*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

os.path.join(dirname, filename)

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

# butterfly image classification



# Import Python Libraries

In [2]:

import pandas as pd

import numpy as np

import os

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

2025-06-19 05:46:35.191037: E external/local\_xla/xla/stream\_executor/cuda/cuda\_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

E0000 00:00:1750311995.409009 19 cuda\_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered

E0000 00:00:1750311995.472913 19 cuda\_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered

**Load CSV containing train image labels**

In [3]:

*# 1. Load CSV containing train image labels*

df = pd.read\_csv('//kaggle/input/butterfly-image-classification/Training\_set.csv')

**printing csv file**

In [4]:

df

Out[4]:

|  | 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 |
| ... | ... | ... |
| 6494 | Image\_6495.jpg | MANGROVE SKIPPER |
| 6495 | Image\_6496.jpg | MOURNING CLOAK |
| 6496 | Image\_6497.jpg | APPOLLO |
| 6497 | Image\_6498.jpg | ELBOWED PIERROT |
| 6498 | Image\_6499.jpg | ATALA |

6499 rows × 2 columns

**path to image**

In [5]:

*# 2. Path to images*

train\_dir = '/kaggle/input/butterfly-image-classification/train'

test\_dir = '/kaggle/input/butterfly-image-classification/test'

linkcode

# Create ImageDataGenerator with rescaling and augmentation

**Import Python Libraries**

In [2]:

import pandas as pd

import numpy as np

import os

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

2025-06-19 05:46:35.191037: E external/local\_xla/xla/stream\_executor/cuda/cuda\_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

E0000 00:00:1750311995.409009 19 cuda\_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered

E0000 00:00:1750311995.472913 19 cuda\_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered

**Load CSV containing train image labels**

In [3]:

*# 1. Load CSV containing train image labels*

df = pd.read\_csv('//kaggle/input/butterfly-image-classification/Training\_set.csv')

**printing csv file**

In [4]:

df

Out[4]:

|  | 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 |
| ... | ... | ... |
| 6494 | Image\_6495.jpg | MANGROVE SKIPPER |
| 6495 | Image\_6496.jpg | MOURNING CLOAK |
| 6496 | Image\_6497.jpg | APPOLLO |
| 6497 | Image\_6498.jpg | ELBOWED PIERROT |
| 6498 | Image\_6499.jpg | ATALA |

6499 rows × 2 columns

**path to image**

In [5]:

*# 2. Path to images*

train\_dir = '/kaggle/input/butterfly-image-classification/train'

test\_dir = '/kaggle/input/butterfly-image-classification/test'

linkcode

**Create ImageDataGenerator with rescaling and augmentation**

**Import Python Libraries**

In [2]:

import pandas as pd

import numpy as np

import os

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

2025-06-19 05:46:35.191037: E external/local\_xla/xla/stream\_executor/cuda/cuda\_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

E0000 00:00:1750311995.409009 19 cuda\_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered

E0000 00:00:1750311995.472913 19 cuda\_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered

**Load CSV containing train image labels**

In [3]:

*# 1. Load CSV containing train image labels*

df = pd.read\_csv('//kaggle/input/butterfly-image-classification/Training\_set.csv')

**printing csv file**

In [4]:

df

Out[4]:

|  | 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 |
| ... | ... | ... |
| 6494 | Image\_6495.jpg | MANGROVE SKIPPER |
| 6495 | Image\_6496.jpg | MOURNING CLOAK |
| 6496 | Image\_6497.jpg | APPOLLO |
| 6497 | Image\_6498.jpg | ELBOWED PIERROT |
| 6498 | Image\_6499.jpg | ATALA |

6499 rows × 2 columns

**path to image**

In [5]:

*# 2. Path to images*

train\_dir = '/kaggle/input/butterfly-image-classification/train'

test\_dir = '/kaggle/input/butterfly-image-classification/test'

linkcode

# Create ImageDataGenerator with rescaling and augmentation

*# 3. Create ImageDataGenerator with rescaling and augmentation*

train\_datagen = ImageDataGenerator(

rescale=1./255,

validation\_split=0.2, *# Split training into train+validation*

rotation\_range=10,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

zoom\_range=0.1,

horizontal\_flip=True,

fill\_mode='nearest'

)

# Flow from dataframe for training

In [7]:

*# 4. Flow from dataframe for training*

train\_generator = train\_datagen.flow\_from\_dataframe(

dataframe=df,

directory=train\_dir,

x\_col='filename',

y\_col='label',

subset='training',

target\_size=(128, 128),

batch\_size=32,

class\_mode='categorical'

)

val\_generator = train\_datagen.flow\_from\_dataframe(

dataframe=df,

directory=train\_dir,

x\_col='filename',

y\_col='label',

subset='validation',

target\_size=(128, 128),

batch\_size=32,

class\_mode='categorical'

)

Found 5200 validated image filenames belonging to 75 classes.

Found 1299 validated image filenames belonging to 75 classes.

In [8]:

num\_classes = len(train\_generator.class\_indices)

# Building CNN model

In [9]:

linkcode

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(128,128,3)),

BatchNormalization(),

MaxPooling2D(2,2),

Conv2D(64, (3,3), activation='relu'),

BatchNormalization(),

MaxPooling2D(2,2),

Conv2D(128, (3,3), activation='relu'),

BatchNormalization(),

MaxPooling2D(2,2),

Flatten(),

Dense(128, activation='relu'),

Dense(512, activation='relu'),

Dense(num\_classes, activation='softmax')

])

I0000 00:00:1750312014.285460 19 gpu\_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory: -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5

I0000 00:00:1750312014.286176 19 gpu\_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory: -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5

**Model summary**

In [10]:

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, 128) │ 3,211,392 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_1 (Dense) │ (None, 512) │ 66,048 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_2 (Dense) │ (None, 75) │ 38,475 │

└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 3,410,059 (13.01 MB)

**Trainable params:** 3,409,611 (13.01 MB)

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

**compile the model**

In [11]:

*# 6. Compile the model*

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

**Train the model**

In [12]:

linkcode

*# 7. Train the model*

history = model.fit(train\_generator, validation\_data=val\_generator, epochs=30)

Epoch 1/30

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

I0000 00:00:1750312020.688705 62 service.cc:148] XLA service 0x7cef2c003bd0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:

I0000 00:00:1750312020.689809 62 service.cc:156] StreamExecutor device (0): Tesla T4, Compute Capability 7.5

I0000 00:00:1750312020.689833 62 service.cc:156] StreamExecutor device (1): Tesla T4, Compute Capability 7.5

I0000 00:00:1750312021.176593 62 cuda\_dnn.cc:529] Loaded cuDNN version 90300

**1/163** ━━━━━━━━━━━━━━━━━━━━ **23:41** 9s/step - accuracy: 0.0000e+00 - loss: 4.4757

I0000 00:00:1750312025.139125 62 device\_compiler.h:188] Compiled cluster using XLA! This line is logged at most once for the lifetime of the process.

**163/163** ━━━━━━━━━━━━━━━━━━━━ **71s** 387ms/step - accuracy: 0.0870 - loss: 4.1852 - val\_accuracy: 0.0254 - val\_loss: 7.2867

Epoch 2/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **32s** 199ms/step - accuracy: 0.2253 - loss: 3.0376 - val\_accuracy: 0.0654 - val\_loss: 5.2916

Epoch 3/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **33s** 199ms/step - accuracy: 0.3047 - loss: 2.6371 - val\_accuracy: 0.2848 - val\_loss: 2.7827

Epoch 4/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **32s** 199ms/step - accuracy: 0.3876 - loss: 2.2624 - val\_accuracy: 0.3818 - val\_loss: 2.3094

Epoch 5/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 193ms/step - accuracy: 0.4416 - loss: 1.9392 - val\_accuracy: 0.3749 - val\_loss: 2.4217

Epoch 6/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **32s** 196ms/step - accuracy: 0.4720 - loss: 1.8631 - val\_accuracy: 0.4265 - val\_loss: 2.1085

Epoch 7/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 192ms/step - accuracy: 0.5312 - loss: 1.6265 - val\_accuracy: 0.4565 - val\_loss: 2.0563

Epoch 8/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **32s** 195ms/step - accuracy: 0.5747 - loss: 1.4980 - val\_accuracy: 0.4904 - val\_loss: 1.9514

Epoch 9/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 191ms/step - accuracy: 0.6091 - loss: 1.3544 - val\_accuracy: 0.4927 - val\_loss: 1.9216

Epoch 10/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 191ms/step - accuracy: 0.6071 - loss: 1.3111 - val\_accuracy: 0.5327 - val\_loss: 1.7356

Epoch 11/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 190ms/step - accuracy: 0.6518 - loss: 1.1433 - val\_accuracy: 0.5135 - val\_loss: 1.9848

Epoch 12/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 191ms/step - accuracy: 0.6523 - loss: 1.1295 - val\_accuracy: 0.5574 - val\_loss: 1.7448

Epoch 13/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 192ms/step - accuracy: 0.6978 - loss: 0.9628 - val\_accuracy: 0.5635 - val\_loss: 1.6544

Epoch 14/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 188ms/step - accuracy: 0.7142 - loss: 0.9169 - val\_accuracy: 0.5651 - val\_loss: 1.8780

Epoch 15/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 188ms/step - accuracy: 0.7030 - loss: 0.9499 - val\_accuracy: 0.5635 - val\_loss: 1.7692

Epoch 16/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **30s** 187ms/step - accuracy: 0.7415 - loss: 0.8474 - val\_accuracy: 0.5920 - val\_loss: 1.6439

Epoch 17/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 190ms/step - accuracy: 0.7412 - loss: 0.8234 - val\_accuracy: 0.5843 - val\_loss: 1.7790

Epoch 18/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 187ms/step - accuracy: 0.7735 - loss: 0.7233 - val\_accuracy: 0.6020 - val\_loss: 1.5966

Epoch 19/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 192ms/step - accuracy: 0.7793 - loss: 0.7288 - val\_accuracy: 0.5712 - val\_loss: 1.8217

Epoch 20/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 189ms/step - accuracy: 0.7864 - loss: 0.6779 - val\_accuracy: 0.6205 - val\_loss: 1.5947

Epoch 21/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 191ms/step - accuracy: 0.7830 - loss: 0.6589 - val\_accuracy: 0.6282 - val\_loss: 1.6539

Epoch 22/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **30s** 187ms/step - accuracy: 0.8119 - loss: 0.6164 - val\_accuracy: 0.6197 - val\_loss: 1.7065

Epoch 23/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 191ms/step - accuracy: 0.8290 - loss: 0.5483 - val\_accuracy: 0.5889 - val\_loss: 1.7483

Epoch 24/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 189ms/step - accuracy: 0.8211 - loss: 0.5471 - val\_accuracy: 0.5581 - val\_loss: 2.3106

Epoch 25/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 189ms/step - accuracy: 0.8283 - loss: 0.5504 - val\_accuracy: 0.6366 - val\_loss: 1.6143

Epoch 26/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 188ms/step - accuracy: 0.8257 - loss: 0.5482 - val\_accuracy: 0.6613 - val\_loss: 1.5668

Epoch 27/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 192ms/step - accuracy: 0.8531 - loss: 0.4688 - val\_accuracy: 0.6251 - val\_loss: 1.7553

Epoch 28/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 191ms/step - accuracy: 0.8571 - loss: 0.4501 - val\_accuracy: 0.5835 - val\_loss: 2.1541

Epoch 29/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **32s** 193ms/step - accuracy: 0.8377 - loss: 0.5281 - val\_accuracy: 0.6659 - val\_loss: 1.6299

Epoch 30/30

**163/163** ━━━━━━━━━━━━━━━━━━━━ **31s** 192ms/step - accuracy: 0.8667 - loss: 0.4158 - val\_accuracy: 0.6074 - val\_loss: 2.0260

linkcode

**Evaluation model**

In [13]:

linkcode

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

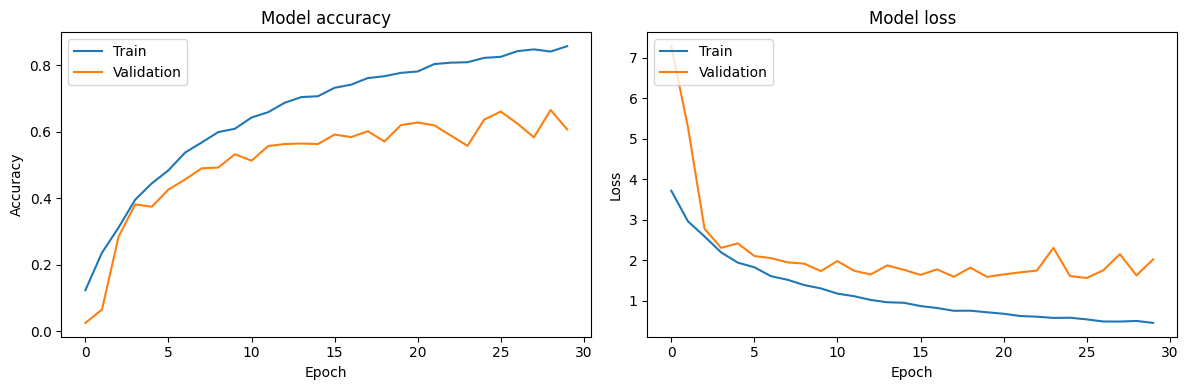
plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight\_layout()

plt.show()plt.show()



# predict images

In [14]:

linkcode

val\_images, val\_labels = next(val\_generator)

pred\_labels = model.predict(val\_images)

pred\_labels = np.argmax(pred\_labels, axis=1)

true\_labels = np.argmax(val\_labels, axis=1)

class\_indices = val\_generator.class\_indices

class\_names = {v: k for k, v **in** class\_indices.items()}

def display\_images(images, true\_labels, pred\_labels, class\_names, num\_images=9):

plt.figure(figsize=(15, 8))

for i **in** range(num\_images):

plt.subplot(3, 3, i + 1)

plt.imshow(images[i])

true\_label = class\_names[int(true\_labels[i])]

pred\_label = class\_names[int(pred\_labels[i])]

plt.title(f"True: **{**true\_label**}\n**Pred: **{**pred\_label**}**")

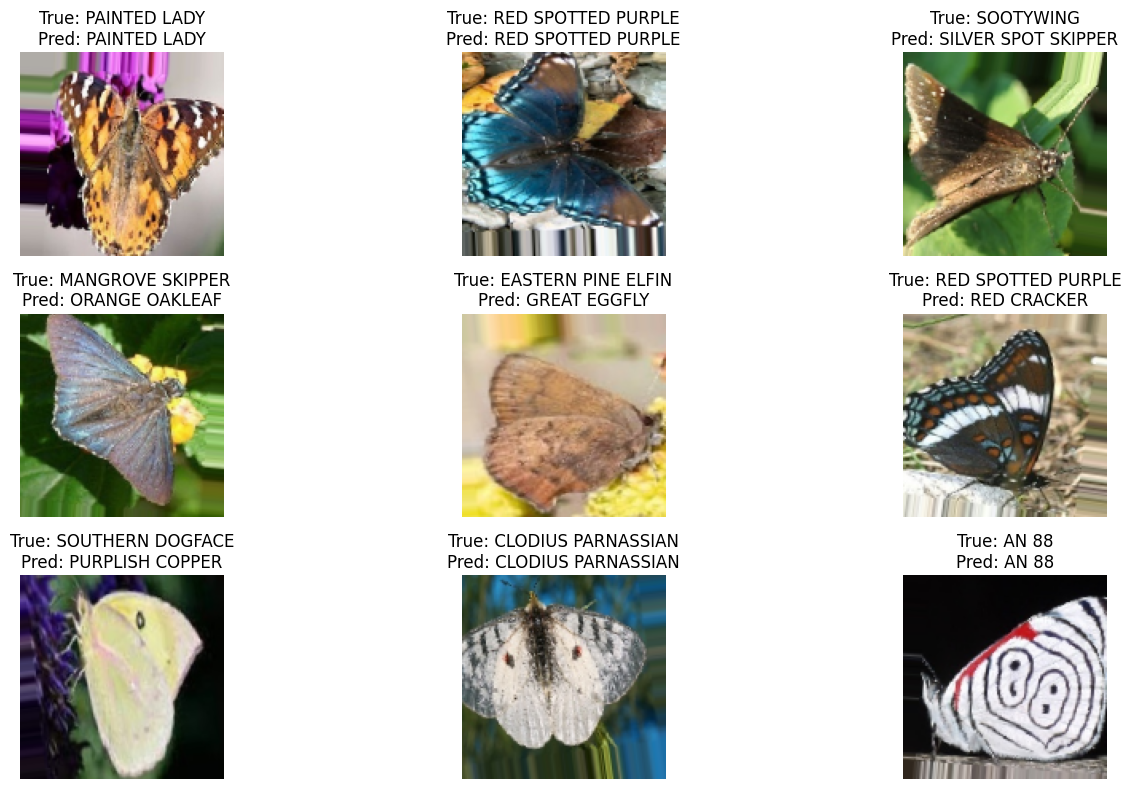
plt.axis('off')

plt.tight\_layout()

plt.show()

display\_images(val\_images, true\_labels, pred\_labels, class\_names, num\_images=9)

**1/1** ━━━━━━━━━━━━━━━━━━━━ **1s** 507ms/step



# Thank you for visiting my code

# if you find any knowledgeable please upvote