final_comp4531_bottle_mcmanus

March 9, 2024

Bird Species Image Recognition

Brendon Bottle and Sean McManus

[17]:
```
import tensorflow as tf
from keras.layers import Dense, Conv2D, BatchNormalization, MaxPooling2D,␣
 ↪Input, Flatten, Dropout, LeakyReLU
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from sklearn.preprocessing import LabelBinarizer
import keras
from keras.models import Sequential, Model
from keras.callbacks import EarlyStopping, ModelCheckpoint
import random
from keras.applications import EfficientNetB0
import pandas as pd
import seaborn as sns
import warnings
import pickle

warnings.filterwarnings("ignore", "use_inf_as_na")

rs = 7284

np.random.seed(rs)
random.seed(rs)
tf.random.set_seed(rs)
```

1 Load Data

Initial data source: https://www.kaggle.com/datasets/gpiosenka/100-bird-species

The inital dataset contained 100's of images for each of 525 species of bird. However, for learning and gaining knowledge on model architecture and design we decided that was too much data to be able to be constantly running for a small change. As a comprimise we randomly selected 20 different species of bird to include in the model.

The available data for each bird was originally unbalanced, we corrected this by selecting a subset of the available images and creating a balanced dataset. This subset conatined 100 images for each of the 20 different species.

The validation and test sets were hand selected by the author of the kaggle dataset as they were considered good examples of that bird species, so we used the same he had selected as our validation and test sets.

[18]:
```
''' Original code for loading and saving data files'''

# # Set file path for loading data
# train_dir = './birds/train'
# val_dir = "./birds/valid"
# test_dir = "./birds/test"

# # Set the number of images per bird to include
# num_img = 100
# num_birds = 20

# # Instantiate the containers for holding image and label data
# train_data = []
# val_data = []
# test_data = []

# bird_num = 0
# # Load train data
# for i in os.listdir(train_dir):
#     if i.startswith('.'):
#         continue
#     bird_num +=1
#     if bird_num > num_birds:
#         break
#     count = 0
#     sub_directory = os.path.join(train_dir, i)
#     for j in os.listdir(sub_directory):
#         count+=1
#         if count > num_img:
#             break
#         img = cv2.imread(os.path.join(sub_directory, j))
#         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
#         train_data.append([img, i])


# bird_num = 0
# # Load validation data
# for i in os.listdir(val_dir):
#     if i.startswith('.'):
#         continue
```

```
#     bird_num += 1
#     if bird_num > num_birds:
#         break
#     sub_directory = os.path.join(val_dir, i)
#     for j in os.listdir(sub_directory):
#         img = cv2.imread(os.path.join(sub_directory, j))
#         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
#         val_data.append([img, i])

# bird_num = 0
# #Load test data
# for i in os.listdir(test_dir):
#     if i.startswith('.'):
#         continue
#     bird_num += 1
#     if bird_num > num_birds:
#         break
#     sub_directory = os.path.join(test_dir, i)
#     for j in os.listdir(sub_directory):
#         img = cv2.imread(os.path.join(sub_directory, j))
#         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
#         test_data.append([img, i])

# print(len(train_data))
# print(len(val_data))
# print(len(test_data))

# pickle.dump(train_data, open('./saved_data/bird_train_data.pkl', 'wb'))
# pickle.dump(val_data, open('./saved_data/bird_val_data.pkl', 'wb'))
# pickle.dump(test_data, open('./saved_data/bird_test_data.pkl', 'wb'))
```

[18]:
```
' Original code for loading and saving data files'
```

During data loading we had issues with being on different OS platforms, and filestructures. We solved this issue by using a pickle file to ensure images and birds were being loaded in the same order for all users.

[19]:
```
num_birds = 20
num_img = 100

train_data = pickle.load(open('./saved_data/bird_train_data.pkl', 'rb'))
val_data = pickle.load(open('./saved_data/bird_val_data.pkl', 'rb'))
test_data = pickle.load(open('./saved_data/bird_test_data.pkl', 'rb'))
```

[20]:
```
bird_names = ['ALEXANDRINE PARAKEET',
 'ARARIPE MANAKIN',
 'ASIAN CRESTED IBIS',
 'BALD EAGLE',
```

```
 'BALD IBIS',
 'BLUE HERON',
 'BOBOLINK',
 'BORNEAN PHEASANT',
 'CALIFORNIA QUAIL',
 'CAMPO FLICKER',
 'CEDAR WAXWING',
 'COPPERSMITH BARBET',
 'CUBAN TROGON',
 'D-ARNAUDS BARBET',
 'EURASIAN BULLFINCH',
 'EURASIAN GOLDEN ORIOLE',
 'FLAME TANAGER',
 'FRILL BACK PIGEON',
 'GAMBELS QUAIL',
 'GREAT KISKADEE']
```

[21]:
```
# Check if all images are the same shape

count_t = 0
count_v = 0

for t in train_data:
    if t[0].shape != (224, 224, 3):
        count_t += 1

for v in val_data:
    if v[0].shape != (224, 224, 3):
        count_v += 1

print(f'There are {count_t} images of different shape in the train data.\nThere␣
 ↪are {count_v} images of different shape in the validation data.')
```

```
There are 0 images of different shape in the train data.
There are 0 images of different shape in the validation data.
```

2 Explore Data

The random selection of birds includes a wide variety of birds in size, coloration, shape, and poses. The images are already closely cropped, and the birds are genreally centered, but there is a lot of background noise between the images. Particularly for the smaller birds that mostly are perched on top of plants.

[22]:
```
# Show an image for each bird
fig, axs = plt.subplots(int(num_birds/5), 5, figsize=(20, 20))
```

```
axs = axs.ravel()

ax = 0
for idx in np.arange(0, num_birds*num_img, num_img):
    bird = train_data[idx][1]
    img = train_data[idx][0]
    axs[ax].xaxis.set_ticks([])
    axs[ax].yaxis.set_ticks([])
    axs[ax].set_title(bird)
    # Note that cv2 reads images in BGR, but plt.imshow is expecting RGB,␣
 ↪cvtColor reverses the channels so the correct color is shown when plotted.
    axs[ax].imshow(img)
    ax +=1

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

3 Preapre Data for Training

Because this is image data, and we've already confirmed they are the same shape, there is less cleaning than we'd need to do with other types of data. However, we do need to prepare the data for our model. We'll need to do a few things for this:

- Shuffle the data in case the images were loaded in any particular order.
- Split the X features (pixel information) and y labels (bird names)
- Vectorize the y values to allow for machine interpretation.

Because the baseline mdoel we'll use (EfficientNet) has a scaling layer added into it already, we won't normalize the data here. Instead we'll do that when we call model.fit() as needed.

[23]:
```
# Shuffle data to randomize batches

np.random.seed(rs)
np.random.shuffle(train_data)
np.random.shuffle(val_data)
```

[24]:
```
# Preprocess training data and validation data
lb = LabelBinarizer()

X_train = []
y_train = []
for x, y in train_data:
    X_train.append(x)
    y_train.append(y)

X_train = np.array(X_train)
y_train = np.array(y_train)

y_train_vect = lb.fit_transform(y_train)

X_val = []
y_val = []
for x, y in val_data:
    X_val.append(x)
    y_val.append(y)

X_val = np.array(X_val)
y_val = np.array(y_val)

y_val_vect = lb.fit_transform(y_val)
```

4 Function Creation

We'll first create few functions that we'll need for running and evaluating models.

[25]:
```
def set_callbacks(filename, patience= 8, start_from_eopch= 10):

    '''Sets early stopping and model checkpoint variables

        Args:
            filename (str): Filename/path for saving
            patience (int): How many epochs to wait before stopping (default 8)
            start_from_epoch (int): What epoch to start checking validation␣
 ↪accuracy (default 10)

        Returns:
            stopping: EarlyStopping call
            checkpoint: ModelCheckpoint call
    '''

    stopping = EarlyStopping(monitor= 'val_accuracy', patience= 8,␣
 ↪start_from_epoch=10, restore_best_weights= True)
    checkpoint = ModelCheckpoint(filename, monitor = 'val_accuracy',␣
 ↪save_best_only= True)

    return stopping, checkpoint

def plot_loss (mod_hist):

    '''Plots the loss and accuracy for model training

        Args:
            mod_hist: History results from a model.fit() call
    '''

    results_df = pd.DataFrame({'Train Accuracy': mod_hist.history['accuracy'],
                           'Val Accuracy': mod_hist.history['val_accuracy'],
                           'Train Loss': mod_hist.history['loss'],
                           'Val Loss': mod_hist.history['val_loss']
                           })

    fig, axs = plt.subplots(ncols=2, figsize = (12, 4))

    sns.lineplot(results_df['Val Loss'], ax= axs[0], label = 'Val Loss')
    sns.lineplot(results_df['Train Loss'], ax= axs[0], label = 'Train Loss')

    sns.lineplot(results_df['Val Accuracy'], ax= axs[1], label = 'Val Acc')
    sns.lineplot(results_df['Train Accuracy'], ax= axs[1], label = 'Train Acc')
```

```
    # axs[1].set_yticks(np.arange(.1, 1.01, .05))
    # axs[0].set_yticks(np.arange(0, 5.1, .2))

    plt.legend()

    plt.show()

def model_setup(rs=7284):
    '''Sets the 3 seeds necessary for consistency in keras'''
    np.random.seed(rs)
    random.seed(rs)
    tf.random.set_seed(rs)

def resize_img(X_train ,X_val, new_size):
    '''Resizes an array of images

        Args:
            X_train (array): Array of image data for training
            X_val (array): Array of img data for validation
            new_size (tuple): The shape to resize the images to

        Returns:
            X_train_rs (array): An array of the same length as X_train with␣
 ↪images resized
            X_val_rs (array): An array of the same length as X_val with images␣
 ↪resized

    '''
    X_train_rs = []
    X_val_rs = []

    #Resize training data
    for img in X_train:
        rsimg = cv2.resize(img, new_size)
        X_train_rs.append(rsimg)

    X_train_rs = np.array(X_train_rs)

    # Resize validation data
    for img in X_val:
        rsimg = cv2.resize(img, new_size)
        X_val_rs.append(rsimg)

    X_val_rs = np.array(X_val_rs)

    return X_train_rs, X_val_rs
```

```
def pred_eval(model, X = X_val, y= y_val):

    ''' Evaluates predictions for specificed model

        Args:
            model: The model to be used for predictions

        Returns:
            preds (df): A dataframe of the ground truth and predicted class␣
 ↪labels for each observation

    '''

    predictions = np.argmax(model.predict(X), axis=1)

    y_hat = np.array([bird_names[i] for i in predictions])

    preds = pd.DataFrame({'y_true': y,
                                'y_hat': y_hat
                                })

    return preds

def count_correct(preds):

    '''Creates a dummary dataframe counting the total correct predictions for␣
 ↪each bird species

        Args:
            preds (df): The output dataframe from pred_eval()

        Returns:
            correct_count(df): A dataframe with each bird name and the total␣
 ↪number correctly predicted
    '''

    # Look at number of correct predictions for each bird
    preds['Correct'] = preds.apply(lambda x: 0 if x['y_true'] != x['y_hat']␣
 ↪else 1, axis= 1)

    correct_count = pd.DataFrame(preds.groupby(by='y_true').sum()['Correct'])

    return correct_count

def show_class(bird, preds):
```

```
    '''Shows validation images and predicted class for specified species,␣
↪divided by incorrect and correct if applicable

        Args:
            bird (str): Name of bird to show
            preds (df): The output dataframe from pred_eval()
    '''

    # List the indexes for the worst bird correctly and incorrectly predicted␣
↪and show images
    wrong_bird = list(preds[(preds['y_true'] == bird) & (preds['y_hat'] !=␣
↪bird)].index)
    right_bird = list(preds[(preds['y_true'] == bird) & (preds['y_hat'] ==␣
↪bird)].index)

    if len(wrong_bird) > 0:

        print(f'Incorrectly labeled {bird}')
        fig, axs = plt.subplots(ncols=len(wrong_bird), figsize = (15, 5))

        for idx, i in enumerate(wrong_bird):
            if len(wrong_bird) > 1:
                ax = axs[idx]
            elif len(wrong_bird) <= 1:
                ax = axs

            ax.imshow(X_val[i])
            ax.xaxis.set_ticks([])
            ax.xaxis.set_ticks([])
            ax.set_title(preds['y_hat'][i])
            ax.axis('off')

        plt.tight_layout()
        plt.show()

    if len(right_bird) > 0:

        print(f'Correctly labeled {bird}')

        fig, axs = plt.subplots(ncols=len(right_bird), figsize = (10, 5))

        for idx, i in enumerate(right_bird):

            if len(right_bird) > 1:
                ax = axs[idx]
            elif len(right_bird) <= 1:
                ax = axs
```

```
            ax.imshow(X_val[i])
            ax.xaxis.set_ticks([])
            ax.xaxis.set_ticks([])
            ax.set_title(preds['y_hat'][i])
            ax.axis('off')

        plt.tight_layout()
        plt.show()

def plot_birds(birds, num_show):
    '''Shows the selected number of images from teh training set for the␣
 ↪selected species of birds

        Args:
            birds (list): A list of the name(s) of bird(s) that you want to show
            num_show(int): The number of images to show for each species
    '''
    for bird in birds:
        fig, axs = plt.subplots(ncols= num_show, figsize = (20, 5))

        bird_imgs = X_train[y_train == bird][:num_show]

        for idx, b in enumerate(bird_imgs):

            axs[idx].imshow(b)
            axs[idx].xaxis.set_ticks([])
            axs[idx].xaxis.set_ticks([])
            axs[idx].axis('off')

        fig.suptitle(bird)
        plt.tight_layout()
        plt.show()
```

5 Check Baseline

To validate that this is a problem we'll be able to solve with a reasonable level of accuracy, we'll set a baseline expectation for how well our model could potentially perform using EfficientNetB0, a popular pre-trained model used for image classification. We'll just run it for 5 epochs to get an idea of what kind of accuracy we might expect to be able to get from this dataset.

[26]:
```
# Just running a couple epochs on baseline to validate that it should still be␣
 ↪able to get decent results
keras.backend.clear_session()
model_setup()

# Load Model
```

```
enet = EfficientNetB0(include_top= False, input_shape=(224, 224, 3), weights=␣
 ↪'imagenet')

layers = enet.layers

for layer in layers:
    layer.trainable = False

# Add a flatten and softmax layer
base_model= Sequential([
    enet,
    Flatten(),
    Dense(num_birds, activation='softmax')
])

# Compile model
base_model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

base_history = base_model.fit(x = X_train, y= y_train_vect, batch_size=80,␣
 ↪validation_data= (X_val, y_val_vect), verbose = 1, epochs = 5)
```

```
Epoch 1/5
25/25 [==============================] - 19s 612ms/step - loss: 0.9192 -
accuracy: 0.8315 - val_loss: 0.2380 - val_accuracy: 0.9600
Epoch 2/5
25/25 [==============================] - 14s 550ms/step - loss: 0.1177 -
accuracy: 0.9835 - val_loss: 0.0883 - val_accuracy: 0.9800
Epoch 3/5
25/25 [==============================] - 14s 548ms/step - loss: 0.0199 -
accuracy: 0.9960 - val_loss: 0.0258 - val_accuracy: 0.9900
Epoch 4/5
25/25 [==============================] - 14s 580ms/step - loss: 0.0130 -
accuracy: 0.9950 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 5/5
25/25 [==============================] - 13s 533ms/step - loss: 0.0029 -
accuracy: 0.9995 - val_loss: 0.0018 - val_accuracy: 1.0000
```

6 Simple Model

With only 5 epochs, we already had an average accuracy of about 97%, so this is clearly a problem that can be solved very well by a properly trained neural net. To give ourselves a starting point, we'll build a very simple model just using 1 CNN layer, a pooling layer, and a softmax for predictions. We'll use 32 filters, a 3x3 kernel, and rmsprop as an optimizer. Because we balanced our classes, accuracy will be a reasonable metric to use for evaluation, although we will want to check the actual class predictions to determine which classes our model struggles with the most.

We'll use early stopping to limit run times and to avoid overfitting on this model.

[27]:
```
keras.backend.clear_session()

model_setup()

inputs = Input(shape=(224, 224, 3))

cnn1 = Conv2D(32, 3, activation="leaky_relu")(inputs)

pool1 = MaxPooling2D(2)(cnn1)

flat = Flatten()(pool1)

predictions = Dense(num_birds, activation='softmax')(flat)

simp_model = Model(inputs=inputs, outputs=predictions)

simp_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 flatten (Flatten)           (None, 394272)            0

 dense (Dense)               (None, 20)                7885460

=================================================================
Total params: 7886356 (30.08 MB)
Trainable params: 7886356 (30.08 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[28]:
```
# Fit model
stopping, checkpoint = set_callbacks('simple_model.keras')

simp_model.compile(optimizer='rmsprop', loss= 'categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

```
simp_history = simp_model.fit(x = X_train/255, y= y_train_vect, batch_size=128,␣
 ↪validation_data= (X_val/255, y_val_vect), verbose = 0, epochs = 100,␣
 ↪callbacks=[stopping, checkpoint])

simp_model.evaluate(X_val/255, y_val_vect)
```

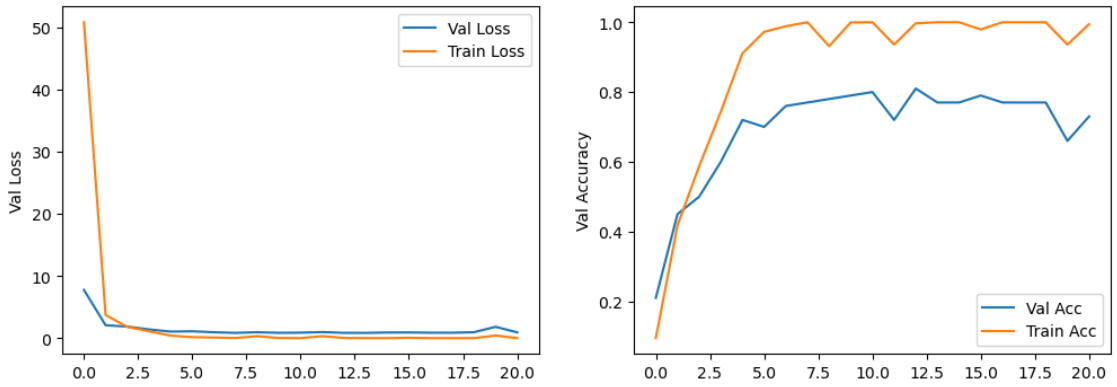
```
4/4 [==============================] - 0s 15ms/step - loss: 0.8571 - accuracy:
0.8100
```

[28]:
```
[0.8571308851242065, 0.8100000023841858]
```

6.1 Model Evaluation

Our model performed suprisingly well for such a simple model. With only 20 epochs we were able to get our model up to 81% accuracy. However, looking at our loss and accuracy over the epochs, we can see that the model is overfitting the training data. Our training accuracy jumps up to 100% relatively quickly while the validation accuracy seems to have leveled out, and is starting to drop as we reach the 20th epoch.

[29]:
```
# Look at epoch loss/accuracy
plot_loss(simp_history)
```



6.2 Prediction Evaluation

Let's now look at what our model actually predicted and how it did on different classes. We can see already that, while the overall accuracy was good, only 4 of our 20 species were correctly predicted 100% of the time and Bald Eagles were incoreectly classified every time. So our models clearly needs some work

[30]:
```
preds = pred_eval(simp_model)
correct_count = count_correct(preds)
correct_count.sort_values(by='Correct')
```

```
4/4 [==============================] - 0s 17ms/step
```

[30]:

y_true	Correct
BALD EAGLE	0
BLUE HERON	1
BALD IBIS	2
CAMPO FLICKER	3
FLAME TANAGER	3
BOBOLINK	3
EURASIAN GOLDEN ORIOLE	3
EURASIAN BULLFINCH	3
D-ARNAUDS BARBET	3
CEDAR WAXWING	3
FRILL BACK PIGEON	4
ALEXANDRINE PARAKEET	4
CALIFORNIA QUAIL	4
BORNEAN PHEASANT	4
ARARIPE MANAKIN	4
GREAT KISKADEE	4
CUBAN TROGON	5
GAMBELS QUAIL	5
ASIAN CRESTED IBIS	5
COPPERSMITH BARBET	5

6.3 Images of Incorrect Bird Classifications

As a first step we will look the images that were misclassified and compare them to their predicted labels to see if there's might be an obvious reason our model misclassified them. The two worst predicted classes were the Bald Eagle and Blue Heron.

For the Bald Eagle, the flying picture being misclassified is understadable as it may not have encountered an in flight eagle in training and unless its learned the coloration of the head and body extremely well it would be challenging to make that leap.

Looking at the training images we we have a lot of images of the Asian Crested Ibis flying, so the simple model likely just connects open wings with that bird. The frill back pigeion appears to be related based on the brown/white coloration.

But we were stumped as to why it would predict the Alexandrine Parakeet on this bald eagle, the parakeet is a brightly colored green bird with a large orange beak.

The Blue Heron was classified as the Frill Back Pigeon, Ibis and Pheasant. We began to think that this was indicative that our simple model was learning the basic color patterns for a species but not so much the structure of the bird itself. This is effect is magnified because the dataset ensures that each bird occupies atleast 50% of the pixels so the scale of the bird will be a less defining feature in our model.

[31]:
```
show_class('BALD EAGLE', preds)

show_class('BLUE HERON', preds)
```

Incorrectly labeled BALD EAGLE

Incorrectly labeled BLUE HERON

Correctly labeled BLUE HERON

[32]:
```
plot_birds(['BALD EAGLE', 'BLUE HERON', 'ASIAN CRESTED IBIS', 'FRILL BACK␣
↪PIGEON', 'ALEXANDRINE PARAKEET'], 6)
```

BLUE HERON

ASIAN CRESTED IBIS

FRILL BACK PIGEON

ALEXANDRINE PARAKEET

6.4 Images of Correct Bird Classifications

We've seen where the model is falling short but where is it performing really well, to get an idea lets look at where the model predicted species correct every time.

Looking at these birds, it does appear that they each have something distinctive about them. For teh Barbet and the Trogon they have very distinctive coloration, whiel the Quail and the Ibis have unique features on their head/face. Additionally, as noted earlier, the Ibis seems to be depicted as flying quite frequently making is distinctive for the pose as well.

[33]:
```
correct_count[(correct_count['Correct']==5) | (correct_count['Correct']==4)]
```

[33]:

y_true	Correct
ALEXANDRINE PARAKEET	4
ARARIPE MANAKIN	4
ASIAN CRESTED IBIS	5
BORNEAN PHEASANT	4
CALIFORNIA QUAIL	4
COPPERSMITH BARBET	5
CUBAN TROGON	5
FRILL BACK PIGEON	4
GAMBELS QUAIL	5
GREAT KISKADEE	4

[34]:
```
# Look at correctly classified birds

show_class('CUBAN TROGON', preds)
show_class('GAMBELS QUAIL', preds)
```

Correctly labeled CUBAN TROGON

Correctly labeled GAMBELS QUAIL

7 Model Complexity Test

Looking where our model is predicting very accurately gives us a few insights: + Coloration seems to be the prominant feature it is learning + Birds with very distinctive features and sharp lines are predicted well

This leads us to believe that our simple model is limited to pretty "shallow" learning, and that's not extremely surprising as we have only a single convolution layer.

In order to predict birds based on more subtle but unique features, we can try to increase our models complexity by adding another set of CCN and Pooling layers.

[35]:
```
keras.backend.clear_session()
model_setup()

comp_model = Sequential()

comp_model.add(Conv2D(32, 3, activation='leaky_relu', input_shape=(224, 224,␣
↪3)))

comp_model.add(MaxPooling2D(2))

comp_model.add(Conv2D(32, 3, activation='leaky_relu'))

comp_model.add(MaxPooling2D(2))

comp_model.add(Conv2D(32, 3, activation='leaky_relu'))

comp_model.add(MaxPooling2D(2))

# comp_model.add(Dropout(.2))

comp_model.add(Flatten())

# comp_model.add(Dense(128, activation='leaky_relu'))

comp_model.add(Dense(num_birds, activation='softmax'))

comp_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)
```

```
 conv2d_1 (Conv2D)           (None, 109, 109, 32)      9248

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 32)        9248

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 32)        0
 g2D)

 flatten (Flatten)           (None, 21632)             0

 dense (Dense)               (None, 20)                432660

=================================================================
Total params: 452052 (1.72 MB)
Trainable params: 452052 (1.72 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[36]:
```
stopping, checkpoint = set_callbacks('complex_model.keras')

comp_model.compile(optimizer='rmsprop', loss= 'categorical_crossentropy',␣
 ↪metrics=['accuracy'])

comp_history = comp_model.fit(x = X_train/255, y= y_train_vect, batch_size=128,␣
 ↪validation_data= (X_val/255, y_val_vect), verbose = 0, epochs = 100,␣
 ↪callbacks=[stopping, checkpoint])

comp_model.evaluate(X_val/255, y_val_vect)
```

```
4/4 [==============================] - 0s 33ms/step - loss: 1.1084 - accuracy:
0.7900
```

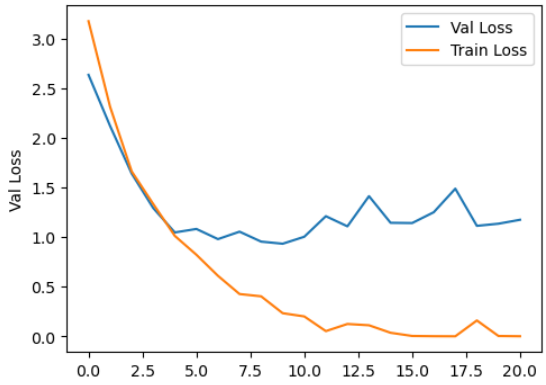
[36]:
```
[1.1083741188049316, 0.7900000214576721]
```

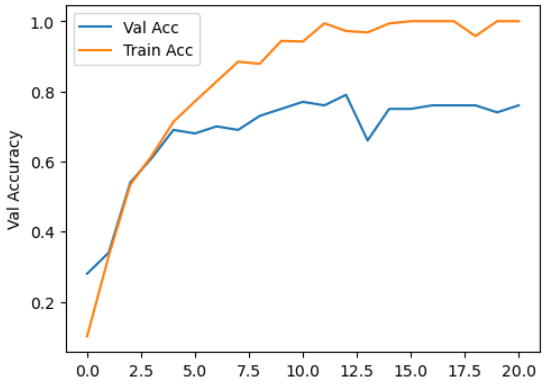
7.1 Model Evaluation

Increasing the complexity actually made our model worse, not better. Our loss increased and our accuracy decreased showing that our model was both worse at predicting correctly, and less confident in it's predictions. We're also seeing that, even with the dropoiut layer, our model is shooting up to near 100% on training very quickly while validation leveled out relatively quickly.

But before we write it off completely, we'll look at whether the specific classes are still having the same issues.

[37]:
```
plot_loss(comp_history)
```





7.2 Prediction Evaluation

Looking at the predictions, while our overall accuracy declined, our individual class accuracies actaully showed some improvements. While the Ibis and the Barbet are no longer being predicted with 100% accuracy, we now have 6 classes with 100% accuracy, and our Bald Eagle class has one correct prediction. So the increased complexity does appear to be helping the model learn more nuanced information about the data.

To evaluate what this model might be doing right, we can again look at the images from the actual classes.

[38]:
```
preds = pred_eval(comp_model)
correct = count_correct(preds)

correct.sort_values('Correct')
```

```
4/4 [==============================] - 0s 30ms/step
```

[38]:

y_true	Correct
BALD EAGLE	1
BALD IBIS	1
EURASIAN GOLDEN ORIOLE	1
BLUE HERON	2
BOBOLINK	3
CEDAR WAXWING	3
D-ARNAUDS BARBET	3
ALEXANDRINE PARAKEET	4
FRILL BACK PIGEON	4
ASIAN CRESTED IBIS	4
FLAME TANAGER	4
EURASIAN BULLFINCH	4
CAMPO FLICKER	5
COPPERSMITH BARBET	5

```
GAMBELS QUAIL                   5
CALIFORNIA QUAIL                5
BORNEAN PHEASANT                5
ARARIPE MANAKIN                 5
CUBAN TROGON                    5
GREAT KISKADEE                  5
```

7.3 Images of Incorrect Classifications

Again, Bald Eagle is doing pretty badly, and while Blue Heron improved, Bald Ibis is doing worse.

For the Bald Eagle, once again the flying picture is a Crested Ibis so it seems like we were correct to think the mdoel is biased towards flight = Crested Ibis and it still thinks picture 3 is the Parakeet. However, the model is no longer confusing it with Frill Back Pigeons, so it seems like there's been improvement there.

The Bald Ibis however seems to be a bit all over the place. I'm not sure exactly what the model is looking at to make those predictions.

[39]:
```
show_class('BALD EAGLE', preds)

show_class('BALD IBIS', preds)
```

```
Incorrectly labeled BALD EAGLE
```


```
Correctly labeled BALD EAGLE
```

BALD EAGLE

Incorrectly labeled BALD IBIS

Correctly labeled BALD IBIS

[40]:
```
plot_birds(['BALD EAGLE', 'ASIAN CRESTED IBIS', 'CALIFORNIA QUAIL',␣
 ↪'ALEXANDRINE PARAKEET', 'CEDAR WAXWING', 'BALD IBIS', 'CAMPO FLICKER',␣
 ↪'COPPERSMITH BARBET'], 6)
```

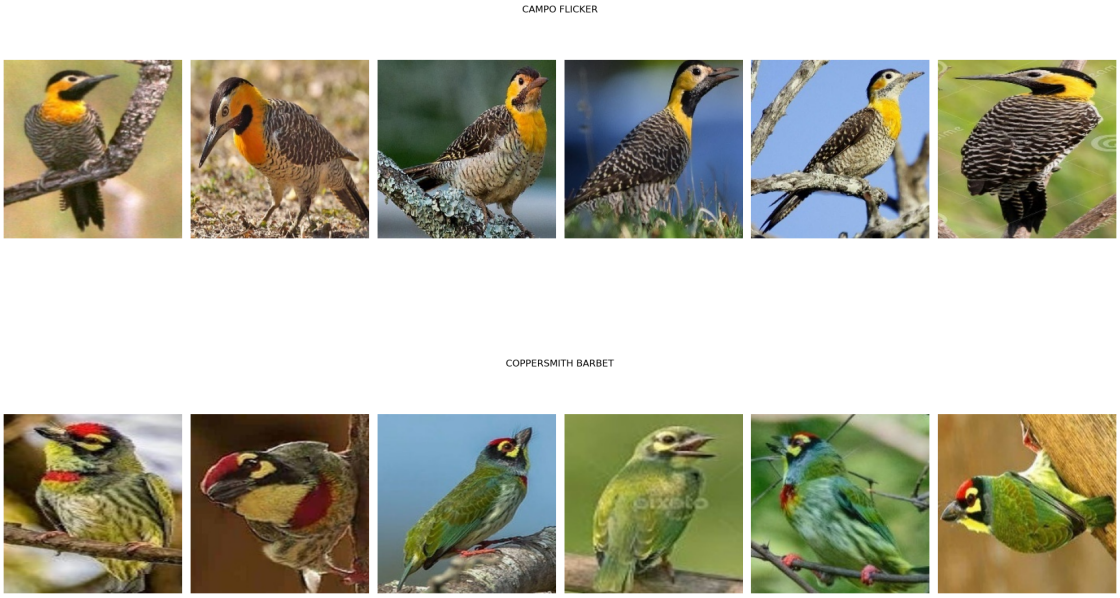
ASIAN CRESTED IBIS

CALIFORNIA QUAIL

ALEXANDRINE PARAKEET

CEDAR WAXWING

BALD IBIS



7.4 Images of Correct Classifications

Let's now look at the birds that were correctly classified and see what the model is doing well.

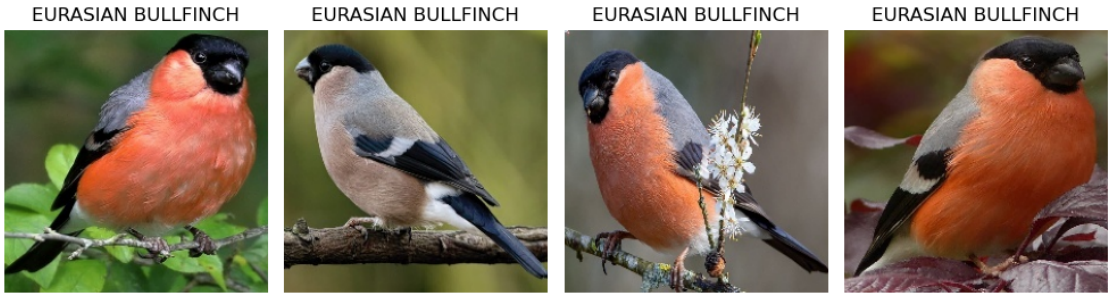
Once again, the majority of correctly classified birds seem to have some specific, distinctive feature. And again, bright color patches seems to be a good indicator of a bird being predicted. This, combined with the models quick jump to overfitting, indicate that the model may be learning these features too well in the early layers, making it inflexible to learning the more nuanced features of the other types of birds. Color in particular seems to be a focus for the model. We cna test this by training the model on grayscale versions of the image and seeing if it still performs at about the same level.

[41]:
```
show_class('GAMBELS QUAIL', preds)
show_class('EURASIAN BULLFINCH', preds)
show_class('CALIFORNIA QUAIL', preds)
show_class('BORNEAN PHEASANT', preds)
show_class('ARARIPE MANAKIN', preds)
show_class('CUBAN TROGON', preds)
```

```
Correctly labeled GAMBELS QUAIL
```

Incorrectly labeled EURASIAN BULLFINCH

Correctly labeled EURASIAN BULLFINCH



Correctly labeled CALIFORNIA QUAIL

Correctly labeled BORNEAN PHEASANT

Correctly labeled ARARIPE MANAKIN

```
Correctly labeled CUBAN TROGON
```

8 Grayscale Model Testing

INSERT SEAN'S CODE HERE

Findings: Removing color tanks the model's accuracy, indicating it was a key contributor to the model. Because color is obviously a relevant feature, we can't eliminate it entirely, but we can mitigate it's effect by adjusting what the image is pulling in when…

9 Progressive Resizing Model

Because the model is pulling in these distinctive features early in the process, which appears to prevent it from learning more diverse features, we can try and mitigate it's effect through a method called Progressive Resizing. This involves scaling down the original image, training a model on that, and then iteratively adding additional layers that have been trained on progressively larger images. The idea is that you can prevent the layers later in the model from learning the images too well, by locking their weights on the more downscaled versions.

We'll try a 3 tier resizing model starting with 56x56 images, and doubling them with each additional model.

9.1 56x56

Because we are scaling the image down so much, I want to counteract the impact of losing the detailed information by using a more complex model. We'll start with the complex model we built, add another set of CNN/Pooling layers and increase filter sizes to 128, 64, and 32 respectively. I'll also still retain the dropout layer to aid in overfitting.

Between the dropout layer and the downscaled images, our concern now is actually that we'll underfit the model. To prevent this, I'll drop the early stopping call and run each model for 100 epochs to find the best option.

[42]:
```
# Resize imags to 56x56

X_train_56, X_val_56 = resize_img(X_train, X_val, (56, 56))

fig, axs = plt.subplots(ncols=2, figsize = (10, 10))
```

```
# Compare to original data

axs[0].imshow(X_train_56[5])
axs[0].axis('off')

axs[1].imshow(X_train[5])
axs[1].axis('off')
plt.show()
```

[43]:
```
# Create model to train 56x56 images

keras.backend.clear_session()
model_setup()

model = Sequential()

# Need to add 2nd conv2D layer so that it can become the new input layer,␣
 ↪setting padding as same to maintain matrix sizes
model.add(Conv2D(32, 3, activation='leaky_relu', padding='same',␣
 ↪input_shape=(56, 56, 3)))

model.add(Conv2D(128, 3, activation='leaky_relu', padding='same'))

model.add(MaxPooling2D(2))

model.add(Conv2D(64, 3, activation='leaky_relu'))

model.add(MaxPooling2D(2))
```

```
model.add(Conv2D(32, 3, activation='leaky_relu'))

model.add(MaxPooling2D(2))

model.add(Dropout(.2))

model.add(Flatten())

model.add(Dense(128, activation='leaky_relu'))

model.add(Dense(num_birds, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 56, 56, 32)        896

 conv2d_1 (Conv2D)           (None, 56, 56, 128)       36992

 max_pooling2d (MaxPooling2  (None, 28, 28, 128)       0
 D)

 conv2d_2 (Conv2D)           (None, 26, 26, 64)        73792

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 32)        18464

 max_pooling2d_2 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 dropout (Dropout)           (None, 5, 5, 32)          0

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 128)               102528

 dense_1 (Dense)             (None, 20)                2580

=================================================================
Total params: 235252 (918.95 KB)
Trainable params: 235252 (918.95 KB)
Non-trainable params: 0 (0.00 Byte)
```

---

[44]:
```
# Set callbacks
stopping, checkpoint = set_callbacks('model_56.keras', patience=20)

# Build and bit model
model.compile(optimizer='RMSProp', loss='categorical_crossentropy',␣
 ↪metrics='accuracy')

history_56 = model.fit(x = X_train_56/255, y= y_train_vect, batch_size=128,␣
 ↪validation_data= (X_val_56/255, y_val_vect), verbose = 1, epochs = 100,␣
 ↪callbacks=[checkpoint])
```

```
Epoch 1/100
16/16 [==============================] - 6s 326ms/step - loss: 3.0073 -
accuracy: 0.0500 - val_loss: 2.9782 - val_accuracy: 0.0400
Epoch 2/100
16/16 [==============================] - 5s 335ms/step - loss: 2.9645 -
accuracy: 0.1075 - val_loss: 3.0627 - val_accuracy: 0.1000
Epoch 3/100
16/16 [==============================] - 6s 343ms/step - loss: 2.6709 -
accuracy: 0.1970 - val_loss: 2.3822 - val_accuracy: 0.2700
Epoch 4/100
16/16 [==============================] - 6s 362ms/step - loss: 2.4312 -
accuracy: 0.2665 - val_loss: 2.1243 - val_accuracy: 0.3100
Epoch 5/100
16/16 [==============================] - 6s 351ms/step - loss: 2.0374 -
accuracy: 0.3850 - val_loss: 1.6841 - val_accuracy: 0.4800
Epoch 6/100
16/16 [==============================] - 5s 334ms/step - loss: 1.8221 -
accuracy: 0.4540 - val_loss: 1.7722 - val_accuracy: 0.4500
Epoch 7/100
16/16 [==============================] - 6s 355ms/step - loss: 1.6115 -
accuracy: 0.5155 - val_loss: 1.4120 - val_accuracy: 0.5300
Epoch 8/100
16/16 [==============================] - 6s 348ms/step - loss: 1.4363 -
accuracy: 0.5615 - val_loss: 1.2717 - val_accuracy: 0.5800
Epoch 9/100
16/16 [==============================] - 5s 342ms/step - loss: 1.2937 -
accuracy: 0.6090 - val_loss: 1.0209 - val_accuracy: 0.6200
Epoch 10/100
16/16 [==============================] - 5s 337ms/step - loss: 1.1387 -
accuracy: 0.6580 - val_loss: 1.0289 - val_accuracy: 0.6600
Epoch 11/100
16/16 [==============================] - 5s 315ms/step - loss: 1.0393 -
accuracy: 0.6810 - val_loss: 0.9779 - val_accuracy: 0.6700
Epoch 12/100
16/16 [==============================] - 5s 345ms/step - loss: 0.8897 -
```

```
accuracy: 0.7210 - val_loss: 0.9631 - val_accuracy: 0.6900
Epoch 13/100
16/16 [==============================] - 5s 337ms/step - loss: 0.8545 -
accuracy: 0.7380 - val_loss: 0.8576 - val_accuracy: 0.7400
Epoch 14/100
16/16 [==============================] - 5s 315ms/step - loss: 0.7242 -
accuracy: 0.7700 - val_loss: 0.7977 - val_accuracy: 0.7300
Epoch 15/100
16/16 [==============================] - 6s 347ms/step - loss: 0.6338 -
accuracy: 0.8045 - val_loss: 0.7574 - val_accuracy: 0.7200
Epoch 16/100
16/16 [==============================] - 5s 329ms/step - loss: 0.5712 -
accuracy: 0.8190 - val_loss: 0.8890 - val_accuracy: 0.7300
Epoch 17/100
16/16 [==============================] - 5s 339ms/step - loss: 0.5249 -
accuracy: 0.8300 - val_loss: 0.8332 - val_accuracy: 0.7700
Epoch 18/100
16/16 [==============================] - 5s 338ms/step - loss: 0.4698 -
accuracy: 0.8585 - val_loss: 1.0475 - val_accuracy: 0.7000
Epoch 19/100
16/16 [==============================] - 5s 335ms/step - loss: 0.4022 -
accuracy: 0.8735 - val_loss: 0.5625 - val_accuracy: 0.7600
Epoch 20/100
16/16 [==============================] - 5s 324ms/step - loss: 0.4051 -
accuracy: 0.8640 - val_loss: 0.7539 - val_accuracy: 0.7900
Epoch 21/100
16/16 [==============================] - 5s 335ms/step - loss: 0.3025 -
accuracy: 0.9060 - val_loss: 0.6055 - val_accuracy: 0.8400
Epoch 22/100
16/16 [==============================] - 6s 357ms/step - loss: 0.2986 -
accuracy: 0.9060 - val_loss: 0.6363 - val_accuracy: 0.8200
Epoch 23/100
16/16 [==============================] - 6s 353ms/step - loss: 0.3117 -
accuracy: 0.9000 - val_loss: 0.6322 - val_accuracy: 0.8000
Epoch 24/100
16/16 [==============================] - 6s 352ms/step - loss: 0.2161 -
accuracy: 0.9320 - val_loss: 0.6556 - val_accuracy: 0.8200
Epoch 25/100
16/16 [==============================] - 6s 360ms/step - loss: 0.1859 -
accuracy: 0.9385 - val_loss: 0.7339 - val_accuracy: 0.8200
Epoch 26/100
16/16 [==============================] - 6s 349ms/step - loss: 0.2578 -
accuracy: 0.9155 - val_loss: 0.5532 - val_accuracy: 0.8100
Epoch 27/100
16/16 [==============================] - 6s 370ms/step - loss: 0.1730 -
accuracy: 0.9425 - val_loss: 0.9375 - val_accuracy: 0.7200
Epoch 28/100
16/16 [==============================] - 6s 344ms/step - loss: 0.1168 -
```

```
accuracy: 0.9655 - val_loss: 0.5640 - val_accuracy: 0.8400
Epoch 29/100
16/16 [==============================] - 6s 354ms/step - loss: 0.1975 -
accuracy: 0.9400 - val_loss: 0.5186 - val_accuracy: 0.8500
Epoch 30/100
16/16 [==============================] - 6s 358ms/step - loss: 0.0973 -
accuracy: 0.9725 - val_loss: 0.5338 - val_accuracy: 0.8400
Epoch 31/100
16/16 [==============================] - 5s 327ms/step - loss: 0.1288 -
accuracy: 0.9580 - val_loss: 0.6253 - val_accuracy: 0.7900
Epoch 32/100
16/16 [==============================] - 5s 331ms/step - loss: 0.1275 -
accuracy: 0.9590 - val_loss: 0.5819 - val_accuracy: 0.8500
Epoch 33/100
16/16 [==============================] - 5s 327ms/step - loss: 0.1259 -
accuracy: 0.9635 - val_loss: 0.8129 - val_accuracy: 0.8100
Epoch 34/100
16/16 [==============================] - 6s 349ms/step - loss: 0.0964 -
accuracy: 0.9695 - val_loss: 0.8701 - val_accuracy: 0.8000
Epoch 35/100
16/16 [==============================] - 5s 342ms/step - loss: 0.0967 -
accuracy: 0.9685 - val_loss: 0.7481 - val_accuracy: 0.8000
Epoch 36/100
16/16 [==============================] - 5s 337ms/step - loss: 0.1018 -
accuracy: 0.9690 - val_loss: 0.8254 - val_accuracy: 0.7900
Epoch 37/100
16/16 [==============================] - 6s 352ms/step - loss: 0.1399 -
accuracy: 0.9630 - val_loss: 0.8799 - val_accuracy: 0.7600
Epoch 38/100
16/16 [==============================] - 5s 314ms/step - loss: 0.0854 -
accuracy: 0.9775 - val_loss: 0.4658 - val_accuracy: 0.8300
Epoch 39/100
16/16 [==============================] - 6s 346ms/step - loss: 0.0808 -
accuracy: 0.9775 - val_loss: 0.6500 - val_accuracy: 0.8400
Epoch 40/100
16/16 [==============================] - 5s 329ms/step - loss: 0.1341 -
accuracy: 0.9570 - val_loss: 0.5405 - val_accuracy: 0.8500
Epoch 41/100
16/16 [==============================] - 5s 334ms/step - loss: 0.0440 -
accuracy: 0.9865 - val_loss: 0.6677 - val_accuracy: 0.8400
Epoch 42/100
16/16 [==============================] - 6s 346ms/step - loss: 0.0533 -
accuracy: 0.9850 - val_loss: 0.6049 - val_accuracy: 0.8300
Epoch 43/100
16/16 [==============================] - 5s 342ms/step - loss: 0.0900 -
accuracy: 0.9745 - val_loss: 0.5626 - val_accuracy: 0.8600
Epoch 44/100
16/16 [==============================] - 5s 334ms/step - loss: 0.0583 -
```

```
accuracy: 0.9845 - val_loss: 1.0961 - val_accuracy: 0.7700
Epoch 45/100
16/16 [==============================] - 6s 355ms/step - loss: 0.1031 -
accuracy: 0.9685 - val_loss: 0.6527 - val_accuracy: 0.8300
Epoch 46/100
16/16 [==============================] - 6s 347ms/step - loss: 0.0549 -
accuracy: 0.9840 - val_loss: 0.5795 - val_accuracy: 0.8300
Epoch 47/100
16/16 [==============================] - 6s 346ms/step - loss: 0.0628 -
accuracy: 0.9795 - val_loss: 0.5320 - val_accuracy: 0.8600
Epoch 48/100
16/16 [==============================] - 6s 356ms/step - loss: 0.0340 -
accuracy: 0.9880 - val_loss: 0.7774 - val_accuracy: 0.7900
Epoch 49/100
16/16 [==============================] - 6s 347ms/step - loss: 0.0741 -
accuracy: 0.9780 - val_loss: 0.6108 - val_accuracy: 0.8500
Epoch 50/100
16/16 [==============================] - 6s 349ms/step - loss: 0.0643 -
accuracy: 0.9830 - val_loss: 0.5499 - val_accuracy: 0.8200
Epoch 51/100
16/16 [==============================] - 5s 339ms/step - loss: 0.0429 -
accuracy: 0.9840 - val_loss: 0.8869 - val_accuracy: 0.8100
Epoch 52/100
16/16 [==============================] - 5s 315ms/step - loss: 0.0433 -
accuracy: 0.9890 - val_loss: 0.5851 - val_accuracy: 0.8800
Epoch 53/100
16/16 [==============================] - 6s 367ms/step - loss: 0.0394 -
accuracy: 0.9855 - val_loss: 0.5607 - val_accuracy: 0.8500
Epoch 54/100
16/16 [==============================] - 6s 356ms/step - loss: 0.0497 -
accuracy: 0.9810 - val_loss: 0.6130 - val_accuracy: 0.8500
Epoch 55/100
16/16 [==============================] - 6s 348ms/step - loss: 0.0486 -
accuracy: 0.9830 - val_loss: 0.6125 - val_accuracy: 0.8700
Epoch 56/100
16/16 [==============================] - 6s 355ms/step - loss: 0.0347 -
accuracy: 0.9910 - val_loss: 0.5728 - val_accuracy: 0.8800
Epoch 57/100
16/16 [==============================] - 6s 348ms/step - loss: 0.0819 -
accuracy: 0.9805 - val_loss: 0.6926 - val_accuracy: 0.8200
Epoch 58/100
16/16 [==============================] - 5s 329ms/step - loss: 0.0174 -
accuracy: 0.9950 - val_loss: 0.8175 - val_accuracy: 0.8200
Epoch 59/100
16/16 [==============================] - 5s 322ms/step - loss: 0.0701 -
accuracy: 0.9775 - val_loss: 0.4770 - val_accuracy: 0.8700
Epoch 60/100
16/16 [==============================] - 5s 320ms/step - loss: 0.0334 -
```

```
accuracy: 0.9895 - val_loss: 0.5404 - val_accuracy: 0.8800
Epoch 61/100
16/16 [==============================] - 5s 312ms/step - loss: 0.0612 -
accuracy: 0.9810 - val_loss: 0.8738 - val_accuracy: 0.8200
Epoch 62/100
16/16 [==============================] - 5s 336ms/step - loss: 0.0546 -
accuracy: 0.9845 - val_loss: 0.5868 - val_accuracy: 0.8200
Epoch 63/100
16/16 [==============================] - 5s 340ms/step - loss: 0.0175 -
accuracy: 0.9950 - val_loss: 0.5928 - val_accuracy: 0.8300
Epoch 64/100
16/16 [==============================] - 5s 315ms/step - loss: 0.0745 -
accuracy: 0.9795 - val_loss: 0.9999 - val_accuracy: 0.7800
Epoch 65/100
16/16 [==============================] - 5s 341ms/step - loss: 0.0319 -
accuracy: 0.9905 - val_loss: 0.5728 - val_accuracy: 0.8600
Epoch 66/100
16/16 [==============================] - 5s 336ms/step - loss: 0.0410 -
accuracy: 0.9860 - val_loss: 0.7975 - val_accuracy: 0.7900
Epoch 67/100
16/16 [==============================] - 5s 311ms/step - loss: 0.0312 -
accuracy: 0.9905 - val_loss: 0.9044 - val_accuracy: 0.7800
Epoch 68/100
16/16 [==============================] - 5s 341ms/step - loss: 0.0546 -
accuracy: 0.9830 - val_loss: 0.8682 - val_accuracy: 0.8200
Epoch 69/100
16/16 [==============================] - 6s 359ms/step - loss: 0.0196 -
accuracy: 0.9965 - val_loss: 0.7950 - val_accuracy: 0.8200
Epoch 70/100
16/16 [==============================] - 6s 356ms/step - loss: 0.0546 -
accuracy: 0.9850 - val_loss: 0.9876 - val_accuracy: 0.7900
Epoch 71/100
16/16 [==============================] - 6s 361ms/step - loss: 0.0401 -
accuracy: 0.9870 - val_loss: 0.8698 - val_accuracy: 0.8000
Epoch 72/100
16/16 [==============================] - 5s 330ms/step - loss: 0.0237 -
accuracy: 0.9905 - val_loss: 1.1388 - val_accuracy: 0.7400
Epoch 73/100
16/16 [==============================] - 5s 334ms/step - loss: 0.0456 -
accuracy: 0.9880 - val_loss: 1.1686 - val_accuracy: 0.8000
Epoch 74/100
16/16 [==============================] - 6s 385ms/step - loss: 0.1286 -
accuracy: 0.9685 - val_loss: 0.8205 - val_accuracy: 0.8400
Epoch 75/100
16/16 [==============================] - 6s 360ms/step - loss: 0.0120 -
accuracy: 0.9960 - val_loss: 0.6297 - val_accuracy: 0.8600
Epoch 76/100
16/16 [==============================] - 6s 383ms/step - loss: 0.0244 -
```

```
accuracy: 0.9930 - val_loss: 0.6523 - val_accuracy: 0.8400
Epoch 77/100
16/16 [==============================] - 6s 406ms/step - loss: 0.0079 -
accuracy: 0.9985 - val_loss: 0.6554 - val_accuracy: 0.8500
Epoch 78/100
16/16 [==============================] - 7s 413ms/step - loss: 0.0439 -
accuracy: 0.9895 - val_loss: 0.8199 - val_accuracy: 0.8500
Epoch 79/100
16/16 [==============================] - 6s 393ms/step - loss: 0.0216 -
accuracy: 0.9935 - val_loss: 0.7087 - val_accuracy: 0.8300
Epoch 80/100
16/16 [==============================] - 6s 388ms/step - loss: 0.0206 -
accuracy: 0.9950 - val_loss: 0.8979 - val_accuracy: 0.8400
Epoch 81/100
16/16 [==============================] - 5s 319ms/step - loss: 0.0569 -
accuracy: 0.9840 - val_loss: 0.9490 - val_accuracy: 0.8200
Epoch 82/100
16/16 [==============================] - 6s 347ms/step - loss: 0.0300 -
accuracy: 0.9915 - val_loss: 0.8002 - val_accuracy: 0.8300
Epoch 83/100
16/16 [==============================] - 6s 344ms/step - loss: 0.0151 -
accuracy: 0.9955 - val_loss: 1.4066 - val_accuracy: 0.7100
Epoch 84/100
16/16 [==============================] - 5s 343ms/step - loss: 0.0483 -
accuracy: 0.9905 - val_loss: 0.7865 - val_accuracy: 0.8600
Epoch 85/100
16/16 [==============================] - 6s 359ms/step - loss: 0.0344 -
accuracy: 0.9925 - val_loss: 0.9383 - val_accuracy: 0.8200
Epoch 86/100
16/16 [==============================] - 5s 329ms/step - loss: 0.0416 -
accuracy: 0.9885 - val_loss: 0.8063 - val_accuracy: 0.8900
Epoch 87/100
16/16 [==============================] - 5s 326ms/step - loss: 0.0278 -
accuracy: 0.9905 - val_loss: 0.7575 - val_accuracy: 0.8500
Epoch 88/100
16/16 [==============================] - 6s 345ms/step - loss: 0.0262 -
accuracy: 0.9905 - val_loss: 0.8464 - val_accuracy: 0.8500
Epoch 89/100
16/16 [==============================] - 5s 332ms/step - loss: 0.0237 -
accuracy: 0.9930 - val_loss: 0.5249 - val_accuracy: 0.8900
Epoch 90/100
16/16 [==============================] - 6s 353ms/step - loss: 0.0516 -
accuracy: 0.9860 - val_loss: 0.6892 - val_accuracy: 0.7900
Epoch 91/100
16/16 [==============================] - 6s 347ms/step - loss: 0.0188 -
accuracy: 0.9955 - val_loss: 0.7217 - val_accuracy: 0.8400
Epoch 92/100
16/16 [==============================] - 5s 316ms/step - loss: 0.0247 -
```

```
accuracy: 0.9925 - val_loss: 0.9539 - val_accuracy: 0.8800
Epoch 93/100
16/16 [==============================] - 6s 347ms/step - loss: 0.0236 -
accuracy: 0.9940 - val_loss: 1.3058 - val_accuracy: 0.8000
Epoch 94/100
16/16 [==============================] - 6s 350ms/step - loss: 0.0428 -
accuracy: 0.9870 - val_loss: 1.3770 - val_accuracy: 0.7800
Epoch 95/100
16/16 [==============================] - 6s 352ms/step - loss: 0.0153 -
accuracy: 0.9960 - val_loss: 0.8698 - val_accuracy: 0.8300
Epoch 96/100
16/16 [==============================] - 5s 332ms/step - loss: 0.0079 -
accuracy: 0.9985 - val_loss: 1.5114 - val_accuracy: 0.8200
Epoch 97/100
16/16 [==============================] - 6s 354ms/step - loss: 0.0394 -
accuracy: 0.9855 - val_loss: 1.1668 - val_accuracy: 0.7800
Epoch 98/100
16/16 [==============================] - 5s 311ms/step - loss: 0.0063 -
accuracy: 0.9980 - val_loss: 0.8333 - val_accuracy: 0.8400
Epoch 99/100
16/16 [==============================] - 5s 310ms/step - loss: 0.0632 -
accuracy: 0.9795 - val_loss: 0.7232 - val_accuracy: 0.8100
Epoch 100/100
16/16 [==============================] - 5s 315ms/step - loss: 0.0272 -
accuracy: 0.9930 - val_loss: 0.8502 - val_accuracy: 0.8600
```

[45]:
```
model_56 = keras.saving.load_model('model_56.keras')

model_56.evaluate(X_val_56/255, y_val_vect)
```

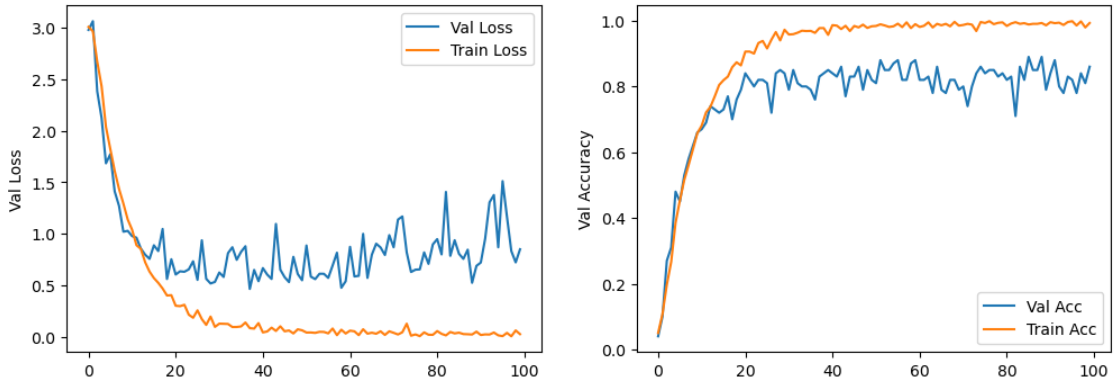
```
4/4 [==============================] - 0s 20ms/step - loss: 0.8063 - accuracy:
0.8900
```

[45]:
```
[0.8062862157821655, 0.8899999856948853]
```

9.1.1 Model Evaluation

Downscaling the images has given us a significant boost in accuracy and improved our loss as well. This is great news and exactly what we were hoping to see. While the model is still reaching near 100% on the training relatively early, we're seeing the validation and training scores aligned longer and we're still getting validation improvement in later epochs thanks to the dropout.

[46]:
```
plot_loss(history_56)
```



9.1.2 Prediction Evaluations

While the model is still struggling with Bald Eagles and Bald Ibis', we've now increased our total number of correctly classified species to 9, meaning almost half of our classes are being correctly predicted. This is looking like a good start for this process.

[47]:
```
preds = pred_eval(model_56, X= X_val_56)
correct = count_correct(preds)

correct.sort_values('Correct')
```

```
4/4 [==============================] - 0s 19ms/step
```

[47]:

y_true	Correct
BALD EAGLE	1
BALD IBIS	1
CEDAR WAXWING	3
ALEXANDRINE PARAKEET	4
FRILL BACK PIGEON	4
BLUE HERON	4
BOBOLINK	4
BORNEAN PHEASANT	4
GAMBELS QUAIL	4
COPPERSMITH BARBET	4
D-ARNAUDS BARBET	4
FLAME TANAGER	5
EURASIAN GOLDEN ORIOLE	5
EURASIAN BULLFINCH	5
CAMPO FLICKER	5
CALIFORNIA QUAIL	5
ASIAN CRESTED IBIS	5
ARARIPE MANAKIN	5
CUBAN TROGON	5

```
GREAT KISKADEE                  5
```

9.1.3 Images of Misclassified Birds

The same Bald Eagle pcitures have been misclassified, however, the birds it was misclassified as seem to have shifted. The model actually did slightly worse on Bald Ibis, but again, has shifted what birds it was considering.

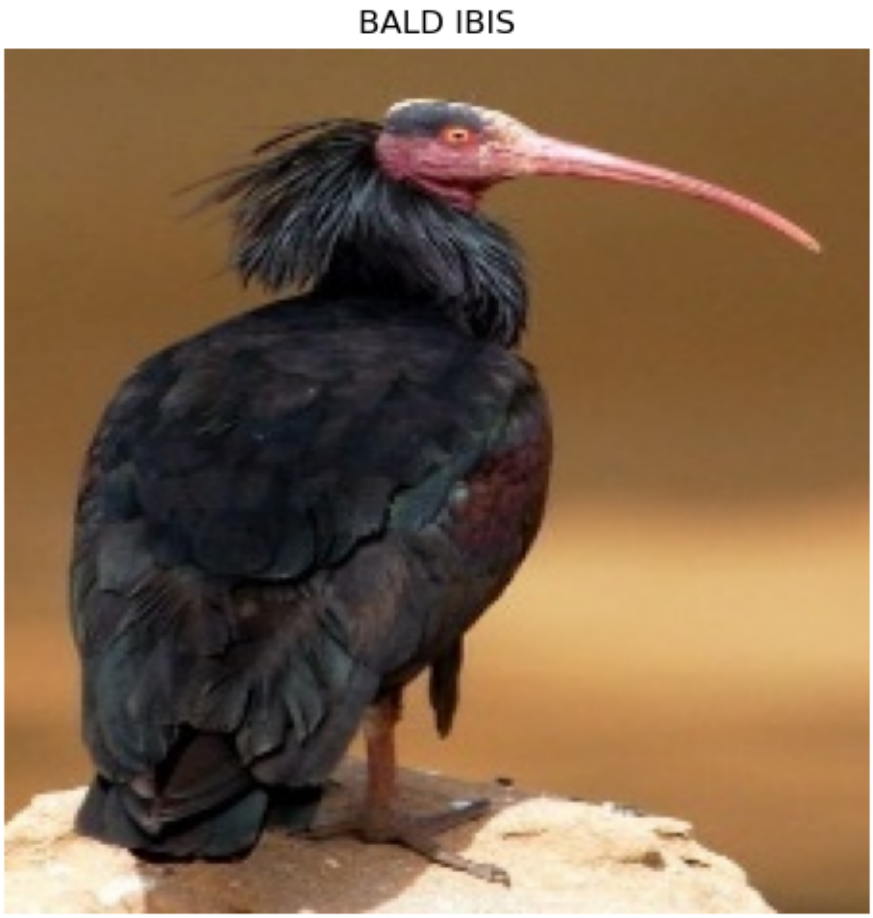
[48]:
```
show_class('BALD EAGLE', preds)

show_class('BALD IBIS', preds)
```

```
Incorrectly labeled BALD EAGLE
```


```
Correctly labeled BALD EAGLE
```

Incorrectly labeled BALD IBIS

Correctly labeled BALD IBIS



9.2 112x112

Now, we'll scale the images up to 112x112, then link that to the 56x56 model, and retrain with the new images.

To avoid overlearning the more detailed pictures I will keep this model simpler and I will also lock the weights on the 56x56 model.

Of note, to make the linking work, each model starts with 2 CNN layers, the current input and the "linking" layer. When connecting the models, the input layer for the original moel will be dropped and the "linking' layer will be the layer that the next model's data is fed into. I'll also need to start introducing padding in the models to ensure each model ends with teh correct size.

[49]:
```
# Resize imags to 112x112

X_train_112, X_val_112 = resize_img(X_train, X_val, (112, 112))

# Compare to original data
fig, axs = plt.subplots(ncols=3, figsize = (15, 15))
```

```
# Compare to original data

axs[0].imshow(X_train_56[5])
axs[0].axis('off')
axs[0].set_title('56x56')

axs[1].imshow(X_train_112[5])
axs[1].axis('off')
axs[1].set_title('112x112')

axs[2].imshow(X_train[5])
axs[2].axis('off')
axs[2].set_title('224x224')

plt.show()
```

[50]:
```
model_setup()

model = Sequential()

model.add(Conv2D(16, 3, padding='same', activation='leaky_relu',␣
↪input_shape=(112, 112, 3)))

model.add(Conv2D(32, 3, padding='same', activation='leaky_relu'))

model.add(MaxPooling2D(2))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 conv2d_4 (Conv2D)           (None, 112, 112, 16)      448

 conv2d_5 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_3 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

=================================================================
Total params: 5088 (19.88 KB)
Trainable params: 5088 (19.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[51]:
```
model_56 = keras.saving.load_model('model_56.keras')

# Check size of model:
print(f'End model has {len(model_56.layers)} layers\n')

# Check input sizes for first two layers
for layer in model_56.layers[:2]:
    print(layer.input)

# Add all layers except for input layer to the new model
for layer in model_56.layers[1:]:
    model.add(layer)

# Lock weights for model_56 layers
for layer in model.layers[-10:]:
    layer.trainable = False

model.summary()
```

```
End model has 11 layers

KerasTensor(type_spec=TensorSpec(shape=(None, 56, 56, 3), dtype=tf.float32,
name='conv2d_input'), name='conv2d_input', description="created by layer
'conv2d_input'")
KerasTensor(type_spec=TensorSpec(shape=(None, 56, 56, 32), dtype=tf.float32,
name=None), name='conv2d/LeakyRelu:0', description="created by layer 'conv2d'")
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 112, 112, 16)      448

 conv2d_5 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_3 (MaxPoolin  (None, 56, 56, 32)        0
```

```
g2D)

conv2d_1 (Conv2D)           (None, 56, 56, 128)       36992

max_pooling2d (MaxPooling2  (None, 28, 28, 128)       0
D)

conv2d_2 (Conv2D)           (None, 26, 26, 64)        73792

max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
g2D)

conv2d_3 (Conv2D)           (None, 11, 11, 32)        18464

max_pooling2d_2 (MaxPoolin  (None, 5, 5, 32)          0
g2D)

dropout (Dropout)           (None, 5, 5, 32)          0

flatten (Flatten)           (None, 800)               0

dense (Dense)               (None, 128)               102528

dense_1 (Dense)             (None, 20)                2580

=================================================================
Total params: 239444 (935.33 KB)
Trainable params: 5088 (19.88 KB)
Non-trainable params: 234356 (915.45 KB)
_________________________________________________________________
```

[52]:
```
# Set callbacks

stopping, checkpoint = set_callbacks('model_112.keras')

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',␣
 ↪metrics='accuracy')

history_112 = model.fit(x = X_train_112/255, y= y_train_vect, batch_size=128,␣
 ↪validation_data= (X_val_112/255, y_val_vect), verbose = 1, epochs = 100,␣
 ↪callbacks=[checkpoint])
```

```
Epoch 1/100
16/16 [==============================] - 6s 376ms/step - loss: 5.7147 -
accuracy: 0.0975 - val_loss: 2.9342 - val_accuracy: 0.1600
Epoch 2/100
16/16 [==============================] - 6s 358ms/step - loss: 2.4164 -
accuracy: 0.3020 - val_loss: 1.9593 - val_accuracy: 0.4500
```

```
Epoch 3/100
16/16 [==============================] - 6s 370ms/step - loss: 1.4529 -
accuracy: 0.5570 - val_loss: 1.2689 - val_accuracy: 0.6100
Epoch 4/100
16/16 [==============================] - 6s 397ms/step - loss: 0.7909 -
accuracy: 0.7830 - val_loss: 0.7977 - val_accuracy: 0.8100
Epoch 5/100
16/16 [==============================] - 6s 394ms/step - loss: 0.3816 -
accuracy: 0.8805 - val_loss: 0.9181 - val_accuracy: 0.7600
Epoch 6/100
16/16 [==============================] - 6s 391ms/step - loss: 0.3239 -
accuracy: 0.9010 - val_loss: 0.8537 - val_accuracy: 0.8200
Epoch 7/100
16/16 [==============================] - 7s 401ms/step - loss: 0.0903 -
accuracy: 0.9720 - val_loss: 0.7386 - val_accuracy: 0.8100
Epoch 8/100
16/16 [==============================] - 6s 375ms/step - loss: 0.1576 -
accuracy: 0.9485 - val_loss: 1.2543 - val_accuracy: 0.7500
Epoch 9/100
16/16 [==============================] - 6s 403ms/step - loss: 0.1581 -
accuracy: 0.9490 - val_loss: 0.7301 - val_accuracy: 0.8100
Epoch 10/100
16/16 [==============================] - 6s 399ms/step - loss: 0.0880 -
accuracy: 0.9685 - val_loss: 1.1752 - val_accuracy: 0.7100
Epoch 11/100
16/16 [==============================] - 6s 362ms/step - loss: 0.1692 -
accuracy: 0.9560 - val_loss: 0.7033 - val_accuracy: 0.8500
Epoch 12/100
16/16 [==============================] - 6s 378ms/step - loss: 0.0601 -
accuracy: 0.9810 - val_loss: 0.6913 - val_accuracy: 0.8300
Epoch 13/100
16/16 [==============================] - 6s 365ms/step - loss: 0.1538 -
accuracy: 0.9535 - val_loss: 0.7629 - val_accuracy: 0.8500
Epoch 14/100
16/16 [==============================] - 6s 370ms/step - loss: 0.0369 -
accuracy: 0.9885 - val_loss: 0.6887 - val_accuracy: 0.8500
Epoch 15/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0666 -
accuracy: 0.9770 - val_loss: 0.9296 - val_accuracy: 0.8100
Epoch 16/100
16/16 [==============================] - 6s 359ms/step - loss: 0.0560 -
accuracy: 0.9855 - val_loss: 0.7425 - val_accuracy: 0.8400
Epoch 17/100
16/16 [==============================] - 7s 408ms/step - loss: 0.0890 -
accuracy: 0.9745 - val_loss: 0.7354 - val_accuracy: 0.8500
Epoch 18/100
16/16 [==============================] - 6s 404ms/step - loss: 0.0232 -
accuracy: 0.9925 - val_loss: 0.7206 - val_accuracy: 0.8500
```

```
Epoch 19/100
16/16 [==============================] - 6s 403ms/step - loss: 0.1220 -
accuracy: 0.9655 - val_loss: 0.7502 - val_accuracy: 0.8700
Epoch 20/100
16/16 [==============================] - 7s 415ms/step - loss: 0.0263 -
accuracy: 0.9915 - val_loss: 0.9847 - val_accuracy: 0.7800
Epoch 21/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0318 -
accuracy: 0.9915 - val_loss: 0.7351 - val_accuracy: 0.8100
Epoch 22/100
16/16 [==============================] - 7s 415ms/step - loss: 0.0438 -
accuracy: 0.9855 - val_loss: 0.9144 - val_accuracy: 0.8200
Epoch 23/100
16/16 [==============================] - 7s 421ms/step - loss: 0.0847 -
accuracy: 0.9790 - val_loss: 0.7420 - val_accuracy: 0.8600
Epoch 24/100
16/16 [==============================] - 7s 419ms/step - loss: 0.0527 -
accuracy: 0.9815 - val_loss: 0.7549 - val_accuracy: 0.8700
Epoch 25/100
16/16 [==============================] - 7s 431ms/step - loss: 0.0247 -
accuracy: 0.9930 - val_loss: 0.9089 - val_accuracy: 0.8200
Epoch 26/100
16/16 [==============================] - 6s 394ms/step - loss: 0.0379 -
accuracy: 0.9890 - val_loss: 1.2677 - val_accuracy: 0.7800
Epoch 27/100
16/16 [==============================] - 6s 374ms/step - loss: 0.0532 -
accuracy: 0.9845 - val_loss: 0.7381 - val_accuracy: 0.8700
Epoch 28/100
16/16 [==============================] - 6s 382ms/step - loss: 0.0246 -
accuracy: 0.9930 - val_loss: 0.7069 - val_accuracy: 0.8500
Epoch 29/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0408 -
accuracy: 0.9875 - val_loss: 0.8909 - val_accuracy: 0.8400
Epoch 30/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0273 -
accuracy: 0.9900 - val_loss: 0.8256 - val_accuracy: 0.8400
Epoch 31/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0322 -
accuracy: 0.9880 - val_loss: 1.0253 - val_accuracy: 0.8100
Epoch 32/100
16/16 [==============================] - 7s 407ms/step - loss: 0.0308 -
accuracy: 0.9925 - val_loss: 0.7972 - val_accuracy: 0.8700
Epoch 33/100
16/16 [==============================] - 6s 404ms/step - loss: 0.0330 -
accuracy: 0.9905 - val_loss: 0.7983 - val_accuracy: 0.8500
Epoch 34/100
16/16 [==============================] - 6s 362ms/step - loss: 0.0345 -
accuracy: 0.9895 - val_loss: 0.8541 - val_accuracy: 0.8600
```

```
Epoch 35/100
16/16 [==============================] - 6s 400ms/step - loss: 0.0243 -
accuracy: 0.9910 - val_loss: 0.6249 - val_accuracy: 0.8800
Epoch 36/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0304 -
accuracy: 0.9895 - val_loss: 0.6499 - val_accuracy: 0.8800
Epoch 37/100
16/16 [==============================] - 7s 411ms/step - loss: 0.0354 -
accuracy: 0.9885 - val_loss: 0.6321 - val_accuracy: 0.9000
Epoch 38/100
16/16 [==============================] - 7s 411ms/step - loss: 0.0194 -
accuracy: 0.9950 - val_loss: 0.7407 - val_accuracy: 0.8400
Epoch 39/100
16/16 [==============================] - 6s 377ms/step - loss: 0.0162 -
accuracy: 0.9955 - val_loss: 0.7784 - val_accuracy: 0.8500
Epoch 40/100
16/16 [==============================] - 6s 391ms/step - loss: 0.0198 -
accuracy: 0.9930 - val_loss: 0.8878 - val_accuracy: 0.8300
Epoch 41/100
16/16 [==============================] - 6s 370ms/step - loss: 0.0323 -
accuracy: 0.9880 - val_loss: 0.7291 - val_accuracy: 0.8300
Epoch 42/100
16/16 [==============================] - 6s 358ms/step - loss: 0.0164 -
accuracy: 0.9970 - val_loss: 0.6561 - val_accuracy: 0.8800
Epoch 43/100
16/16 [==============================] - 6s 382ms/step - loss: 0.0177 -
accuracy: 0.9915 - val_loss: 0.8353 - val_accuracy: 0.8400
Epoch 44/100
16/16 [==============================] - 6s 396ms/step - loss: 0.0303 -
accuracy: 0.9910 - val_loss: 0.7385 - val_accuracy: 0.8700
Epoch 45/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0202 -
accuracy: 0.9940 - val_loss: 0.7862 - val_accuracy: 0.8600
Epoch 46/100
16/16 [==============================] - 6s 380ms/step - loss: 0.0205 -
accuracy: 0.9930 - val_loss: 0.7050 - val_accuracy: 0.8500
Epoch 47/100
16/16 [==============================] - 6s 371ms/step - loss: 0.0167 -
accuracy: 0.9955 - val_loss: 0.6365 - val_accuracy: 0.8400
Epoch 48/100
16/16 [==============================] - 6s 374ms/step - loss: 0.0253 -
accuracy: 0.9900 - val_loss: 0.7171 - val_accuracy: 0.8900
Epoch 49/100
16/16 [==============================] - 6s 404ms/step - loss: 0.0333 -
accuracy: 0.9910 - val_loss: 0.6262 - val_accuracy: 0.8700
Epoch 50/100
16/16 [==============================] - 6s 376ms/step - loss: 0.0168 -
accuracy: 0.9930 - val_loss: 0.7055 - val_accuracy: 0.8800
```

```
Epoch 51/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0181 -
accuracy: 0.9945 - val_loss: 0.8988 - val_accuracy: 0.8300
Epoch 52/100
16/16 [==============================] - 6s 380ms/step - loss: 0.0254 -
accuracy: 0.9915 - val_loss: 0.8322 - val_accuracy: 0.8800
Epoch 53/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0121 -
accuracy: 0.9955 - val_loss: 1.6361 - val_accuracy: 0.7700
Epoch 54/100
16/16 [==============================] - 7s 406ms/step - loss: 0.0477 -
accuracy: 0.9915 - val_loss: 0.7997 - val_accuracy: 0.8500
Epoch 55/100
16/16 [==============================] - 6s 392ms/step - loss: 0.0106 -
accuracy: 0.9965 - val_loss: 0.7126 - val_accuracy: 0.8500
Epoch 56/100
16/16 [==============================] - 7s 424ms/step - loss: 0.0268 -
accuracy: 0.9910 - val_loss: 0.9490 - val_accuracy: 0.8700
Epoch 57/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0060 -
accuracy: 0.9985 - val_loss: 0.8021 - val_accuracy: 0.8700
Epoch 58/100
16/16 [==============================] - 6s 405ms/step - loss: 0.0222 -
accuracy: 0.9950 - val_loss: 0.7769 - val_accuracy: 0.8500
Epoch 59/100
16/16 [==============================] - 7s 411ms/step - loss: 0.0203 -
accuracy: 0.9920 - val_loss: 0.9962 - val_accuracy: 0.8600
Epoch 60/100
16/16 [==============================] - 6s 390ms/step - loss: 0.0137 -
accuracy: 0.9945 - val_loss: 0.7484 - val_accuracy: 0.8700
Epoch 61/100
16/16 [==============================] - 7s 444ms/step - loss: 0.0146 -
accuracy: 0.9950 - val_loss: 0.7449 - val_accuracy: 0.8700
Epoch 62/100
16/16 [==============================] - 6s 385ms/step - loss: 0.0192 -
accuracy: 0.9940 - val_loss: 0.9904 - val_accuracy: 0.8400
Epoch 63/100
16/16 [==============================] - 6s 366ms/step - loss: 0.0185 -
accuracy: 0.9925 - val_loss: 0.6566 - val_accuracy: 0.8600
Epoch 64/100
16/16 [==============================] - 6s 400ms/step - loss: 0.0210 -
accuracy: 0.9920 - val_loss: 0.8935 - val_accuracy: 0.8600
Epoch 65/100
16/16 [==============================] - 6s 384ms/step - loss: 0.0352 -
accuracy: 0.9895 - val_loss: 0.6954 - val_accuracy: 0.8800
Epoch 66/100
16/16 [==============================] - 6s 383ms/step - loss: 0.0175 -
accuracy: 0.9940 - val_loss: 0.7045 - val_accuracy: 0.8600
```

```
Epoch 67/100
16/16 [==============================] - 6s 399ms/step - loss: 0.0113 -
accuracy: 0.9970 - val_loss: 0.9715 - val_accuracy: 0.8700
Epoch 68/100
16/16 [==============================] - 6s 386ms/step - loss: 0.0188 -
accuracy: 0.9950 - val_loss: 0.8638 - val_accuracy: 0.8300
Epoch 69/100
16/16 [==============================] - 6s 405ms/step - loss: 0.0176 -
accuracy: 0.9945 - val_loss: 0.7477 - val_accuracy: 0.8800
Epoch 70/100
16/16 [==============================] - 6s 386ms/step - loss: 0.0099 -
accuracy: 0.9975 - val_loss: 0.6996 - val_accuracy: 0.9000
Epoch 71/100
16/16 [==============================] - 6s 376ms/step - loss: 0.0122 -
accuracy: 0.9960 - val_loss: 0.9165 - val_accuracy: 0.8600
Epoch 72/100
16/16 [==============================] - 6s 385ms/step - loss: 0.0110 -
accuracy: 0.9960 - val_loss: 1.1637 - val_accuracy: 0.7700
Epoch 73/100
16/16 [==============================] - 6s 385ms/step - loss: 0.0228 -
accuracy: 0.9920 - val_loss: 1.0175 - val_accuracy: 0.8000
Epoch 74/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0154 -
accuracy: 0.9950 - val_loss: 0.8724 - val_accuracy: 0.8700
Epoch 75/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0127 -
accuracy: 0.9955 - val_loss: 0.7081 - val_accuracy: 0.9000
Epoch 76/100
16/16 [==============================] - 6s 399ms/step - loss: 0.0102 -
accuracy: 0.9970 - val_loss: 0.8194 - val_accuracy: 0.8600
Epoch 77/100
16/16 [==============================] - 6s 391ms/step - loss: 0.0291 -
accuracy: 0.9915 - val_loss: 0.8668 - val_accuracy: 0.8800
Epoch 78/100
16/16 [==============================] - 6s 387ms/step - loss: 0.0141 -
accuracy: 0.9950 - val_loss: 0.7284 - val_accuracy: 0.8500
Epoch 79/100
16/16 [==============================] - 7s 410ms/step - loss: 0.0109 -
accuracy: 0.9955 - val_loss: 1.1724 - val_accuracy: 0.8300
Epoch 80/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0316 -
accuracy: 0.9915 - val_loss: 0.8092 - val_accuracy: 0.8700
Epoch 81/100
16/16 [==============================] - 6s 406ms/step - loss: 0.0053 -
accuracy: 0.9975 - val_loss: 0.8868 - val_accuracy: 0.8700
Epoch 82/100
16/16 [==============================] - 6s 395ms/step - loss: 0.0221 -
accuracy: 0.9920 - val_loss: 0.7205 - val_accuracy: 0.8600
```

```
Epoch 83/100
16/16 [==============================] - 6s 379ms/step - loss: 0.0060 -
accuracy: 0.9975 - val_loss: 1.0194 - val_accuracy: 0.8700
Epoch 84/100
16/16 [==============================] - 7s 416ms/step - loss: 0.0110 -
accuracy: 0.9960 - val_loss: 0.9122 - val_accuracy: 0.8800
Epoch 85/100
16/16 [==============================] - 6s 400ms/step - loss: 0.0055 -
accuracy: 0.9990 - val_loss: 0.7957 - val_accuracy: 0.8600
Epoch 86/100
16/16 [==============================] - 6s 380ms/step - loss: 0.0266 -
accuracy: 0.9930 - val_loss: 0.8322 - val_accuracy: 0.8500
Epoch 87/100
16/16 [==============================] - 6s 366ms/step - loss: 0.0194 -
accuracy: 0.9930 - val_loss: 0.7897 - val_accuracy: 0.8900
Epoch 88/100
16/16 [==============================] - 6s 384ms/step - loss: 0.0152 -
accuracy: 0.9945 - val_loss: 0.9141 - val_accuracy: 0.8200
Epoch 89/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0221 -
accuracy: 0.9930 - val_loss: 0.7707 - val_accuracy: 0.8600
Epoch 90/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0193 -
accuracy: 0.9950 - val_loss: 0.9370 - val_accuracy: 0.8500
Epoch 91/100
16/16 [==============================] - 7s 423ms/step - loss: 0.0294 -
accuracy: 0.9920 - val_loss: 0.9675 - val_accuracy: 0.8400
Epoch 92/100
16/16 [==============================] - 6s 403ms/step - loss: 0.0246 -
accuracy: 0.9925 - val_loss: 0.7766 - val_accuracy: 0.8900
Epoch 93/100
16/16 [==============================] - 6s 391ms/step - loss: 0.0157 -
accuracy: 0.9960 - val_loss: 0.7358 - val_accuracy: 0.8600
Epoch 94/100
16/16 [==============================] - 6s 385ms/step - loss: 0.0231 -
accuracy: 0.9910 - val_loss: 1.0152 - val_accuracy: 0.8400
Epoch 95/100
16/16 [==============================] - 6s 388ms/step - loss: 0.0151 -
accuracy: 0.9940 - val_loss: 0.8413 - val_accuracy: 0.8800
Epoch 96/100
16/16 [==============================] - 6s 383ms/step - loss: 0.0115 -
accuracy: 0.9960 - val_loss: 0.7504 - val_accuracy: 0.8700
Epoch 97/100
16/16 [==============================] - 6s 401ms/step - loss: 0.0300 -
accuracy: 0.9915 - val_loss: 0.8409 - val_accuracy: 0.8800
Epoch 98/100
16/16 [==============================] - 6s 382ms/step - loss: 0.0091 -
accuracy: 0.9965 - val_loss: 0.7519 - val_accuracy: 0.8700
```

```
Epoch 99/100
16/16 [==============================] - 6s 388ms/step - loss: 0.0083 -
accuracy: 0.9985 - val_loss: 0.7649 - val_accuracy: 0.8700
Epoch 100/100
16/16 [==============================] - 6s 377ms/step - loss: 0.0261 -
accuracy: 0.9930 - val_loss: 0.9930 - val_accuracy: 0.8600
```

[53]:
```
model_112 = keras.saving.load_model('model_112.keras')

model_112.evaluate(X_val_112/255, y_val_vect)
```

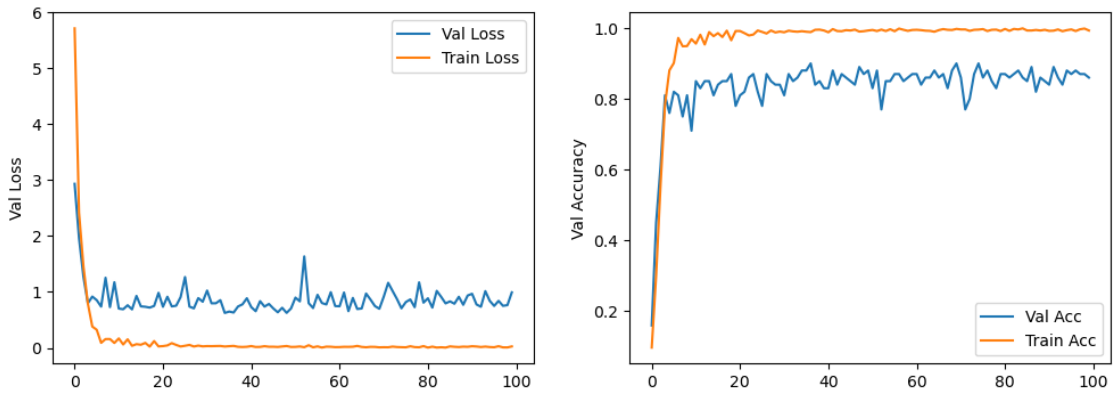
```
4/4 [==============================] - 0s 42ms/step - loss: 0.6321 - accuracy:
0.9000
```

[53]:
```
[0.632105827331543, 0.8999999761581421]
```

9.2.1 Model Evaluation

Scaling up has improved our model once again, both gaining us more accuracy and reducing our loss. However, we are starting to see the overfitting issue spike again, we'd expect this to some extent since the later layers have been trained on very similar data but it could also indicate that even with the resizing, the larger images are introducing too much nose for the model to learn well.

[54]:
```
plot_loss(history_112)
```



9.2.2 Prediction Evaluation

For the first time, our Bald Eagles are no longer the wors classified class. We've also had some shifting of our properly classified classes, indicating the model is learning different types of information with the rescaling process.

[55]:
```
preds = pred_eval(model_112, X= X_val_112)
correct = count_correct(preds)
```

```
correct.sort_values('Correct')
```

```
4/4 [==============================] - 0s 26ms/step
```

[55]:

y_true	Correct
BALD IBIS	1
BORNEAN PHEASANT	1
D-ARNAUDS BARBET	3
BALD EAGLE	3
ALEXANDRINE PARAKEET	4
COPPERSMITH BARBET	4
CEDAR WAXWING	4
GAMBELS QUAIL	4
CAMPO FLICKER	4
BOBOLINK	4
BLUE HERON	4
ASIAN CRESTED IBIS	5
CUBAN TROGON	5
ARARIPE MANAKIN	5
EURASIAN BULLFINCH	5
EURASIAN GOLDEN ORIOLE	5
FLAME TANAGER	5
FRILL BACK PIGEON	5
CALIFORNIA QUAIL	5
GREAT KISKADEE	5

[56]:
```
show_class('BALD IBIS', preds)

show_class('BORNEAN PHEASANT', preds)
```

```
Incorrectly labeled BALD IBIS
```


```
Correctly labeled BALD IBIS
```

BALD IBIS

Incorrectly labeled BORNEAN PHEASANT

Correctly labeled BORNEAN PHEASANT

9.3 224x224

Finally. we'll add on a layer trained on the 224x224 images to see if introducing more information helps our model learn. Again, we'll leave this model very simple to try and avoid overlearning the noise in the images.

[66]:
```
model_setup()

model = Sequential()

model.add(Conv2D(8, 3, padding='same', activation='leaky_relu',␣
↪input_shape=(224, 224, 3)))

model.add(Conv2D(16, 3, padding='same', activation='leaky_relu'))

model.add(MaxPooling2D(2))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 224, 224, 8)       224

 conv2d_9 (Conv2D)           (None, 224, 224, 16)      1168

 max_pooling2d_5 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

=================================================================
Total params: 1392 (5.44 KB)
Trainable params: 1392 (5.44 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[67]:
```python
model_112 = keras.saving.load_model('model_112.keras')

# Check size of model:
print(f'End model has {len(model_112.layers)} layers\n')

# Check input sizes for first two layers
for layer in model_112.layers[:2]:
    print(layer.input)

# Add all layers except for input layer to the new model
for layer in model_112.layers[1:]:
    model.add(layer)

# Lock weights for model_56 layers
for layer in model.layers[3:]:
    layer.trainable = False

model.summary()
```

```
End model has 13 layers

KerasTensor(type_spec=TensorSpec(shape=(None, 112, 112, 3), dtype=tf.float32,
name='conv2d_4_input'), name='conv2d_4_input', description="created by layer
'conv2d_4_input'")
KerasTensor(type_spec=TensorSpec(shape=(None, 112, 112, 16), dtype=tf.float32,
name=None), name='conv2d_4/LeakyRelu:0', description="created by layer
'conv2d_4'")
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d_8 (Conv2D)           (None, 224, 224, 8)       224

 conv2d_9 (Conv2D)           (None, 224, 224, 16)      1168

 max_pooling2d_5 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

 conv2d_5 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_3 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_1 (Conv2D)           (None, 56, 56, 128)       36992

 max_pooling2d (MaxPooling2  (None, 28, 28, 128)       0
 D)

 conv2d_2 (Conv2D)           (None, 26, 26, 64)        73792

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 32)        18464

 max_pooling2d_2 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 dropout (Dropout)           (None, 5, 5, 32)          0

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 128)               102528

 dense_1 (Dense)             (None, 20)                2580

=================================================================
Total params: 240388 (939.02 KB)
Trainable params: 1392 (5.44 KB)
Non-trainable params: 238996 (933.58 KB)
_________________________________________________________________
```

[68]:
```
# Set callbacks

stopping, checkpoint = set_callbacks('model_224.keras')

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',␣
 ↪metrics='accuracy')
```

```
history_224 = model.fit(x = X_train/255, y= y_train_vect, batch_size=128,␣
 ↪validation_data= (X_val/255, y_val_vect), verbose = 1, epochs = 100,␣
 ↪callbacks=[checkpoint])
```

```
Epoch 1/100
16/16 [==============================] - 11s 597ms/step - loss: 4.9157 -
accuracy: 0.1825 - val_loss: 2.7909 - val_accuracy: 0.2900
Epoch 2/100
16/16 [==============================] - 9s 563ms/step - loss: 2.1927 -
accuracy: 0.3550 - val_loss: 1.7798 - val_accuracy: 0.4500
Epoch 3/100
16/16 [==============================] - 9s 573ms/step - loss: 1.5941 -
accuracy: 0.5245 - val_loss: 1.3484 - val_accuracy: 0.6100
Epoch 4/100
16/16 [==============================] - 9s 559ms/step - loss: 1.1951 -
accuracy: 0.6300 - val_loss: 1.0990 - val_accuracy: 0.6300
Epoch 5/100
16/16 [==============================] - 9s 568ms/step - loss: 0.8146 -
accuracy: 0.7405 - val_loss: 1.4542 - val_accuracy: 0.6000
Epoch 6/100
16/16 [==============================] - 9s 578ms/step - loss: 0.7001 -
accuracy: 0.7670 - val_loss: 0.8725 - val_accuracy: 0.7100
Epoch 7/100
16/16 [==============================] - 9s 580ms/step - loss: 0.5386 -
accuracy: 0.8145 - val_loss: 0.8467 - val_accuracy: 0.7500
Epoch 8/100
16/16 [==============================] - 10s 606ms/step - loss: 0.4769 -
accuracy: 0.8415 - val_loss: 0.7524 - val_accuracy: 0.7600
Epoch 9/100
16/16 [==============================] - 10s 606ms/step - loss: 0.3276 -
accuracy: 0.8875 - val_loss: 0.7424 - val_accuracy: 0.7400
Epoch 10/100
16/16 [==============================] - 10s 606ms/step - loss: 0.3065 -
accuracy: 0.8930 - val_loss: 0.9037 - val_accuracy: 0.7500
Epoch 11/100
16/16 [==============================] - 10s 589ms/step - loss: 0.2898 -
accuracy: 0.9105 - val_loss: 0.5883 - val_accuracy: 0.8000
Epoch 12/100
16/16 [==============================] - 10s 592ms/step - loss: 0.2740 -
accuracy: 0.9165 - val_loss: 0.6431 - val_accuracy: 0.8100
Epoch 13/100
16/16 [==============================] - 9s 576ms/step - loss: 0.1977 -
accuracy: 0.9365 - val_loss: 1.0790 - val_accuracy: 0.7000
Epoch 14/100
16/16 [==============================] - 10s 599ms/step - loss: 0.1760 -
accuracy: 0.9415 - val_loss: 0.5730 - val_accuracy: 0.8300
Epoch 15/100
```

```
16/16 [==============================] - 10s 600ms/step - loss: 0.2473 -
accuracy: 0.9245 - val_loss: 0.6327 - val_accuracy: 0.8000
Epoch 16/100
16/16 [==============================] - 10s 640ms/step - loss: 0.0919 -
accuracy: 0.9715 - val_loss: 0.5481 - val_accuracy: 0.8200
Epoch 17/100
16/16 [==============================] - 10s 639ms/step - loss: 0.1823 -
accuracy: 0.9410 - val_loss: 0.5431 - val_accuracy: 0.8100
Epoch 18/100
16/16 [==============================] - 10s 604ms/step - loss: 0.1007 -
accuracy: 0.9680 - val_loss: 0.6738 - val_accuracy: 0.8200
Epoch 19/100
16/16 [==============================] - 10s 600ms/step - loss: 0.1232 -
accuracy: 0.9590 - val_loss: 0.6464 - val_accuracy: 0.8300
Epoch 20/100
16/16 [==============================] - 10s 600ms/step - loss: 0.1281 -
accuracy: 0.9545 - val_loss: 0.5541 - val_accuracy: 0.8200
Epoch 21/100
16/16 [==============================] - 10s 612ms/step - loss: 0.0980 -
accuracy: 0.9740 - val_loss: 0.5986 - val_accuracy: 0.8100
Epoch 22/100
16/16 [==============================] - 10s 617ms/step - loss: 0.1148 -
accuracy: 0.9635 - val_loss: 0.7311 - val_accuracy: 0.8500
Epoch 23/100
16/16 [==============================] - 10s 593ms/step - loss: 0.0879 -
accuracy: 0.9710 - val_loss: 0.6564 - val_accuracy: 0.8400
Epoch 24/100
16/16 [==============================] - 10s 592ms/step - loss: 0.1028 -
accuracy: 0.9680 - val_loss: 0.5555 - val_accuracy: 0.8400
Epoch 25/100
16/16 [==============================] - 9s 586ms/step - loss: 0.0602 -
accuracy: 0.9780 - val_loss: 0.6482 - val_accuracy: 0.8500
Epoch 26/100
16/16 [==============================] - 9s 586ms/step - loss: 0.1031 -
accuracy: 0.9680 - val_loss: 0.5735 - val_accuracy: 0.8600
Epoch 27/100
16/16 [==============================] - 10s 618ms/step - loss: 0.0509 -
accuracy: 0.9840 - val_loss: 0.7026 - val_accuracy: 0.8400
Epoch 28/100
16/16 [==============================] - 10s 594ms/step - loss: 0.0920 -
accuracy: 0.9695 - val_loss: 0.6107 - val_accuracy: 0.8400
Epoch 29/100
16/16 [==============================] - 10s 607ms/step - loss: 0.0504 -
accuracy: 0.9825 - val_loss: 0.6164 - val_accuracy: 0.8300
Epoch 30/100
16/16 [==============================] - 9s 592ms/step - loss: 0.1095 -
accuracy: 0.9645 - val_loss: 0.5841 - val_accuracy: 0.8600
Epoch 31/100
```

```
16/16 [==============================] - 10s 603ms/step - loss: 0.0391 -
accuracy: 0.9875 - val_loss: 0.6505 - val_accuracy: 0.8600
Epoch 32/100
16/16 [==============================] - 10s 626ms/step - loss: 0.0454 -
accuracy: 0.9880 - val_loss: 0.8598 - val_accuracy: 0.8000
Epoch 33/100
16/16 [==============================] - 10s 634ms/step - loss: 0.0855 -
accuracy: 0.9720 - val_loss: 0.6041 - val_accuracy: 0.8500
Epoch 34/100
16/16 [==============================] - 11s 664ms/step - loss: 0.0479 -
accuracy: 0.9850 - val_loss: 0.8758 - val_accuracy: 0.8300
Epoch 35/100
16/16 [==============================] - 10s 627ms/step - loss: 0.0410 -
accuracy: 0.9860 - val_loss: 0.6221 - val_accuracy: 0.8500
Epoch 36/100
16/16 [==============================] - 9s 550ms/step - loss: 0.0745 -
accuracy: 0.9795 - val_loss: 0.5654 - val_accuracy: 0.8800
Epoch 37/100
16/16 [==============================] - 9s 553ms/step - loss: 0.0351 -
accuracy: 0.9900 - val_loss: 0.5410 - val_accuracy: 0.8700
Epoch 38/100
16/16 [==============================] - 9s 548ms/step - loss: 0.0415 -
accuracy: 0.9850 - val_loss: 0.5992 - val_accuracy: 0.8500
Epoch 39/100
16/16 [==============================] - 9s 557ms/step - loss: 0.0379 -
accuracy: 0.9895 - val_loss: 0.5591 - val_accuracy: 0.8800
Epoch 40/100
16/16 [==============================] - 9s 563ms/step - loss: 0.0556 -
accuracy: 0.9820 - val_loss: 0.5807 - val_accuracy: 0.8700
Epoch 41/100
16/16 [==============================] - 9s 561ms/step - loss: 0.0320 -
accuracy: 0.9915 - val_loss: 0.6263 - val_accuracy: 0.8300
Epoch 42/100
16/16 [==============================] - 9s 586ms/step - loss: 0.0696 -
accuracy: 0.9775 - val_loss: 0.6936 - val_accuracy: 0.8200
Epoch 43/100
16/16 [==============================] - 10s 603ms/step - loss: 0.0419 -
accuracy: 0.9855 - val_loss: 0.6095 - val_accuracy: 0.8500
Epoch 44/100
16/16 [==============================] - 10s 596ms/step - loss: 0.0359 -
accuracy: 0.9895 - val_loss: 0.5934 - val_accuracy: 0.8900
Epoch 45/100
16/16 [==============================] - 10s 605ms/step - loss: 0.0329 -
accuracy: 0.9890 - val_loss: 0.7040 - val_accuracy: 0.8100
Epoch 46/100
16/16 [==============================] - 10s 607ms/step - loss: 0.0289 -
accuracy: 0.9905 - val_loss: 0.6349 - val_accuracy: 0.8500
Epoch 47/100
```

```
16/16 [==============================] - 10s 598ms/step - loss: 0.0312 -
accuracy: 0.9900 - val_loss: 0.5949 - val_accuracy: 0.8500
Epoch 48/100
16/16 [==============================] - 11s 687ms/step - loss: 0.0620 -
accuracy: 0.9790 - val_loss: 0.6325 - val_accuracy: 0.8600
Epoch 49/100
16/16 [==============================] - 12s 740ms/step - loss: 0.0448 -
accuracy: 0.9860 - val_loss: 0.6082 - val_accuracy: 0.8900
Epoch 50/100
16/16 [==============================] - 12s 765ms/step - loss: 0.0274 -
accuracy: 0.9920 - val_loss: 0.6815 - val_accuracy: 0.8500
Epoch 51/100
16/16 [==============================] - 12s 718ms/step - loss: 0.0475 -
accuracy: 0.9855 - val_loss: 0.6181 - val_accuracy: 0.8400
Epoch 52/100
16/16 [==============================] - 9s 577ms/step - loss: 0.0578 -
accuracy: 0.9860 - val_loss: 0.7808 - val_accuracy: 0.8600
Epoch 53/100
16/16 [==============================] - 9s 576ms/step - loss: 0.0218 -
accuracy: 0.9950 - val_loss: 0.9008 - val_accuracy: 0.7700
Epoch 54/100
16/16 [==============================] - 9s 586ms/step - loss: 0.0401 -
accuracy: 0.9870 - val_loss: 1.0459 - val_accuracy: 0.7600
Epoch 55/100
16/16 [==============================] - 9s 572ms/step - loss: 0.0329 -
accuracy: 0.9880 - val_loss: 0.6898 - val_accuracy: 0.8800
Epoch 56/100
16/16 [==============================] - 9s 587ms/step - loss: 0.0459 -
accuracy: 0.9845 - val_loss: 0.7176 - val_accuracy: 0.8500
Epoch 57/100
16/16 [==============================] - 10s 598ms/step - loss: 0.0148 -
accuracy: 0.9945 - val_loss: 0.6728 - val_accuracy: 0.8600
Epoch 58/100
16/16 [==============================] - 9s 574ms/step - loss: 0.0501 -
accuracy: 0.9830 - val_loss: 0.6565 - val_accuracy: 0.8700
Epoch 59/100
16/16 [==============================] - 9s 581ms/step - loss: 0.0307 -
accuracy: 0.9940 - val_loss: 0.7187 - val_accuracy: 0.8300
Epoch 60/100
16/16 [==============================] - 10s 604ms/step - loss: 0.0271 -
accuracy: 0.9910 - val_loss: 0.6827 - val_accuracy: 0.8700
Epoch 61/100
16/16 [==============================] - 10s 597ms/step - loss: 0.0351 -
accuracy: 0.9885 - val_loss: 0.7095 - val_accuracy: 0.8700
Epoch 62/100
16/16 [==============================] - 9s 578ms/step - loss: 0.0291 -
accuracy: 0.9900 - val_loss: 0.7383 - val_accuracy: 0.8400
Epoch 63/100
```

```
16/16 [==============================] - 9s 566ms/step - loss: 0.0316 -
accuracy: 0.9880 - val_loss: 0.6511 - val_accuracy: 0.8800
Epoch 64/100
16/16 [==============================] - 9s 554ms/step - loss: 0.0384 -
accuracy: 0.9875 - val_loss: 0.8094 - val_accuracy: 0.8500
Epoch 65/100
16/16 [==============================] - 9s 572ms/step - loss: 0.0456 -
accuracy: 0.9890 - val_loss: 0.6996 - val_accuracy: 0.8800
Epoch 66/100
16/16 [==============================] - 9s 583ms/step - loss: 0.0172 -
accuracy: 0.9955 - val_loss: 0.8385 - val_accuracy: 0.8900
Epoch 67/100
16/16 [==============================] - 9s 564ms/step - loss: 0.0184 -
accuracy: 0.9930 - val_loss: 0.7716 - val_accuracy: 0.8700
Epoch 68/100
16/16 [==============================] - 9s 587ms/step - loss: 0.0382 -
accuracy: 0.9835 - val_loss: 0.6874 - val_accuracy: 0.8700
Epoch 69/100
16/16 [==============================] - 9s 570ms/step - loss: 0.0149 -
accuracy: 0.9945 - val_loss: 0.6958 - val_accuracy: 0.8600
Epoch 70/100
16/16 [==============================] - 9s 574ms/step - loss: 0.0186 -
accuracy: 0.9935 - val_loss: 0.7483 - val_accuracy: 0.8600
Epoch 71/100
16/16 [==============================] - 9s 570ms/step - loss: 0.0437 -
accuracy: 0.9840 - val_loss: 0.7614 - val_accuracy: 0.8700
Epoch 72/100
16/16 [==============================] - 9s 567ms/step - loss: 0.0154 -
accuracy: 0.9945 - val_loss: 1.1656 - val_accuracy: 0.7900
Epoch 73/100
16/16 [==============================] - 9s 573ms/step - loss: 0.0345 -
accuracy: 0.9875 - val_loss: 0.7994 - val_accuracy: 0.8200
Epoch 74/100
16/16 [==============================] - 9s 590ms/step - loss: 0.0215 -
accuracy: 0.9940 - val_loss: 0.7765 - val_accuracy: 0.8600
Epoch 75/100
16/16 [==============================] - 10s 627ms/step - loss: 0.0245 -
accuracy: 0.9915 - val_loss: 1.3169 - val_accuracy: 0.7600
Epoch 76/100
16/16 [==============================] - 9s 583ms/step - loss: 0.0271 -
accuracy: 0.9890 - val_loss: 0.7722 - val_accuracy: 0.8600
Epoch 77/100
16/16 [==============================] - 9s 571ms/step - loss: 0.0237 -
accuracy: 0.9930 - val_loss: 0.7205 - val_accuracy: 0.8400
Epoch 78/100
16/16 [==============================] - 9s 592ms/step - loss: 0.0289 -
accuracy: 0.9905 - val_loss: 0.7612 - val_accuracy: 0.8800
Epoch 79/100
```

```
16/16 [==============================] - 9s 572ms/step - loss: 0.0195 -
accuracy: 0.9945 - val_loss: 0.7805 - val_accuracy: 0.8600
Epoch 80/100
16/16 [==============================] - 9s 577ms/step - loss: 0.0523 -
accuracy: 0.9840 - val_loss: 0.6945 - val_accuracy: 0.8600
Epoch 81/100
16/16 [==============================] - 9s 575ms/step - loss: 0.0142 -
accuracy: 0.9945 - val_loss: 0.7691 - val_accuracy: 0.8600
Epoch 82/100
16/16 [==============================] - 9s 571ms/step - loss: 0.0517 -
accuracy: 0.9865 - val_loss: 0.7178 - val_accuracy: 0.8800
Epoch 83/100
16/16 [==============================] - 9s 587ms/step - loss: 0.0122 -
accuracy: 0.9985 - val_loss: 0.7221 - val_accuracy: 0.8500
Epoch 84/100
16/16 [==============================] - 9s 586ms/step - loss: 0.0152 -
accuracy: 0.9955 - val_loss: 0.7865 - val_accuracy: 0.8600
Epoch 85/100
16/16 [==============================] - 10s 599ms/step - loss: 0.0327 -
accuracy: 0.9890 - val_loss: 0.7798 - val_accuracy: 0.8600
Epoch 86/100
16/16 [==============================] - 9s 582ms/step - loss: 0.0079 -
accuracy: 0.9975 - val_loss: 0.7361 - val_accuracy: 0.8700
Epoch 87/100
16/16 [==============================] - 9s 577ms/step - loss: 0.0696 -
accuracy: 0.9825 - val_loss: 0.7262 - val_accuracy: 0.8700
Epoch 88/100
16/16 [==============================] - 9s 573ms/step - loss: 0.0152 -
accuracy: 0.9935 - val_loss: 0.7560 - val_accuracy: 0.8600
Epoch 89/100
16/16 [==============================] - 9s 580ms/step - loss: 0.0323 -
accuracy: 0.9935 - val_loss: 0.7398 - val_accuracy: 0.8800
Epoch 90/100
16/16 [==============================] - 9s 569ms/step - loss: 0.0299 -
accuracy: 0.9910 - val_loss: 0.6863 - val_accuracy: 0.8600
Epoch 91/100
16/16 [==============================] - 9s 588ms/step - loss: 0.0359 -
accuracy: 0.9915 - val_loss: 0.7762 - val_accuracy: 0.8300
Epoch 92/100
16/16 [==============================] - 9s 583ms/step - loss: 0.0597 -
accuracy: 0.9865 - val_loss: 0.7208 - val_accuracy: 0.8700
Epoch 93/100
16/16 [==============================] - 9s 581ms/step - loss: 0.0203 -
accuracy: 0.9930 - val_loss: 0.6148 - val_accuracy: 0.8800
Epoch 94/100
16/16 [==============================] - 10s 603ms/step - loss: 0.0258 -
accuracy: 0.9915 - val_loss: 0.7224 - val_accuracy: 0.8800
Epoch 95/100
```

```
16/16 [==============================] - 9s 574ms/step - loss: 0.0223 -
accuracy: 0.9935 - val_loss: 0.7985 - val_accuracy: 0.8500
Epoch 96/100
16/16 [==============================] - 9s 558ms/step - loss: 0.0165 -
accuracy: 0.9940 - val_loss: 0.7749 - val_accuracy: 0.8500
Epoch 97/100
16/16 [==============================] - 9s 569ms/step - loss: 0.0273 -
accuracy: 0.9925 - val_loss: 0.7151 - val_accuracy: 0.8600
Epoch 98/100
16/16 [==============================] - 9s 562ms/step - loss: 0.0216 -
accuracy: 0.9930 - val_loss: 0.8444 - val_accuracy: 0.8100
Epoch 99/100
16/16 [==============================] - 10s 607ms/step - loss: 0.0229 -
accuracy: 0.9915 - val_loss: 0.7075 - val_accuracy: 0.8700
Epoch 100/100
16/16 [==============================] - 10s 621ms/step - loss: 0.0247 -
accuracy: 0.9905 - val_loss: 0.8108 - val_accuracy: 0.8700
```

[69]:
```
model_224 = keras.saving.load_model('model_224.keras')

model_224.evaluate(X_val/255, y_val_vect)
```

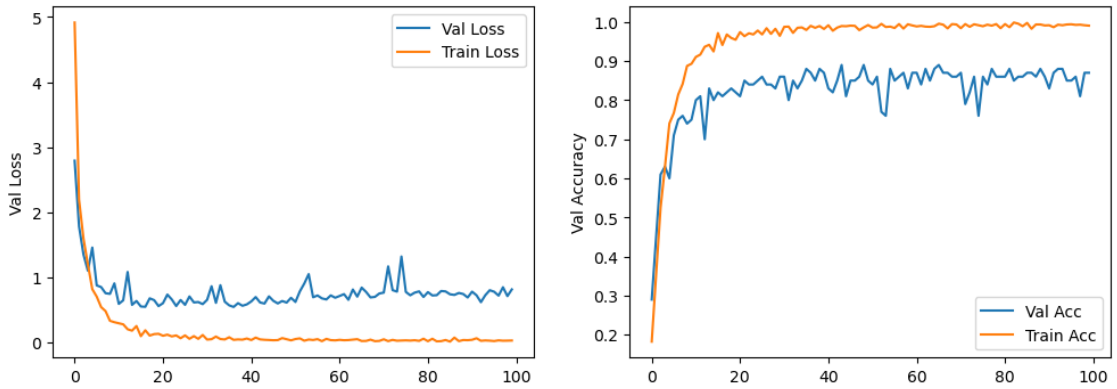
```
4/4 [==============================] - 1s 59ms/step - loss: 0.5934 - accuracy:
0.8900
```

[69]:
```
[0.5933658480644226, 0.8899999856948853]
```

9.3.1 Model Evaluation

Upping the images to 224x224 actually reduced our accuracy somewhat, although loss improved slightly indicating our model is mildly more confident. However, the drop in accuracy does indicate it may actually be better to just scale the images down and train off of those as a standard for this model.

[70]:
```
plot_loss(history_224)
```



9.3.2 Prediction Evaluation

Bald Eagles are back on the bottom of the list for predictions again although it's now primarily confused with the Crested Ibis. Our total number of 100% correctly predicted classes is also now down to only 7, so the model is definitely struggling a bit more.

Based on this, it does seem like the high quality images are making the model struggle a bit more. It appears additional fine tuning of the 224x224 layers of the model could potentially help us narrow down on more specific targets.

[71]:
```
preds = pred_eval(model_224)
correct = count_correct(preds)

correct.sort_values('Correct')
```

```
4/4 [==============================] - 0s 55ms/step
```

[71]:

y_true	Correct
BALD EAGLE	1
BALD IBIS	1
BORNEAN PHEASANT	1
D-ARNAUDS BARBET	2
ALEXANDRINE PARAKEET	3
BLUE HERON	3
CEDAR WAXWING	3
EURASIAN GOLDEN ORIOLE	4
COPPERSMITH BARBET	4
CAMPO FLICKER	4
BOBOLINK	4
ASIAN CRESTED IBIS	4
GAMBELS QUAIL	4
CALIFORNIA QUAIL	5
CUBAN TROGON	5
EURASIAN BULLFINCH	5
ARARIPE MANAKIN	5
FLAME TANAGER	5
FRILL BACK PIGEON	5
GREAT KISKADEE	5

9.3.3 Images of Misclassified Birds

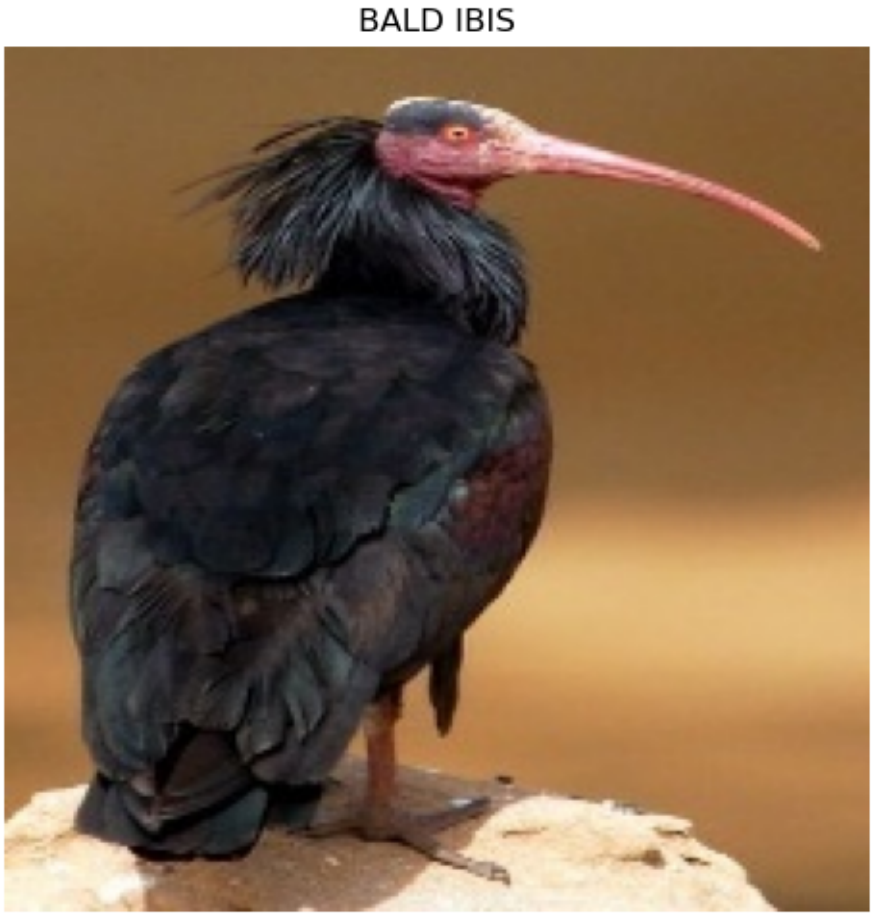
[72]:
```
show_class('BALD EAGLE', preds)

show_class('BALD IBIS', preds)

show_class('BORNEAN PHEASANT', preds)
```

Incorrectly labeled BALD EAGLE

Correctly labeled BALD EAGLE

Incorrectly labeled BALD IBIS

Correctly labeled BALD IBIS



Incorrectly labeled BORNEAN PHEASANT


```
Correctly labeled BORNEAN PHEASANT
```

10 Add more Images

Our model is still struggling with BALD EAGLE, BALD IBIS, and BORNEAN PHEASANT in particular. If we can improve these classes specifically, we can boost the accuracy of our model

again. Since we didn't load the entire dataset previously, we can grab some more images from the original dataset and try to train our last model again.

I want to avoid overbalancing the classes too much, so I'm just going to add 25 images of each one and retrain the 224 model on that to see if I can get it to predict a bit better on those classes.

[73]:
```
new_train_dir = "C:/Users/bcbot/COMP_4531/Final/100-bird-species/train"
# Set the number of images per bird to include
num_img = 50
bird_list = ['BALD EAGLE', 'BALD IBIS', 'BORNEAN PHEASANT']

# Instantiate the containers for holding image and label data
new_train_data = []

bird_num = 0
# Load train data
for i in os.listdir(new_train_dir):
    if i in bird_list:
        count = 0
        sub_directory = os.path.join(new_train_dir, i)
        # Reverse the image order so we're getting different images than the␣
 ↪first time
        for j in os.listdir(sub_directory)[::-1]:
            count+=1
            if count > num_img:
                break
            img = cv2.imread(os.path.join(sub_directory, j))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            new_train_data.append([img, i])
```

[74]:
```
# Add images to training dataset
for item in new_train_data:
    train_data.append(item)

len(train_data)
```

[74]: 2150

[75]:
```
np.random.seed(rs)
np.random.shuffle(train_data)

# Preprocess training data and validation data
lb = LabelBinarizer()

X_train = []
y_train = []
for x, y in train_data:
    X_train.append(x)
```

```
    y_train.append(y)

X_train = np.array(X_train)
y_train = np.array(y_train)

y_train_vect = lb.fit_transform(y_train)
```

[76]:
```
model_setup()

model = Sequential()

model.add(Conv2D(8, 3, padding='same', activation='leaky_relu',␣
 ↪input_shape=(224, 224, 3)))

model.add(Conv2D(16, 3, padding='same', activation='leaky_relu'))

model.add(MaxPooling2D(2))

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 224, 224, 8)       224

 conv2d_11 (Conv2D)          (None, 224, 224, 16)      1168

 max_pooling2d_6 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

=================================================================
Total params: 1392 (5.44 KB)
Trainable params: 1392 (5.44 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[77]:
```
model_112 = keras.saving.load_model('model_112.keras')

# Check size of model:
print(f'End model has {len(model_112.layers)} layers\n')

# Check input sizes for first two layers
for layer in model_112.layers[:2]:
    print(layer.input)

# Add all layers except for input layer to the new model
```

```
for layer in model_112.layers[1:]:
    model.add(layer)

# Lock weights for model_56 layers
for layer in model.layers[3:]:
    layer.trainable = False

model.summary()
```

```
End model has 13 layers

KerasTensor(type_spec=TensorSpec(shape=(None, 112, 112, 3), dtype=tf.float32,
name='conv2d_4_input'), name='conv2d_4_input', description="created by layer
'conv2d_4_input'")
KerasTensor(type_spec=TensorSpec(shape=(None, 112, 112, 16), dtype=tf.float32,
name=None), name='conv2d_4/LeakyRelu:0', description="created by layer
'conv2d_4'")
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 224, 224, 8)       224

 conv2d_11 (Conv2D)          (None, 224, 224, 16)      1168

 max_pooling2d_6 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

 conv2d_5 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_3 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_1 (Conv2D)           (None, 56, 56, 128)       36992

 max_pooling2d (MaxPooling2  (None, 28, 28, 128)       0
 D)

 conv2d_2 (Conv2D)           (None, 26, 26, 64)        73792

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 32)        18464

 max_pooling2d_2 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)
```

```
 dropout (Dropout)           (None, 5, 5, 32)          0

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 128)               102528

 dense_1 (Dense)             (None, 20)                2580

=================================================================
Total params: 240388 (939.02 KB)
Trainable params: 1392 (5.44 KB)
Non-trainable params: 238996 (933.58 KB)
_________________________________________________________________
```

[78]:
```
# Set callbacks

stopping, checkpoint = set_callbacks('model_new_img.keras')

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',␣
 ↪metrics='accuracy')

history_224 = model.fit(x = X_train/255, y= y_train_vect, batch_size=128,␣
 ↪validation_data= (X_val/255, y_val_vect), verbose = 1, epochs = 100,␣
 ↪callbacks=[checkpoint])
```

```
Epoch 1/100
17/17 [==============================] - 11s 615ms/step - loss: 4.6360 -
accuracy: 0.2158 - val_loss: 3.0442 - val_accuracy: 0.1900
Epoch 2/100
17/17 [==============================] - 10s 561ms/step - loss: 2.1423 -
accuracy: 0.3721 - val_loss: 1.8534 - val_accuracy: 0.4500
Epoch 3/100
17/17 [==============================] - 10s 568ms/step - loss: 1.5357 -
accuracy: 0.5307 - val_loss: 1.2384 - val_accuracy: 0.6500
Epoch 4/100
17/17 [==============================] - 10s 602ms/step - loss: 1.1892 -
accuracy: 0.6479 - val_loss: 1.1562 - val_accuracy: 0.6600
Epoch 5/100
17/17 [==============================] - 10s 580ms/step - loss: 0.9545 -
accuracy: 0.6935 - val_loss: 0.9500 - val_accuracy: 0.7200
Epoch 6/100
17/17 [==============================] - 10s 583ms/step - loss: 0.7366 -
accuracy: 0.7726 - val_loss: 0.8700 - val_accuracy: 0.7800
Epoch 7/100
17/17 [==============================] - 10s 586ms/step - loss: 0.6600 -
accuracy: 0.7898 - val_loss: 0.8425 - val_accuracy: 0.7900
Epoch 8/100
17/17 [==============================] - 10s 578ms/step - loss: 0.6096 -
```

```
accuracy: 0.8056 - val_loss: 0.7521 - val_accuracy: 0.7800
Epoch 9/100
17/17 [==============================] - 10s 564ms/step - loss: 0.4813 -
accuracy: 0.8498 - val_loss: 0.7779 - val_accuracy: 0.7500
Epoch 10/100
17/17 [==============================] - 10s 574ms/step - loss: 0.4467 -
accuracy: 0.8647 - val_loss: 1.0322 - val_accuracy: 0.7500
Epoch 11/100
17/17 [==============================] - 10s 585ms/step - loss: 0.3946 -
accuracy: 0.8795 - val_loss: 0.6673 - val_accuracy: 0.7900
Epoch 12/100
17/17 [==============================] - 10s 594ms/step - loss: 0.3052 -
accuracy: 0.9121 - val_loss: 0.6089 - val_accuracy: 0.8100
Epoch 13/100
17/17 [==============================] - 10s 578ms/step - loss: 0.2923 -
accuracy: 0.9116 - val_loss: 0.5740 - val_accuracy: 0.8400
Epoch 14/100
17/17 [==============================] - 10s 582ms/step - loss: 0.2628 -
accuracy: 0.9247 - val_loss: 0.6423 - val_accuracy: 0.7900
Epoch 15/100
17/17 [==============================] - 10s 573ms/step - loss: 0.2713 -
accuracy: 0.9256 - val_loss: 0.5717 - val_accuracy: 0.8200
Epoch 16/100
17/17 [==============================] - 10s 599ms/step - loss: 0.2925 -
accuracy: 0.9191 - val_loss: 0.4987 - val_accuracy: 0.8600
Epoch 17/100
17/17 [==============================] - 10s 572ms/step - loss: 0.2397 -
accuracy: 0.9419 - val_loss: 0.6195 - val_accuracy: 0.8400
Epoch 18/100
17/17 [==============================] - 10s 579ms/step - loss: 0.2044 -
accuracy: 0.9479 - val_loss: 0.6710 - val_accuracy: 0.8000
Epoch 19/100
17/17 [==============================] - 9s 557ms/step - loss: 0.2244 -
accuracy: 0.9400 - val_loss: 0.5328 - val_accuracy: 0.8600
Epoch 20/100
17/17 [==============================] - 10s 585ms/step - loss: 0.2113 -
accuracy: 0.9470 - val_loss: 0.5122 - val_accuracy: 0.8700
Epoch 21/100
17/17 [==============================] - 10s 574ms/step - loss: 0.2289 -
accuracy: 0.9400 - val_loss: 0.6281 - val_accuracy: 0.8300
Epoch 22/100
17/17 [==============================] - 10s 561ms/step - loss: 0.2136 -
accuracy: 0.9433 - val_loss: 0.5165 - val_accuracy: 0.8700
Epoch 23/100
17/17 [==============================] - 10s 570ms/step - loss: 0.2165 -
accuracy: 0.9447 - val_loss: 0.8343 - val_accuracy: 0.8000
Epoch 24/100
17/17 [==============================] - 10s 575ms/step - loss: 0.1987 -
```

```
accuracy: 0.9479 - val_loss: 0.6200 - val_accuracy: 0.8000
Epoch 25/100
17/17 [==============================] - 10s 577ms/step - loss: 0.2156 -
accuracy: 0.9451 - val_loss: 0.5254 - val_accuracy: 0.8800
Epoch 26/100
17/17 [==============================] - 10s 584ms/step - loss: 0.1643 -
accuracy: 0.9628 - val_loss: 0.7151 - val_accuracy: 0.7700
Epoch 27/100
17/17 [==============================] - 10s 586ms/step - loss: 0.1846 -
accuracy: 0.9530 - val_loss: 0.4907 - val_accuracy: 0.8500
Epoch 28/100
17/17 [==============================] - 10s 586ms/step - loss: 0.1685 -
accuracy: 0.9609 - val_loss: 0.4708 - val_accuracy: 0.8700
Epoch 29/100
17/17 [==============================] - 10s 589ms/step - loss: 0.1643 -
accuracy: 0.9535 - val_loss: 0.4922 - val_accuracy: 0.8500
Epoch 30/100
17/17 [==============================] - 10s 612ms/step - loss: 0.1947 -
accuracy: 0.9502 - val_loss: 0.5584 - val_accuracy: 0.8800
Epoch 31/100
17/17 [==============================] - 10s 601ms/step - loss: 0.1620 -
accuracy: 0.9595 - val_loss: 0.4687 - val_accuracy: 0.8400
Epoch 32/100
17/17 [==============================] - 10s 604ms/step - loss: 0.1853 -
accuracy: 0.9526 - val_loss: 0.5256 - val_accuracy: 0.8500
Epoch 33/100
17/17 [==============================] - 10s 602ms/step - loss: 0.1775 -
accuracy: 0.9507 - val_loss: 0.5958 - val_accuracy: 0.8400
Epoch 34/100
17/17 [==============================] - 11s 634ms/step - loss: 0.1810 -
accuracy: 0.9512 - val_loss: 0.4662 - val_accuracy: 0.8900
Epoch 35/100
17/17 [==============================] - 10s 607ms/step - loss: 0.1909 -
accuracy: 0.9521 - val_loss: 0.4950 - val_accuracy: 0.9100
Epoch 36/100
17/17 [==============================] - 11s 639ms/step - loss: 0.1483 -
accuracy: 0.9642 - val_loss: 0.4502 - val_accuracy: 0.8900
Epoch 37/100
17/17 [==============================] - 11s 637ms/step - loss: 0.1657 -
accuracy: 0.9567 - val_loss: 0.4573 - val_accuracy: 0.8700
Epoch 38/100
17/17 [==============================] - 10s 611ms/step - loss: 0.1545 -
accuracy: 0.9605 - val_loss: 0.5150 - val_accuracy: 0.8800
Epoch 39/100
17/17 [==============================] - 11s 618ms/step - loss: 0.1913 -
accuracy: 0.9502 - val_loss: 0.4339 - val_accuracy: 0.9000
Epoch 40/100
17/17 [==============================] - 10s 602ms/step - loss: 0.1584 -
```

```
accuracy: 0.9619 - val_loss: 0.4652 - val_accuracy: 0.8900
Epoch 41/100
17/17 [==============================] - 10s 606ms/step - loss: 0.1630 -
accuracy: 0.9581 - val_loss: 0.5027 - val_accuracy: 0.8700
Epoch 42/100
17/17 [==============================] - 10s 601ms/step - loss: 0.1833 -
accuracy: 0.9530 - val_loss: 0.4683 - val_accuracy: 0.8900
Epoch 43/100
17/17 [==============================] - 10s 612ms/step - loss: 0.1519 -
accuracy: 0.9600 - val_loss: 0.4858 - val_accuracy: 0.8900
Epoch 44/100
17/17 [==============================] - 11s 622ms/step - loss: 0.1584 -
accuracy: 0.9637 - val_loss: 0.4995 - val_accuracy: 0.8500
Epoch 45/100
17/17 [==============================] - 11s 628ms/step - loss: 0.1505 -
accuracy: 0.9595 - val_loss: 0.7208 - val_accuracy: 0.7500
Epoch 46/100
17/17 [==============================] - 10s 607ms/step - loss: 0.1710 -
accuracy: 0.9567 - val_loss: 0.4366 - val_accuracy: 0.9000
Epoch 47/100
17/17 [==============================] - 11s 635ms/step - loss: 0.1551 -
accuracy: 0.9614 - val_loss: 0.4698 - val_accuracy: 0.8900
Epoch 48/100
17/17 [==============================] - 10s 607ms/step - loss: 0.1594 -
accuracy: 0.9586 - val_loss: 0.5619 - val_accuracy: 0.8600
Epoch 49/100
17/17 [==============================] - 10s 599ms/step - loss: 0.1537 -
accuracy: 0.9600 - val_loss: 0.5518 - val_accuracy: 0.8500
Epoch 50/100
17/17 [==============================] - 11s 625ms/step - loss: 0.1492 -
accuracy: 0.9623 - val_loss: 0.4524 - val_accuracy: 0.9000
Epoch 51/100
17/17 [==============================] - 10s 595ms/step - loss: 0.1521 -
accuracy: 0.9581 - val_loss: 0.4952 - val_accuracy: 0.8900
Epoch 52/100
17/17 [==============================] - 10s 614ms/step - loss: 0.1432 -
accuracy: 0.9660 - val_loss: 0.5949 - val_accuracy: 0.8700
Epoch 53/100
17/17 [==============================] - 10s 612ms/step - loss: 0.1742 -
accuracy: 0.9558 - val_loss: 0.4705 - val_accuracy: 0.9000
Epoch 54/100
17/17 [==============================] - 10s 604ms/step - loss: 0.1366 -
accuracy: 0.9665 - val_loss: 0.4362 - val_accuracy: 0.8900
Epoch 55/100
17/17 [==============================] - 12s 686ms/step - loss: 0.1652 -
accuracy: 0.9563 - val_loss: 0.4432 - val_accuracy: 0.9000
Epoch 56/100
17/17 [==============================] - 12s 693ms/step - loss: 0.1325 -
```

```
accuracy: 0.9637 - val_loss: 0.4810 - val_accuracy: 0.8700
Epoch 57/100
17/17 [==============================] - 11s 668ms/step - loss: 0.1519 -
accuracy: 0.9637 - val_loss: 0.5179 - val_accuracy: 0.8300
Epoch 58/100
17/17 [==============================] - 10s 569ms/step - loss: 0.1443 -
accuracy: 0.9647 - val_loss: 0.5614 - val_accuracy: 0.8700
Epoch 59/100
17/17 [==============================] - 10s 591ms/step - loss: 0.1357 -
accuracy: 0.9693 - val_loss: 0.4788 - val_accuracy: 0.8900
Epoch 60/100
17/17 [==============================] - 11s 642ms/step - loss: 0.1546 -
accuracy: 0.9637 - val_loss: 0.4543 - val_accuracy: 0.8700
Epoch 61/100
17/17 [==============================] - 10s 603ms/step - loss: 0.1459 -
accuracy: 0.9577 - val_loss: 0.5326 - val_accuracy: 0.8700
Epoch 62/100
17/17 [==============================] - 10s 614ms/step - loss: 0.1452 -
accuracy: 0.9637 - val_loss: 0.4820 - val_accuracy: 0.8700
Epoch 63/100
17/17 [==============================] - 10s 608ms/step - loss: 0.1258 -
accuracy: 0.9740 - val_loss: 0.4239 - val_accuracy: 0.8900
Epoch 64/100
17/17 [==============================] - 11s 641ms/step - loss: 0.1550 -
accuracy: 0.9609 - val_loss: 0.4533 - val_accuracy: 0.8900
Epoch 65/100
17/17 [==============================] - 11s 622ms/step - loss: 0.1254 -
accuracy: 0.9702 - val_loss: 0.5564 - val_accuracy: 0.8500
Epoch 66/100
17/17 [==============================] - 10s 615ms/step - loss: 0.1339 -
accuracy: 0.9693 - val_loss: 0.4206 - val_accuracy: 0.8900
Epoch 67/100
17/17 [==============================] - 11s 641ms/step - loss: 0.1566 -
accuracy: 0.9558 - val_loss: 0.4644 - val_accuracy: 0.8900
Epoch 68/100
17/17 [==============================] - 11s 622ms/step - loss: 0.1153 -
accuracy: 0.9693 - val_loss: 0.4324 - val_accuracy: 0.8600
Epoch 69/100
17/17 [==============================] - 10s 618ms/step - loss: 0.1390 -
accuracy: 0.9670 - val_loss: 0.5843 - val_accuracy: 0.8800
Epoch 70/100
17/17 [==============================] - 10s 612ms/step - loss: 0.1280 -
accuracy: 0.9679 - val_loss: 0.4756 - val_accuracy: 0.8700
Epoch 71/100
17/17 [==============================] - 11s 667ms/step - loss: 0.1354 -
accuracy: 0.9665 - val_loss: 0.4826 - val_accuracy: 0.8800
Epoch 72/100
17/17 [==============================] - 12s 687ms/step - loss: 0.1259 -
```

```
accuracy: 0.9698 - val_loss: 0.5724 - val_accuracy: 0.8600
Epoch 73/100
17/17 [==============================] - 11s 634ms/step - loss: 0.1339 -
accuracy: 0.9688 - val_loss: 0.5197 - val_accuracy: 0.8700
Epoch 74/100
17/17 [==============================] - 11s 644ms/step - loss: 0.1452 -
accuracy: 0.9586 - val_loss: 0.4334 - val_accuracy: 0.9000
Epoch 75/100
17/17 [==============================] - 11s 653ms/step - loss: 0.1133 -
accuracy: 0.9707 - val_loss: 0.4306 - val_accuracy: 0.9000
Epoch 76/100
17/17 [==============================] - 11s 647ms/step - loss: 0.1463 -
accuracy: 0.9642 - val_loss: 0.4516 - val_accuracy: 0.8900
Epoch 77/100
17/17 [==============================] - 11s 647ms/step - loss: 0.1398 -
accuracy: 0.9679 - val_loss: 0.5215 - val_accuracy: 0.8700
Epoch 78/100
17/17 [==============================] - 11s 627ms/step - loss: 0.1313 -
accuracy: 0.9688 - val_loss: 0.4398 - val_accuracy: 0.9100
Epoch 79/100
17/17 [==============================] - 11s 648ms/step - loss: 0.1451 -
accuracy: 0.9665 - val_loss: 0.4785 - val_accuracy: 0.8900
Epoch 80/100
17/17 [==============================] - 11s 670ms/step - loss: 0.1303 -
accuracy: 0.9684 - val_loss: 0.9660 - val_accuracy: 0.7600
Epoch 81/100
17/17 [==============================] - 12s 717ms/step - loss: 0.1555 -
accuracy: 0.9628 - val_loss: 0.4477 - val_accuracy: 0.8800
Epoch 82/100
17/17 [==============================] - 13s 750ms/step - loss: 0.1290 -
accuracy: 0.9698 - val_loss: 0.4294 - val_accuracy: 0.8800
Epoch 83/100
17/17 [==============================] - 13s 789ms/step - loss: 0.1360 -
accuracy: 0.9637 - val_loss: 0.4119 - val_accuracy: 0.8900
Epoch 84/100
17/17 [==============================] - 11s 628ms/step - loss: 0.1355 -
accuracy: 0.9609 - val_loss: 0.5743 - val_accuracy: 0.8700
Epoch 85/100
17/17 [==============================] - 10s 608ms/step - loss: 0.1335 -
accuracy: 0.9693 - val_loss: 0.5219 - val_accuracy: 0.8700
Epoch 86/100
17/17 [==============================] - 11s 626ms/step - loss: 0.1320 -
accuracy: 0.9660 - val_loss: 0.4591 - val_accuracy: 0.8700
Epoch 87/100
17/17 [==============================] - 11s 647ms/step - loss: 0.1258 -
accuracy: 0.9726 - val_loss: 0.4160 - val_accuracy: 0.8900
Epoch 88/100
17/17 [==============================] - 11s 632ms/step - loss: 0.1284 -
```

```
accuracy: 0.9642 - val_loss: 0.5142 - val_accuracy: 0.8900
Epoch 89/100
17/17 [==============================] - 12s 695ms/step - loss: 0.1244 -
accuracy: 0.9721 - val_loss: 0.4765 - val_accuracy: 0.8700
Epoch 90/100
17/17 [==============================] - 12s 702ms/step - loss: 0.1279 -
accuracy: 0.9698 - val_loss: 0.4018 - val_accuracy: 0.8900
Epoch 91/100
17/17 [==============================] - 12s 735ms/step - loss: 0.1354 -
accuracy: 0.9679 - val_loss: 0.4894 - val_accuracy: 0.8500
Epoch 92/100
17/17 [==============================] - 14s 805ms/step - loss: 0.1241 -
accuracy: 0.9665 - val_loss: 0.4786 - val_accuracy: 0.8900
Epoch 93/100
17/17 [==============================] - 13s 793ms/step - loss: 0.1406 -
accuracy: 0.9656 - val_loss: 0.4677 - val_accuracy: 0.8700
Epoch 94/100
17/17 [==============================] - 10s 614ms/step - loss: 0.1239 -
accuracy: 0.9670 - val_loss: 0.4041 - val_accuracy: 0.9000
Epoch 95/100
17/17 [==============================] - 10s 616ms/step - loss: 0.1426 -
accuracy: 0.9605 - val_loss: 0.4624 - val_accuracy: 0.8900
Epoch 96/100
17/17 [==============================] - 11s 653ms/step - loss: 0.1331 -
accuracy: 0.9642 - val_loss: 0.4572 - val_accuracy: 0.8800
Epoch 97/100
17/17 [==============================] - 12s 718ms/step - loss: 0.1216 -
accuracy: 0.9712 - val_loss: 0.3934 - val_accuracy: 0.8900
Epoch 98/100
17/17 [==============================] - 11s 633ms/step - loss: 0.1305 -
accuracy: 0.9702 - val_loss: 0.5004 - val_accuracy: 0.8700
Epoch 99/100
17/17 [==============================] - 11s 616ms/step - loss: 0.1250 -
accuracy: 0.9702 - val_loss: 0.3824 - val_accuracy: 0.8900
Epoch 100/100
17/17 [==============================] - 11s 618ms/step - loss: 0.1231 -
accuracy: 0.9707 - val_loss: 0.4026 - val_accuracy: 0.8900
```

[79]:
```
mod_new_img = keras.saving.load_model('model_new_img.keras')

mod_new_img.evaluate(X_val/255, y_val_vect)
```

```
4/4 [==============================] - 1s 56ms/step - loss: 0.4950 - accuracy:
0.9100
```

[79]:
```
[0.4949931800365448, 0.9100000262260437]
```

[80]:
```
preds = pred_eval(mod_new_img)
correct = count_correct(preds)

correct.sort_values(by='Correct')
```

```
4/4 [==============================] - 0s 50ms/step
```

[80]:

y_true	Correct
BALD EAGLE	1
BALD IBIS	1
D-ARNAUDS BARBET	2
BORNEAN PHEASANT	2
ALEXANDRINE PARAKEET	3
BLUE HERON	3
CEDAR WAXWING	3
EURASIAN GOLDEN ORIOLE	4
COPPERSMITH BARBET	4
CAMPO FLICKER	4
CALIFORNIA QUAIL	4
BOBOLINK	4
ASIAN CRESTED IBIS	4
GAMBELS QUAIL	4
CUBAN TROGON	5
EURASIAN BULLFINCH	5
ARARIPE MANAKIN	5
FLAME TANAGER	5
FRILL BACK PIGEON	5
GREAT KISKADEE	5

[81]:
```
show_class('BALD EAGLE', preds)

show_class('BALD IBIS', preds)
```

```
Incorrectly labeled BALD EAGLE
```


```
Correctly labeled BALD EAGLE
```

Incorrectly labeled BALD IBIS

Correctly labeled BALD IBIS

BALD IBIS