# Bird Classification using CNN and Transfer Learning

In [1]:
```
# Import 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, ReduceLROnPlateau
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


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

Load and Transform Data

In [3]:
```
BATCH_SIZE = 32
TARGET_SIZE = (224, 224)
```

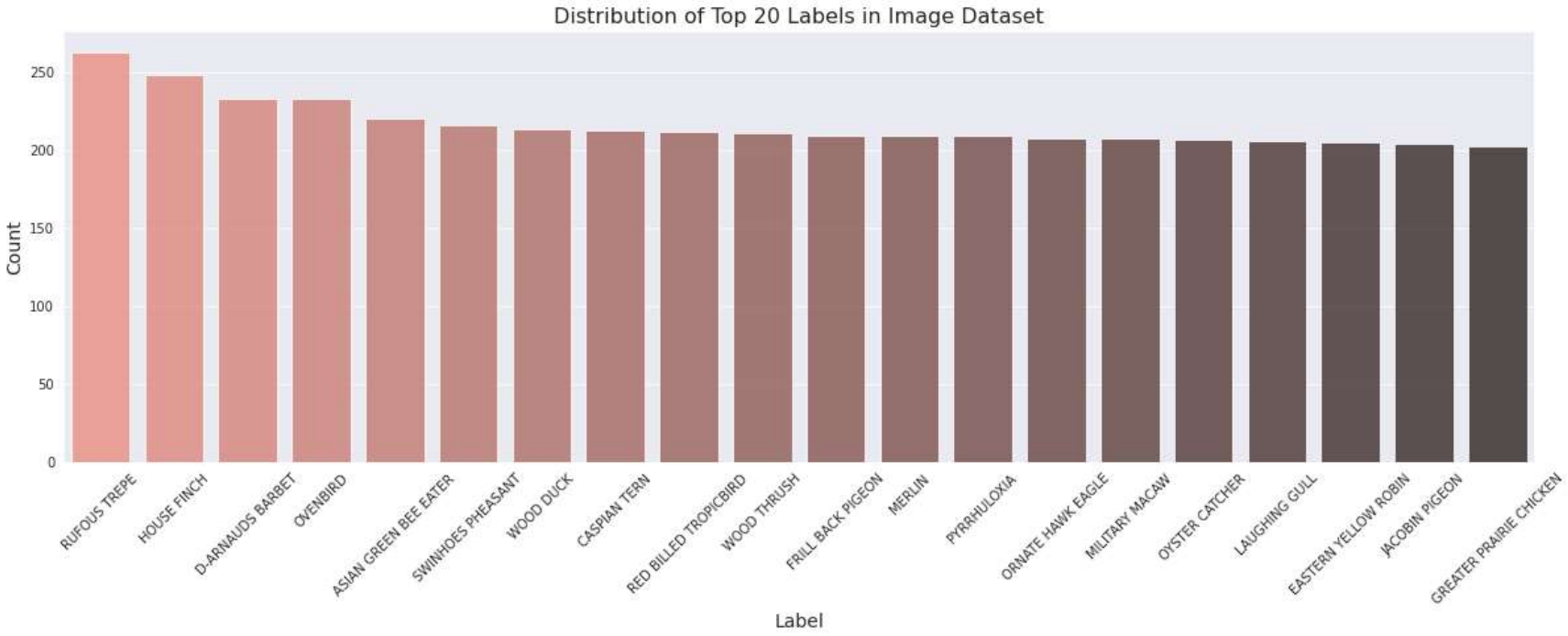
In [ ]:
```
# Walk through each directory
dataset = "../input/100-bird-species/train"
walk_through_dir(dataset);
```

#### Placing Data into a Dataframe

The first column **filepaths** contains the file path location of each individual images. The second column **labels** contains the class label of the corresponding image from the file path</p> 

In [5]:
```
image_dir = Path(dataset)

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

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_r')
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()
```

SNOWY OWL

RUDDY SHELDUCK

GURNEYS PITTA

NORTHERN GOSHAWK

SANDHILL CRANE

GRANDALA

BANDED STILT

RED LEGGED HONEYCREEPER

SAND MARTIN

WOOD THRUSH

DALMATIAN PELICAN

MCKAYS BUNTING

BLACK NECKED STILT

DAURIAN REDSTART

MAGPIE GOOSE

WHITE NECKED RAVEN

Computing Error Rate Analysis

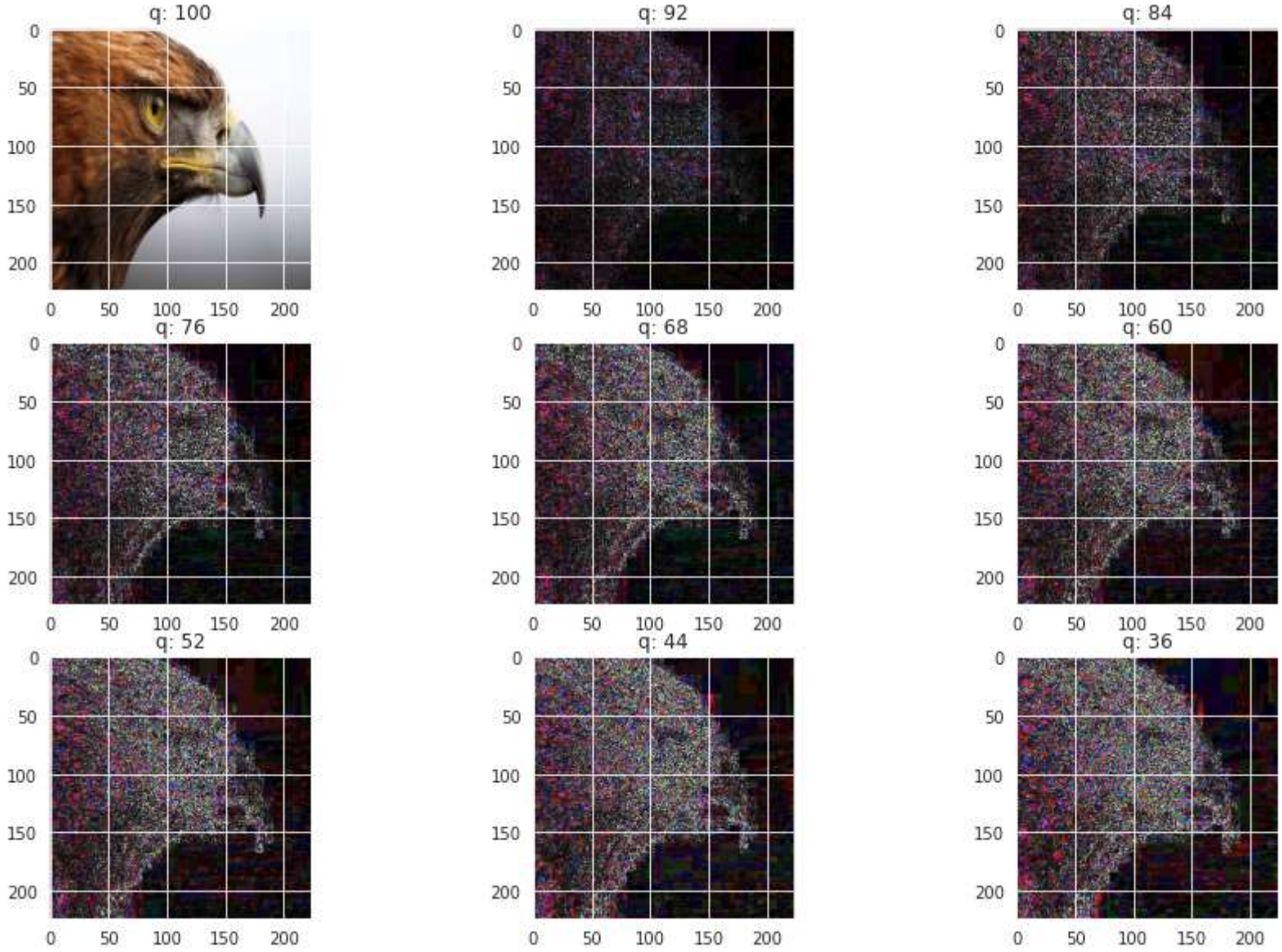
In [8]:

```
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**

The data will be split into three different categories: Training, Validation and Testing. The training data will be used to train the deep learning CNN model and its parameters will be fine tuned with the validation data. Finally, the performance of the data will be evaluated using the test data(data the model has not previously seen).

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=42)
```

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

The model images will be subjected to a pre-trained CNN model called EfficientNetB0. Three callbacks will be utilized to monitor the training. These are: Model Checkpoint, Early Stopping, Tensorboard callback. The summary of the model hyperparameter is shown as follows:

Batch size Epochs Input Shape Output layer

In [14]:
```
# Load the pretrained 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/efficientnetb0_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 3 epochs
early_stopping = EarlyStopping(monitor = "val_loss", # watch the val loss metric
                               patience = 5,
                               restore_best_weights = True) # if val loss decreases for 3 epochs in a row, stop training

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

Train the model

In [17]:

```
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=10,
    callbacks=[
        early_stopping,
        create_tensorboard_callback("training_logs",
                                    "bird_classification"),
        checkpoint_callback,
        reduce_lr
```

```
    ]
)
```

```
Saving TensorBoard log files to: training_logs/bird_classification/20231015-001028
Epoch 1/10
1693/1693 [==============================] - 187s 108ms/step - loss: 6.2139 - accuracy: 0.0092 - val_loss: 5.6838 - val
_accuracy: 0.0606
Epoch 2/10
1693/1693 [==============================] - 174s 102ms/step - loss: 5.3819 - accuracy: 0.0515 - val_loss: 4.1763 - val
_accuracy: 0.2471
Epoch 3/10
1693/1693 [==============================] - 172s 102ms/step - loss: 4.5555 - accuracy: 0.1029 - val_loss: 3.1539 - val
_accuracy: 0.4102
Epoch 4/10
1693/1693 [==============================] - 176s 104ms/step - loss: 4.0132 - accuracy: 0.1550 - val_loss: 2.6117 - val
_accuracy: 0.5217
Epoch 5/10
1693/1693 [==============================] - 180s 106ms/step - loss: 3.6366 - accuracy: 0.2011 - val_loss: 2.2007 - val
_accuracy: 0.5902
Epoch 6/10
1693/1693 [==============================] - 175s 103ms/step - loss: 3.3368 - accuracy: 0.2406 - val_loss: 1.9503 - val
_accuracy: 0.6378
Epoch 7/10
1693/1693 [==============================] - 177s 105ms/step - loss: 3.1148 - accuracy: 0.2755 - val_loss: 1.6897 - val
_accuracy: 0.6714
Epoch 8/10
1693/1693 [==============================] - 178s 105ms/step - loss: 2.9119 - accuracy: 0.3080 - val_loss: 1.5361 - val
_accuracy: 0.6993
Epoch 9/10
1693/1693 [==============================] - 175s 103ms/step - loss: 2.7495 - accuracy: 0.3380 - val_loss: 1.4012 - val
_accuracy: 0.7213
Epoch 10/10
1693/1693 [==============================] - 175s 103ms/step - loss: 2.6104 - accuracy: 0.3624 - val_loss: 1.3168 - val
_accuracy: 0.7343
```

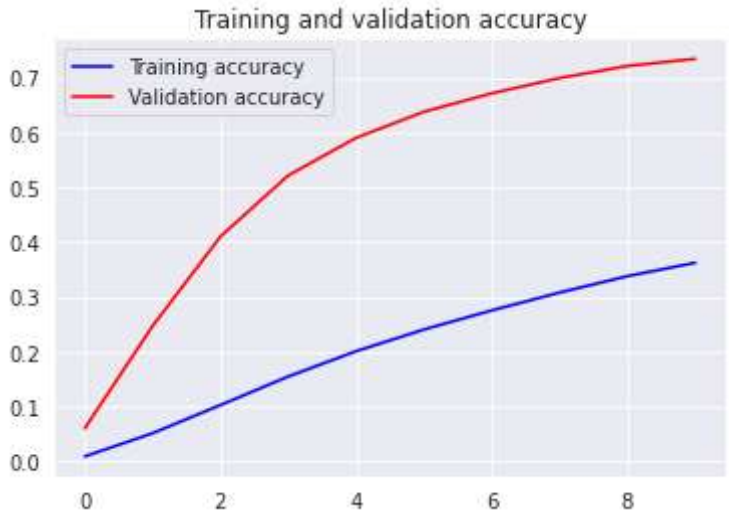
### Model Evaluation

In [18]:
```
results = model.evaluate(test_images, verbose=0)

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

```
    Test Loss: 1.29754
Test Accuracy: 74.22%
```

### Visualizing loss curves

In [19]:
```
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 [20]:
```
# 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', 'CUBAN TROGON', 'EUROPEAN TURTLE DOVE', 'BLUE COA
U']
```

In [21]:
```
# 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):
    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_index[i]]}", color=color)
```

```
plt.show()
plt.tight_layout()
```


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

### Plotting the Classification Reports and Confusion Matrix

In [ ]:
```
y_test = list(test_df.Label)
print(classification_report(y_test, pred))
```

In [23]:
```
report = classification_report(y_test, pred, output_dict=True)
df = pd.DataFrame(report).transpose()
```

```
df
```

Out[23]:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **ABBOTTS BABBLER** | 0.565217 | 0.382353 | 0.456140 | 34.000000 |
| **ABBOTTS BOOBY** | 0.500000 | 0.057143 | 0.102564 | 35.000000 |
| **ABYSSINIAN GROUND HORNBILL** | 0.600000 | 0.705882 | 0.648649 | 34.000000 |
| **AFRICAN CROWNED CRANE** | 0.928571 | 1.000000 | 0.962963 | 26.000000 |
| **AFRICAN EMERALD CUCKOO** | 0.875000 | 0.583333 | 0.700000 | 36.000000 |
| **...** | ... | ... | ... | ... |
| **YELLOW HEADED BLACKBIRD** | 0.727273 | 0.888889 | 0.800000 | 27.000000 |
| **ZEBRA DOVE** | 0.937500 | 0.967742 | 0.952381 | 31.000000 |
| **accuracy** | 0.742187 | 0.742187 | 0.742187 | 0.742187 |
| **macro avg** | 0.759500 | 0.742799 | 0.733582 | 16927.000000 |
| **weighted avg** | 0.762326 | 0.742187 | 0.734778 | 16927.000000 |

528 rows × 4 columns

### Grad-Cam Visualization

Grad-CAM (Gradient-weighted Class Activation Mapping) is a technique used to visualize the regions of an input image that were most relevant for a neural network's prediction. It allows you to see which regions of the image the model focused on while making its prediction. Grad-CAM is a modification of the CAM technique that extends the latter to any model that uses a convolutional neural network (CNN) as its underlying architecture.

In [24]:

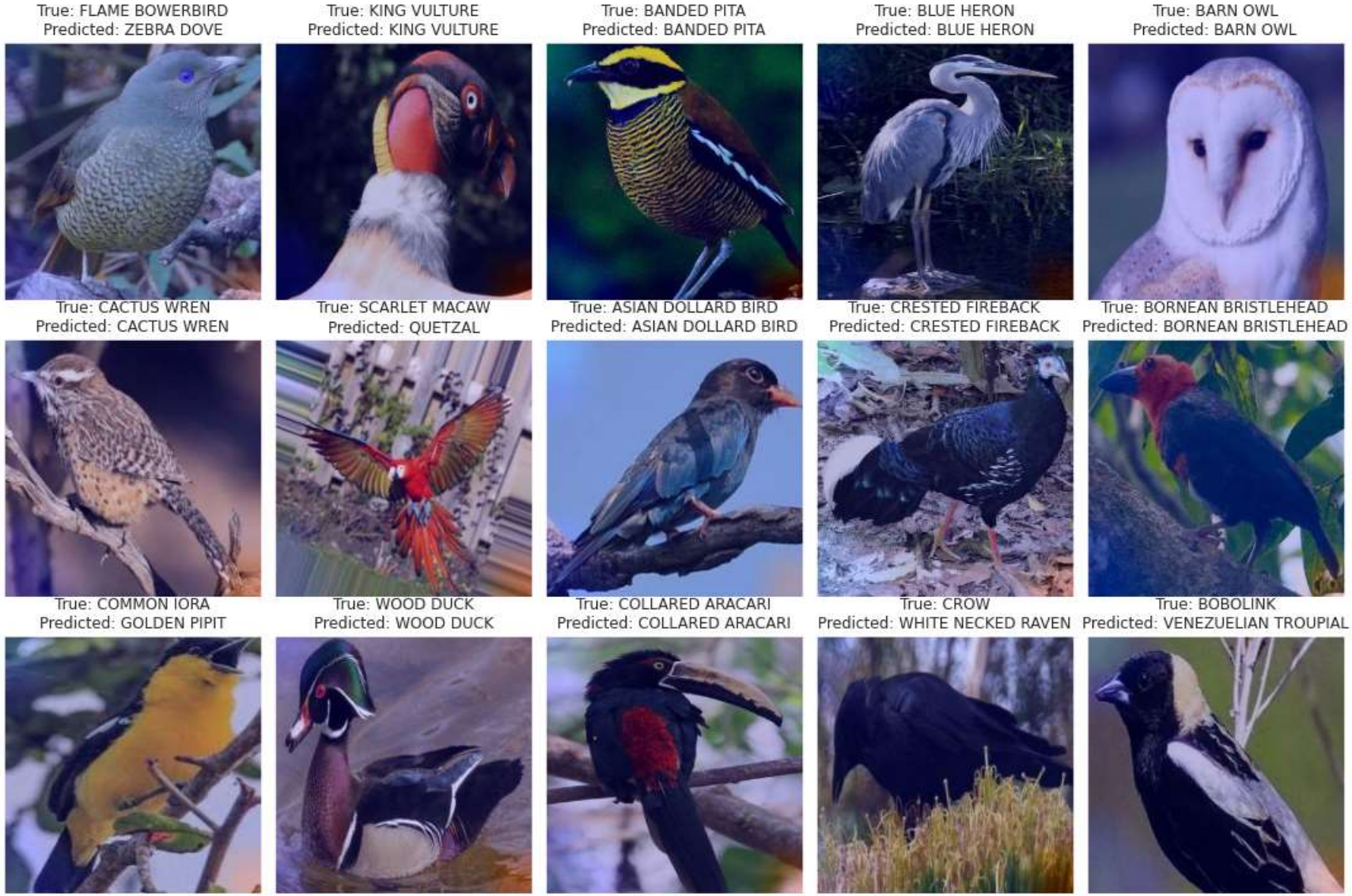
```
def get_img_array(img_path, size):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=size)
    array = tf.keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch" of size "size"
    array = np.expand_dims(array, axis=0)
    return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

Display the part of the pictures used by the neural network to classify the pictures

In [25]:
```
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_index[i]]}")
plt.tight_layout()
plt.show()
```


True: FLAME BOWERBIRD
Predicted: ZEBRA DOVE
True: KING VULTURE
Predicted: KING VULTURE
True: BANDED PITA
Predicted: BANDED PITA
True: BLUE HERON
Predicted: BLUE HERON
True: BARN OWL
Predicted: BARN OWL
True: CACTUS WREN
Predicted: CACTUS WREN
True: SCARLET MACAW
Predicted: QUETZAL
True: ASIAN DOLLARD BIRD
Predicted: ASIAN DOLLARD BIRD
True: CRESTED FIREBACK
Predicted: CRESTED FIREBACK
True: BORNEAN BRISTLEHEAD
Predicted: BORNEAN BRISTLEHEAD
True: COMMON IORA
Predicted: GOLDEN PIPIT
True: WOOD DUCK
Predicted: WOOD DUCK
True: COLLARED ARACARI
Predicted: COLLARED ARACARI
True: CROW
Predicted: WHITE NECKED RAVEN
True: BOBOLINK
Predicted: VENEZUELIAN TROUPIAL