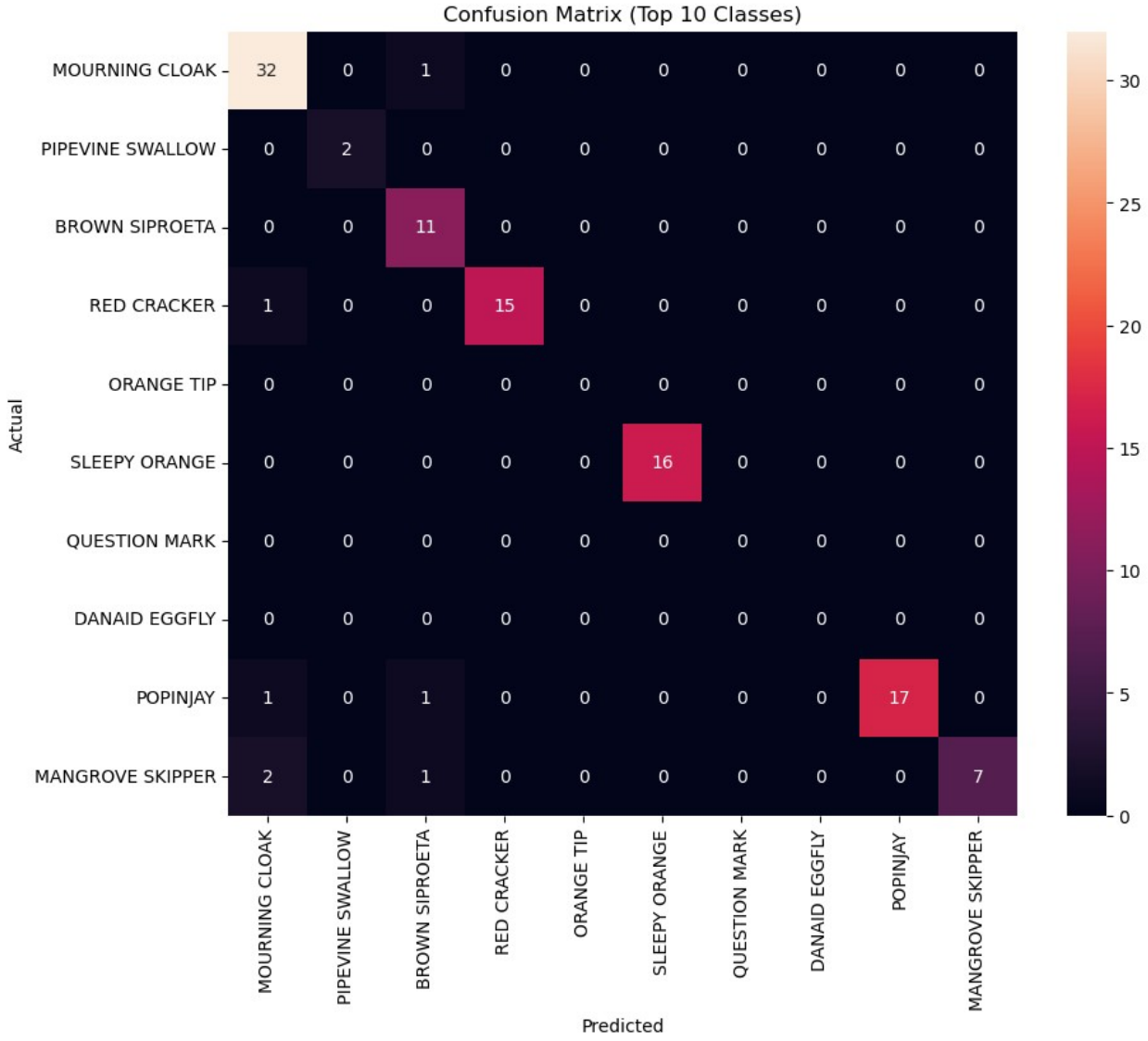
```
print("\nClassification Report (Validation Set):")
print(classification_report(y_val_true, y_val_pred,
target_names=label_encoder.classes_))
```

```
Classification Report (Validation Set):
                           precision    recall  f1-score   support

                   ADONIS       0.62      0.78      0.69        23
AFRICAN GIANT SWALLOWTAIL       0.00      0.00      0.00        24
           AMERICAN SNOOT       0.00      0.00      0.00        19
                   AN 88        0.00      0.00      0.00        20
                  APPOLLO       0.30      0.61      0.40        28
                   ATALA        0.16      0.52      0.24        27
 BANDED ORANGE HELICONIAN       0.04      1.00      0.08        30
           BANDED PEACOCK       0.33      0.32      0.33        22
            BECKERS WHITE       0.00      0.00      0.00        29
         BLACK HAIRSTREAK       0.00      0.00      0.00        30
              BLUE MORPHO       0.38      0.24      0.29        21
        BLUE SPOTTED CROW       0.33      0.12      0.17        26
           BROWN SIPROETA       0.28      0.29      0.29        38
            CABBAGE WHITE       0.20      0.03      0.06        30
          CAIRNS BIRDWING       0.33      0.04      0.08        23
       CHECQUERED SKIPPER       0.00      0.00      0.00        31
                  CHESTNUT      0.00      0.00      0.00        27
                 CLEOPATRA      0.23      0.61      0.34        28
```

CLODIUS PARNASSIAN	0.11	0.39	0.17	23
CLOUDED SULPHUR	0.00	0.00	0.00	27
COMMON BANDED AWL	0.06	0.03	0.04	30
COMMON WOOD-NYMPH	0.07	0.55	0.13	20
COPPER TAIL	0.16	0.16	0.16	25
CRECENT	0.16	0.27	0.20	26
CRIMSON PATCH	0.00	0.00	0.00	16
DANAID EGGFLY	0.00	0.00	0.00	31
EASTERN COMA	0.00	0.00	0.00	30
EASTERN DAPPLE WHITE	0.17	0.63	0.27	27
EASTERN PINE ELFIN	0.00	0.00	0.00	29
ELBOWED PIERROT	0.18	0.09	0.12	22
GOLD BANDED	1.00	0.17	0.29	24
GREAT EGGFLY	0.00	0.00	0.00	22
GREAT JAY	0.13	0.07	0.09	30
GREEN CELLED CATTLEHEART	0.00	0.00	0.00	29
GREY HAIRSTREAK	0.33	0.14	0.20	29
INDRA SWALLOW	0.00	0.00	0.00	25
IPHICLUS SISTER	0.00	0.00	0.00	30
JULIA	0.60	0.12	0.19	26
LARGE MARBLE	0.00	0.00	0.00	21
MALACHITE	0.00	0.00	0.00	18
MANGROVE SKIPPER	0.50	0.23	0.31	31
MESTRA	0.16	0.14	0.15	22
METALMARK	0.00	0.00	0.00	24
MILBERTS TORTOISESHELL	0.11	0.04	0.06	27
MONARCH	0.00	0.00	0.00	31
MOURNING CLOAK	0.74	0.71	0.73	45
ORANGE OAKLEAF	0.33	0.25	0.29	28
ORANGE TIP	0.00	0.00	0.00	36
ORCHARD SWALLOW	0.00	0.00	0.00	27
PAINTED LADY	0.00	0.00	0.00	16
PAPER KITE	0.45	0.91	0.61	22
PEACOCK	0.00	0.00	0.00	22
PINE WHITE	0.00	0.00	0.00	31
PIPEVINE SWALLOW	0.50	0.05	0.10	38
POPINJAY	0.94	0.55	0.69	31
PURPLE HAIRSTREAK	0.00	0.00	0.00	28
PURPLISH COPPER	0.07	0.16	0.09	25
QUESTION MARK	0.00	0.00	0.00	32
RED ADMIRAL	0.38	0.40	0.39	20
RED CRACKER	1.00	0.42	0.59	36
RED POSTMAN	0.05	0.03	0.04	29
RED SPOTTED PURPLE	0.47	0.38	0.42	24
SCARCE SWALLOW	0.62	0.74	0.68	27
SILVER SPOT SKIPPER	0.00	0.00	0.00	22
SLEEPY ORANGE	0.44	0.47	0.46	34
SOOTYWING	0.35	0.29	0.31	28
SOUTHERN DOGFACE	0.00	0.00	0.00	22

```
          STRAITED QUEEN       0.00      0.00      0.00        24
        TROPICAL LEAFWING      0.00      0.00      0.00        18
       TWO BARRED FLASHER      0.41      0.50      0.45        14
                   ULYSES      0.75      0.69      0.72        26
                  VICEROY      0.00      0.00      0.00        19
               WOOD SATYR      0.00      0.00      0.00        17
      YELLOW SWALLOW TAIL      1.00      0.05      0.10        20
          ZEBRA LONG WING      0.00      0.00      0.00        18

                 accuracy                          0.20      1950
                macro avg      0.21      0.19      0.16      1950
             weighted avg      0.22      0.20      0.17      1950


c:\Users\sajee\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1509: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\sajee\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1509: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\sajee\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1509: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

# Confusion matrix (top 10 frequent classes in val set)
unique, counts = np.unique(y_val_true, return_counts=True)
top10_indices = unique[np.argsort(-counts)][:10]
top10_labels = label_encoder.classes_[top10_indices]

mask = np.isin(y_val_true, top10_indices)
filtered_true = y_val_true[mask]
filtered_pred = y_val_pred[mask]

cm = confusion_matrix(filtered_true, filtered_pred,
labels=top10_indices)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=top10_labels,
yticklabels=top10_labels)
plt.title("Confusion Matrix (Top 10 Classes)")
plt.xlabel("Predicted")
```

```
plt.ylabel("Actual")
plt.show()
```




```
# Predict on test set (optional)
test_preds = model.predict(X_test)
test_class_indices = np.argmax(test_preds, axis=1)
test_labels = label_encoder.inverse_transform(test_class_indices)

88/88 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step

# Save test predictions to CSV
test_csv['predicted_label'] = test_labels
test_csv.to_csv("Test_Predictions.csv", index=False)
print("\nTest predictions saved to Test_Predictions.csv")
```

```
Test predictions saved to Test_Predictions.csv
```