Build a CNN model for Bird species

Bird species classification is the process of using machine learning and computer vision techniques to identify and categorize different species of birds basedon their visual characteristics. By analyzing images of birds, models can extract features and patterns to accurately classify bird species. This classification isvital for ecological research, wildlife monitoring, and conservation efforts. Advancements in deep learning and the availability of large annotated datasets haveimproved the accuracy of bird species classification models. Challenges include variations in lighting, pose, and background clutter. Ongoing researchfocuses on methods like transfer learning and data augmentation to enhance classification performance and contribute to avian biodiversity understandingand conservation.

In [ ]:

```
# Import Data Science Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Tensorflow Libraries
from tensorflow import keras
from tensorflow.keras import layers,models
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import Callback, EarlyStopping,ModelCheckpoint, ReduceLR
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import Model
from tensorflow.keras.layers.experimental import preprocessing

# System libraries
from pathlib import Path
import os.path
import random

# Visualization Libraries
import matplotlib.cm as cm
import cv2
import seaborn as sns

sns.set_style('darkgrid')

# Metrics
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

Create helper functions

In [2]:

```
# Import series of helper functions for our notebook
from helper_functions import create_tensorboard_callback, plot_loss_curves, unzip_data,
```

```
--2023-04-26 05:42:13--  https://raw.githubusercontent.com/mrdbourke/tenso
rflow-deep-learning/main/extras/helper_functions.py (https://raw.githubuse
rcontent.com/mrdbourke/tensorflow-deep-learning/main/extras/helper_functio
ns.py)
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.19
9.111.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10246 (10K) [text/plain]
Saving to: ‘helper_functions.py’

helper_functions.py 100%[===================>]  10.01K  --.-KB/s    in 0s

2023-04-26 05:42:13 (48.6 MB/s) - ‘helper_functions.py’ saved [10246/1024
6]
```

Load and Transform Data

In [3]:

```
BATCH_SIZE = 32
TARGET_SIZE = (224, 224)
```

In [4]:

```
# Walk through each directory
dataset = "../input/100-bird-species/train"
walk_through_dir(dataset);
```

```
There are 525 directories and 0 images in '../input/100-bird-species/tr
ain'.
There are 0 directories and 159 images in '../input/100-bird-species/tr
ain/DALMATIAN PELICAN'.
There are 0 directories and 172 images in '../input/100-bird-species/tr
ain/BLACK BREASTED PUFFBIRD'.
There are 0 directories and 138 images in '../input/100-bird-species/tr
ain/WATTLED CURASSOW'.
There are 0 directories and 187 images in '../input/100-bird-species/tr
ain/AMERICAN WIGEON'.
There are 0 directories and 162 images in '../input/100-bird-species/tr
ain/CARMINE BEE-EATER'.
There are 0 directories and 153 images in '../input/100-bird-species/tr
ain/GAMBELS QUAIL'.
There are 0 directories and 155 images in '../input/100-bird-species/tr
ain/UMBRELLA BIRD'.
There are 0 directories and 155 images in '../input/100-bird-species/tr
ain/AMERICAN KESTREL'.
There are 0 directories and 133 images in '../input/100-bird-species/tr
```

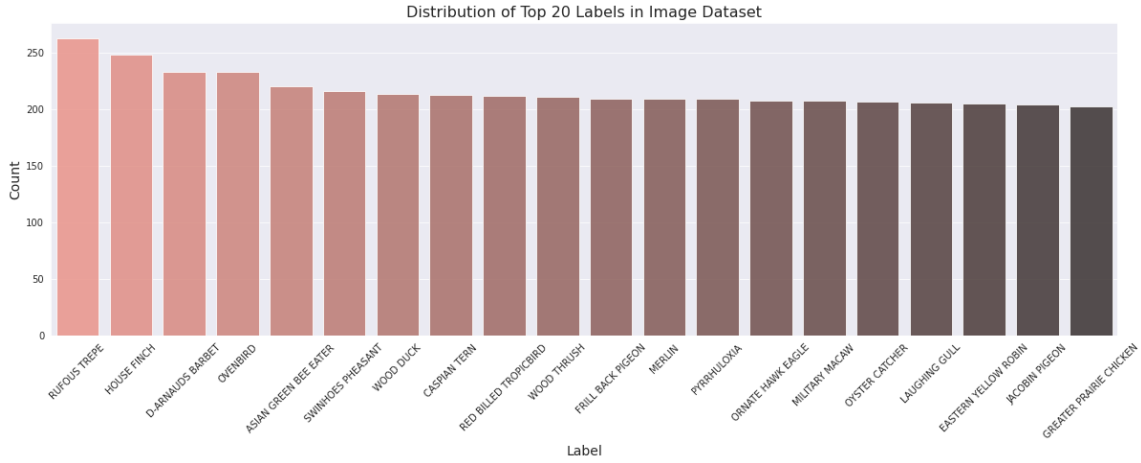
Placing Data into a Dataframe

In [5]:

```
image_dir = Path(dataset)

# Get filepaths and labels
filepaths = list(image_dir.glob(r'**/*.JPG')) + list(image_dir.glob(r'**/*.jpg')) + list

labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))

filepaths = pd.Series(filepaths, name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')

# Concatenate filepaths and labels
image_df = pd.concat([filepaths, labels], axis=1)
```

In [6]:

```
# Get the top 20 labels
label_counts = image_df['Label'].value_counts()[:20]

plt.figure(figsize=(20, 6))
sns.barplot(x=label_counts.index, y=label_counts.values, alpha=0.8, palette='dark:salmon_
plt.title('Distribution of Top 20 Labels in Image Dataset', fontsize=16)
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45)
plt.show()
```



Visualizing images from the dataset

In [7]:

```
# Display 16 picture of the dataset with their labels
random_index = np.random.randint(0, len(image_df), 16)
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(image_df.Filepath[random_index[i]]))
    ax.set_title(image_df.Label[random_index[i]])
plt.tight_layout()
plt.show()
```

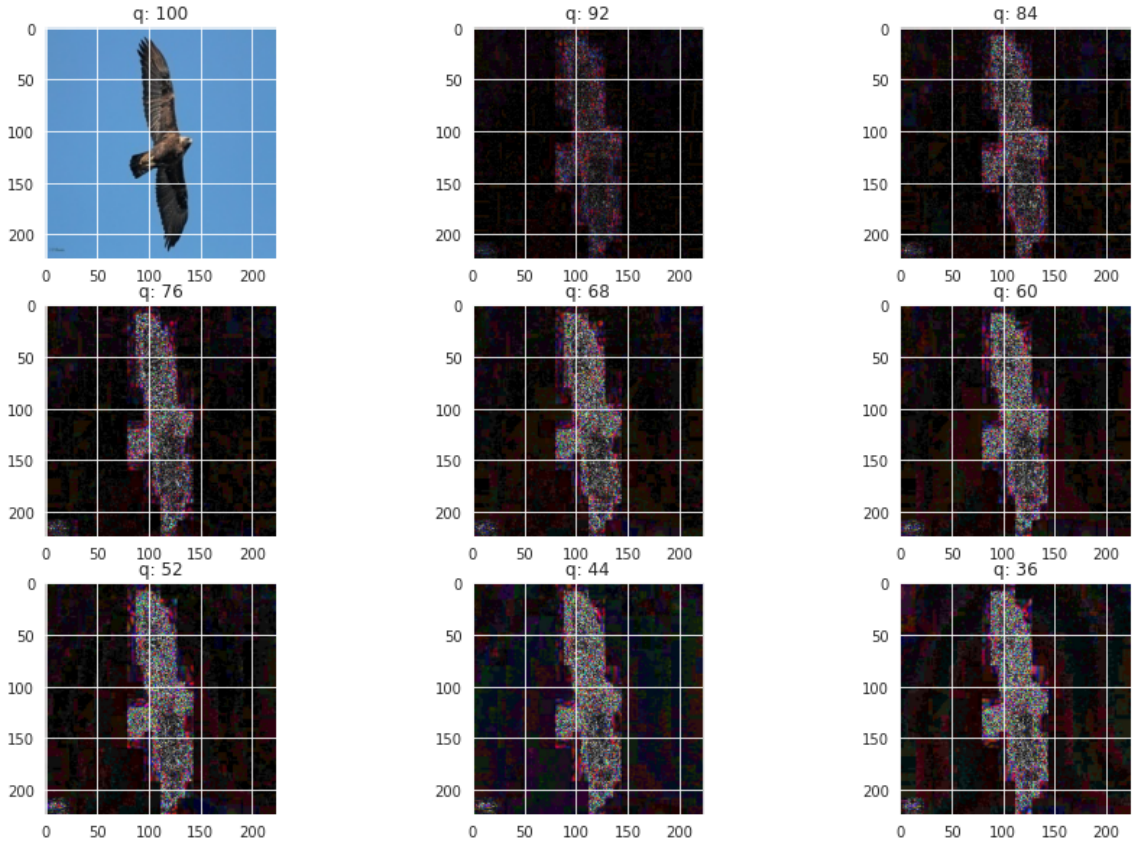
Computing Error Rate Analysis

In [8]:

```
def compute_ela_cv(path, quality):
    temp_filename = 'temp_file_name.jpeg'
    SCALE = 15
    orig_img = cv2.imread(path)
    orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

    cv2.imwrite(temp_filename, orig_img, [cv2.IMWRITE_JPEG_QUALITY, quality])

    # read compressed image
    compressed_img = cv2.imread(temp_filename)

    # get absolute difference between img1 and img2 and multiply by scale
    diff = SCALE * cv2.absdiff(orig_img, compressed_img)
    return diff


def convert_to_ela_image(path, quality):
    temp_filename = 'temp_file_name.jpeg'
    ela_filename = 'temp_ela.png'
    image = Image.open(path).convert('RGB')
    image.save(temp_filename, 'JPEG', quality = quality)
    temp_image = Image.open(temp_filename)

    ela_image = ImageChops.difference(image, temp_image)

    extrema = ela_image.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
        max_diff = 1

    scale = 255.0 / max_diff
    ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)

    return ela_image


def random_sample(path, extension=None):
    if extension:
        items = Path(path).glob(f'*.{extension}')
    else:
        items = Path(path).glob(f'*')

    items = list(items)

    p = random.choice(items)
    return p.as_posix()
```

In [9]:

```
# View random sample from the dataset
p = random_sample('../input/100-bird-species/train/GOLDEN EAGLE')
orig = cv2.imread(p)
orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0
init_val = 100
columns = 3
rows = 3

fig=plt.figure(figsize=(15, 10))
for i in range(1, columns*rows +1):
    quality=init_val - (i-1) * 8
    img = compute_ela_cv(path=p, quality=quality)
    if i == 1:
        img = orig.copy()
    ax = fig.add_subplot(rows, columns, i)
    ax.title.set_text(f'q: {quality}')
    plt.imshow(img)
plt.show()
```



Data Preprocessing

In [10]:

```
# Separate in train and test data
train_df, test_df = train_test_split(image_df, test_size=0.2, shuffle=True, random_state
```

In [11]:

```
train_generator = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.efficientnet.preprocess_input,
    validation_split=0.2
)

test_generator = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.efficientnet.preprocess_input,
)
```

In [12]:

```
# Split the data into three categories.
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=TARGET_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=TARGET_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',
    y_col='Label',
    target_size=TARGET_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 54167 validated image filenames belonging to 525 classes.
Found 13541 validated image filenames belonging to 525 classes.
Found 16927 validated image filenames belonging to 525 classes.
```

In [13]:

```
# Data Augmentation Step
augment = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(224,224),
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.experimental.preprocessing.RandomFlip("horizontal"),
  layers.experimental.preprocessing.RandomRotation(0.1),
  layers.experimental.preprocessing.RandomZoom(0.1),
  layers.experimental.preprocessing.RandomContrast(0.1),
])
```

Training the model

In [14]:

```
# Load the pretained model
pretrained_model = tf.keras.applications.efficientnet.EfficientNetB0(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='max'
)

pretrained_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/keras-applications/ef
ficientnetb0_notop.h5 (https://storage.googleapis.com/keras-applications/e
fficientnetb0_notop.h5)
16711680/16705208 [==============================] - 0s 0us/step
16719872/16705208 [==============================] - 0s 0us/step
```

In [15]:

```
# Create checkpoint callback
checkpoint_path = "birds_classification_model_checkpoint"
checkpoint_callback = ModelCheckpoint(checkpoint_path,
                                      save_weights_only=True,
                                      monitor="val_accuracy",
                                      save_best_only=True)

# Setup EarlyStopping callback to stop training if model's val_loss doesn't improve for
early_stopping = EarlyStopping(monitor = "val_loss", # watch the val loss metric
                               patience = 5,
                               restore_best_weights = True) # if val loss decreases for

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)
```

Training the model

In [16]:

```
inputs = pretrained_model.input
x = augment(inputs)

x = Dense(128, activation='relu')(pretrained_model.output)
x = Dropout(0.45)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.45)(x)


outputs = Dense(525, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer=Adam(0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_images,
    steps_per_epoch=len(train_images),
    validation_data=val_images,
    validation_steps=len(val_images),
    epochs=150,
    callbacks=[
        early_stopping,
        create_tensorboard_callback("training_logs",
                                    "bird_classification"),
        checkpoint_callback,
        reduce_lr
    ]
)
```

```
1693/1693 [==============================] - 188s 111ms/step - loss: 1.
9058 - accuracy: 0.5078 - val_loss: 0.8918 - val_accuracy: 0.8041
Epoch 18/150
1693/1693 [==============================] - 196s 116ms/step - loss: 1.
8593 - accuracy: 0.5185 - val_loss: 0.8566 - val_accuracy: 0.8090
Epoch 19/150
1693/1693 [==============================] - 184s 109ms/step - loss: 1.
8052 - accuracy: 0.5324 - val_loss: 0.8112 - val_accuracy: 0.8186
Epoch 20/150
1693/1693 [==============================] - 191s 113ms/step - loss: 1.
7699 - accuracy: 0.5398 - val_loss: 0.7999 - val_accuracy: 0.8197
Epoch 21/150
1693/1693 [==============================] - 186s 110ms/step - loss: 1.
7353 - accuracy: 0.5482 - val_loss: 0.7970 - val_accuracy: 0.8211
Epoch 22/150
1693/1693 [==============================] - 191s 113ms/step - loss: 1.
6955 - accuracy: 0.5564 - val_loss: 0.7770 - val_accuracy: 0.8248
Epoch 23/150
1693/1693 [==============================] - 181s 107ms/step - loss: 1.
6556 - accuracy: 0.5671 - val_loss: 0.7605 - val_accuracy: 0.8301
```

Model Evaluation

In [17]:

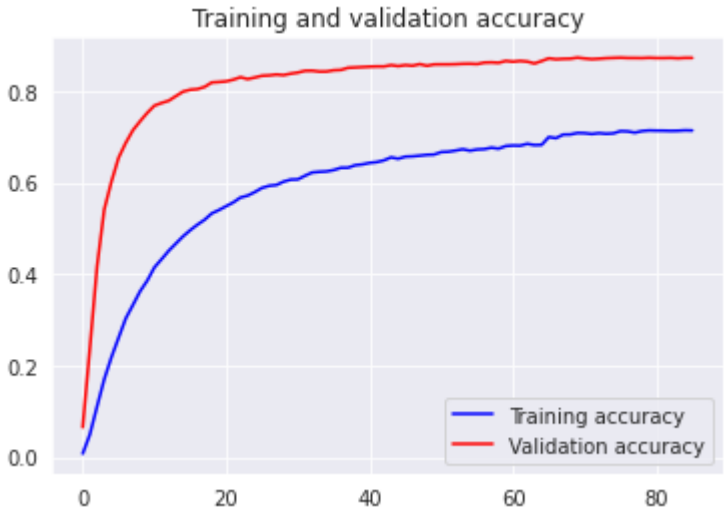
```
results = model.evaluate(test_images, verbose=0)

print("    Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
    Test Loss: 0.52093
Test Accuracy: 87.27%
```

Visualizing loss curves

In [18]:

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')

plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')

plt.title('Training and validation loss')
plt.legend()
plt.show()
```



Making predictions on the Test Data

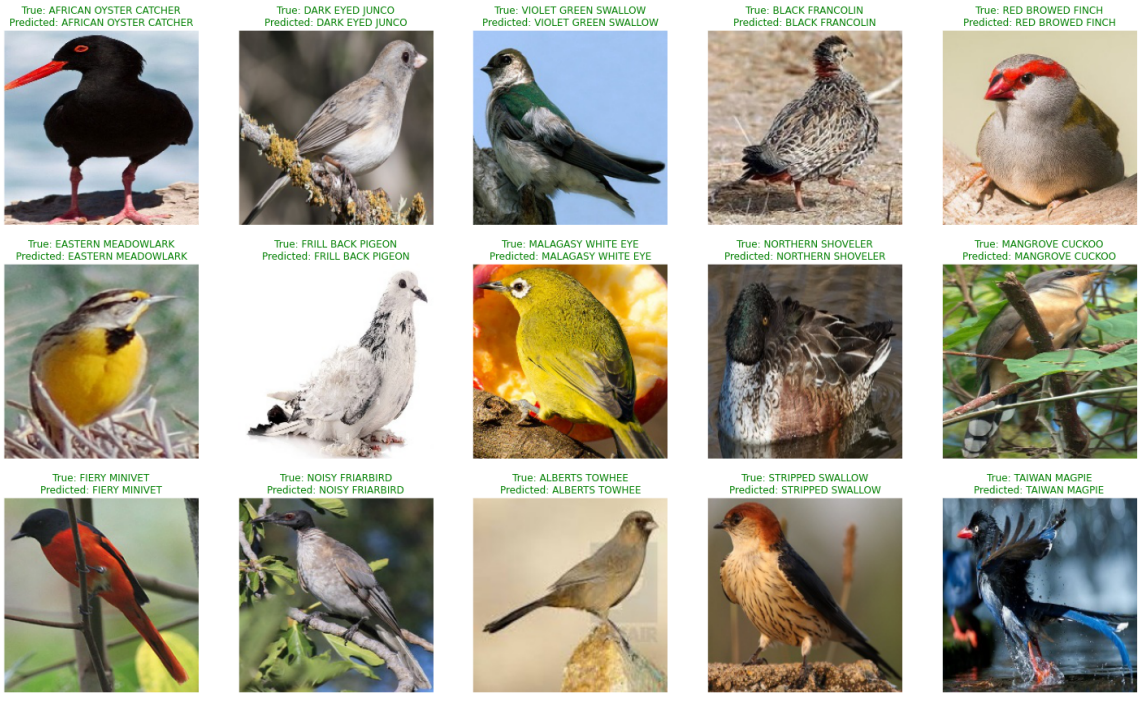
In [19]:

```
# Predict the label of the test_images
pred = model.predict(test_images)
pred = np.argmax(pred,axis=1)

# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]

# Display the result
print(f'The first 5 predictions: {pred[:5]}')
```

```
The first 5 predictions: ['RED CROSSBILL', 'BLACK BREASTED PUFFBIRD', 'CUB
AN TROGON', 'EUROPEAN TURTLE DOVE', 'BLUE COAU']
```

In [20]:

```
  # Display 25 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"True: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred[random_
plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

Plotting the Classification Reports and Confusion Matrix

In [21]:

```
y_test = list(test_df.Label)
print(classification_report(y_test, pred))
```

```
                    BORNEAN BRISTLEHEAD       0.97      0.97      0.97        32
                       BORNEAN LEAFBIRD       0.83      0.88      0.86        34
                       BORNEAN PHEASANT       0.61      0.65      0.63        26
                       BRANDT CORMARANT       0.79      0.87      0.83        31
                     BREWERS BLACKBIRD        0.56      0.60      0.58        40
                          BROWN CREPPER       0.87      0.93      0.90        29
                    BROWN HEADED COWBIRD       0.83      0.73      0.78        48
                             BROWN NOODY       0.94      0.97      0.96        34
                          BROWN THRASHER       0.81      0.97      0.89        36
                              BUFFLEHEAD       0.74      0.79      0.76        33
                       BULWERS PHEASANT        0.94      0.85      0.89        39
                     BURCHELLS COURSER         1.00      0.84      0.91        37
                            BUSH TURKEY        0.95      0.97      0.96        36
                     CAATINGA CACHOLOTE        0.92      0.94      0.93        35
                        CABOTS TRAGOPAN        1.00      0.79      0.89        39
                             CACTUS WREN       0.79      0.96      0.87        24
                     CALIFORNIA CONDOR         0.74      0.72      0.73        36
                       CALIFORNIA GULL         0.90      0.70      0.79        27
                       CALIFORNIA QUAIL        0.90      1.00      0.95        28
                          CAMPO FLICKER        0.91      1.00      0.95        30
```

In [22]:

```
report = classification_report(y_test, pred, output_dict=True)
df = pd.DataFrame(report).transpose()
df
```

Out[22]:

	precision	recall	f1-score	support
ABBOTTS BABBLER	0.866667	0.764706	0.812500	34.000000
ABBOTTS BOOBY	0.714286	0.428571	0.535714	35.000000
ABYSSINIAN GROUND HORNBILL	0.823529	0.823529	0.823529	34.000000
AFRICAN CROWNED CRANE	0.928571	1.000000	0.962963	26.000000
AFRICAN EMERALD CUCKOO	0.962963	0.722222	0.825397	36.000000
...	...	...	...	...
YELLOW HEADED BLACKBIRD	0.900000	1.000000	0.947368	27.000000
ZEBRA DOVE	1.000000	0.967742	0.983607	31.000000
accuracy	0.872748	0.872748	0.872748	0.872748
macro avg	0.874869	0.874508	0.870849	16927.000000
weighted avg	0.878245	0.872748	0.871753	16927.000000

528 rows × 4 columns

Grad-Cam Visualization

In [23]:

```
def get_img_array(img_path, size):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=size)
    array = tf.keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size "size"
    array = np.expand_dims(array, axis=0)
    return array

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]
    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.4):
    # Load the original image
    img = tf.keras.preprocessing.image.load_img(img_path)
    img = tf.keras.preprocessing.image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # Create an image with RGB colorized heatmap
    jet_heatmap = tf.keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = tf.keras.preprocessing.image.img_to_array(jet_heatmap)
```

```
    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = tf.keras.preprocessing.image.array_to_img(superimposed_img)
    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
#     display(Image(cam_path))

    return cam_path


preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions

last_conv_layer_name = "top_conv"
img_size = (224,224, 3)

# Remove last layer's softmax
model.layers[-1].activation = None
```

In [24]:

```
# Display the part of the pictures used by the neural network to classify the pictures
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    img_path = test_df.Filepath.iloc[random_index[i]]
    img_array = preprocess_input(get_img_array(img_path, size=img_size))
    heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
    cam_path = save_and_display_gradcam(img_path, heatmap)
    ax.imshow(plt.imread(cam_path))
    ax.set_title(f"True: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred[random_
plt.tight_layout()
plt.show()
```

In [ ]: