final_comp4531_bottle_mcmanus

March 13, 2024

Bird Species Image Recognition

Brendon Bottle and Sean McManus

[1]:
```
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")
```

```
WARNING:tensorflow:From c:\Users\bcbot\anaconda3\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

1 Note on Randomness

In order to get reproducible results in this notebook, a random seed is set before each model call to ensure the weights are initalized the same way every time. When run on the same machine in the same environment, results should be identical every time. However, due to differences in computer environments, when this codebook is run on a different computer, the models converge in slightly different locations giving slightly different results. In general, we found that the themes presented in this code were consistent on both of our computers although the accuracy was slightly different and our models predicted species slightly differently.

2 Load Data

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

The inital dataset contained 100-200 images each for 525 species of bird. However, for learning and gaining knowledge on model architecture and design we determined that a smaller dataset that could be trained in a more reasonable amount of time would be more beneficial while still allowing us to explore the particular challenges of this dataset. 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.

[2]:
```
''' Original code for loading and saving data files. Outputs saved in pickle␣
 ↪file and included with submission'''

# # 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'))
```

[2]: `' 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.

[90]:
```
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'))
```

[91]:
```
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']
```

[5]:
```
# 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.
```

3 Explore Data

The random selection of birds includes a wide variety of birds in size, coloration, shapes, 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.

[6]:
```
# 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()
```

4 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 model we'll use (EfficientNet) has a scaling layer added into it already, we won't normalize the data here. Instead we'll do that as needed in the function calls.

[92]:
```
# Shuffle data to randomize batches
```

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

[93]:
```
# 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)
```

5 Function Creation

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

[9]:
```
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/255), 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, X = X_val):

    '''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[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[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()
```

6 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.

[94]:
```
# 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 611ms/step - loss: 0.9192 -
accuracy: 0.8315 - val_loss: 0.2380 - val_accuracy: 0.9600
Epoch 2/5
25/25 [==============================] - 13s 541ms/step - loss: 0.1177 -
accuracy: 0.9835 - val_loss: 0.0883 - val_accuracy: 0.9800
Epoch 3/5
25/25 [==============================] - 13s 534ms/step - loss: 0.0199 -
accuracy: 0.9960 - val_loss: 0.0258 - val_accuracy: 0.9900
Epoch 4/5
25/25 [==============================] - 14s 543ms/step - loss: 0.0130 -
accuracy: 0.9950 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 5/5
25/25 [==============================] - 14s 548ms/step - loss: 0.0029 -
accuracy: 0.9995 - val_loss: 0.0018 - val_accuracy: 1.0000
```

7 Simple Model

With only 5 epochs, we already had an accuracy of 100%, 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, a leaky ReLu activation function, 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.

[11]:
```
# Clears any keras global varaibles such as layer names
keras.backend.clear_session()

# Sets the three random seeds for reproducibility of weight initialization
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)
_________________________________________________________________
```

[12]:
```
# Fit model
stopping, checkpoint = set_callbacks('./saved_models/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)
```

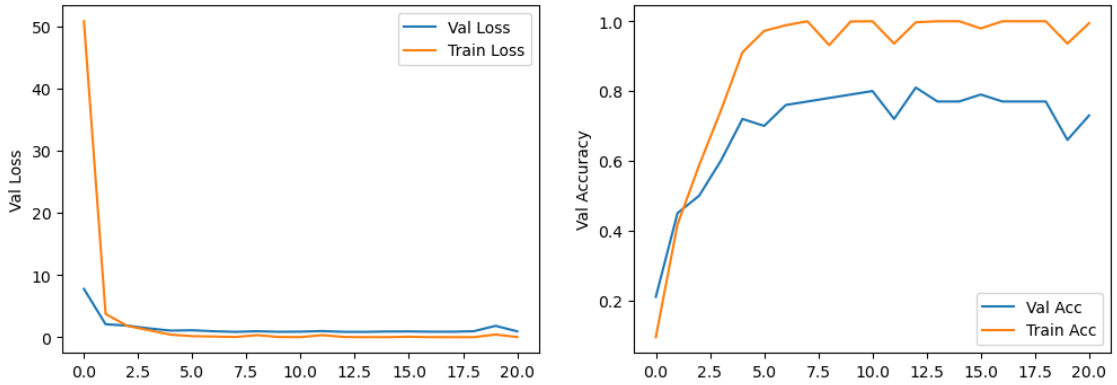
```
4/4 [==============================] - 0s 35ms/step - loss: 0.8571 - accuracy:
0.8100
```

[12]:
```
[0.8571308851242065, 0.8100000023841858]
```

7.1 Model Evaluation

Our model performed suprisingly well for such a simple model. With only 21 epochs we were able to get our model up to 81% accuracy before it plateaued. However, looking at our loss and accuracy over the epochs, we can see that the model is trending towards overfitting. 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.

[13]:
```
# Look at epoch loss/accuracy
plot_loss(simp_history)
```



7.2 Prediction Evaluation

Looking at predictions by class, we can see that while the overall accuracy was good, several species are being very poorly classified including Frill Back Pigeon and Blue Heron.

[14]:
```
simp_preds = pred_eval(simp_model)
simp_correct_count = count_correct(simp_preds)

correct_overall = simp_correct_count

correct_overall.columns = ['Simple']

correct_overall.sort_values('Simple')
```

```
4/4 [==============================] - 0s 35ms/step
```

[14]:

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

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

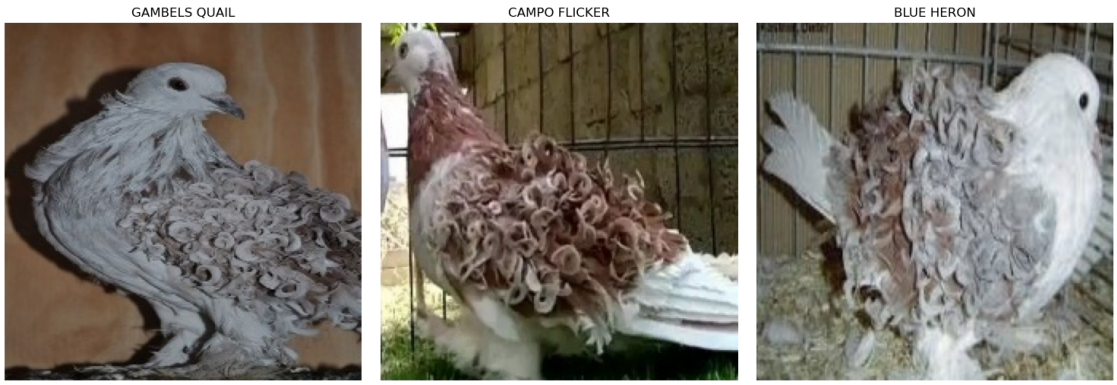
7.3 Images of Incorrect Bird Classifications

Examining the images of those two classes, we can see that the Frill Back Pigeon comes in a variety of colors, and the frill on its back is very different in each picture. The Blue Heron has a wide variety of poses it can take, and depending on the lighting, can have very different coloration but doesn't have any consistent distinctive patterns.

Examining the species that these birds were confused with, it appears our model is making some odd conclusions about these birds that a human would not make based on the images. Comparing these images, it appears that color is the most indicative feature for how a bird will get predicted, although it's often hard to identify exactly how the model matched that.

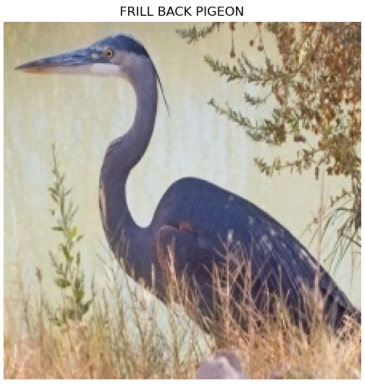
[15]:
```
show_class('FRILL BACK PIGEON', simp_preds)

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

```
Incorrectly labeled FRILL BACK PIGEON
```



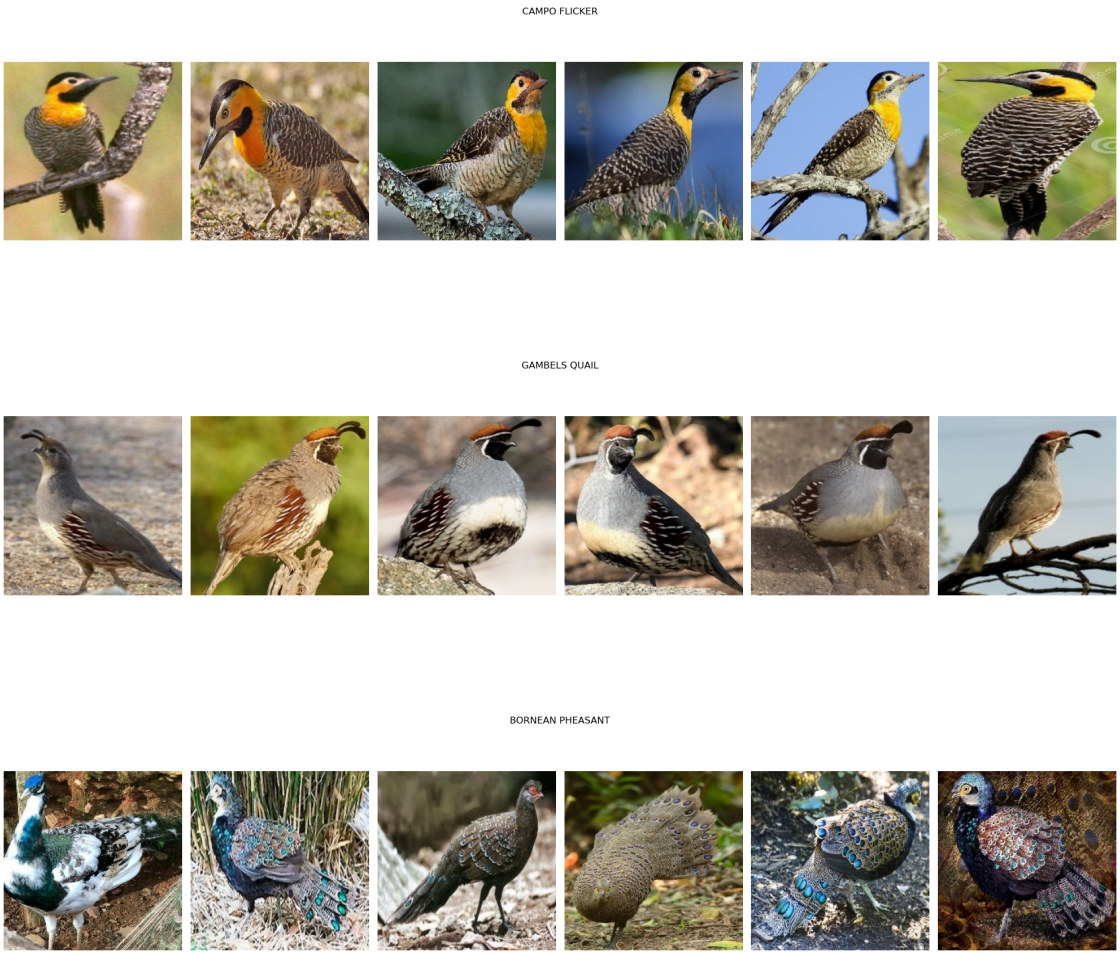

```
Correctly labeled FRILL BACK PIGEON
```

Incorrectly labeled BLUE HERON



Correctly labeled BLUE HERON

[16]:
```
plot_birds(['FRILL BACK PIGEON', 'BLUE HERON', 'CAMPO FLICKER', 'GAMBELS␣
↪QUAIL', 'BORNEAN PHEASANT'], 6)
```



7.4 Images of Correct Bird Classifications

Exploring some of the birds that were well classified, we can see that they contain very distinct features including a very well defined red patch on the Cuban Trogon and a distinctive plume on the Gamble's Quail.

This would indicate that while the model is focused on color, it's also picking out very distinctive shapes from the images.

[17]:
```
# Look at correctly classified birds

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

```
Correctly labeled CUBAN TROGON
```


```
Correctly labeled GAMBELS QUAIL
```

8 Model Complexity Test

After evaluating these classes, it indicated to us that the model was focusing largely on very shallow features, making it an excellent predictor for distinctive birds, and poor classifier for less distinctive or highly varied birds.

To help model learn more features, we increased teh complexity fo the model. We introduced two CNN layers to extract and differentiate more features and included a MaxPooling layer to support model invaraince.

[18]:
```
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(Flatten())

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)
_________________________________________________________________
```

[19]:
```
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)
```

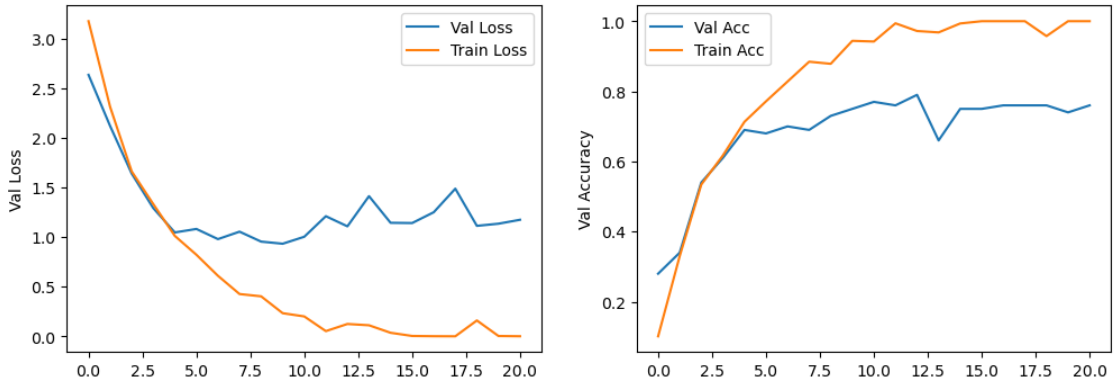
```
4/4 [==============================] - 0s 35ms/step - loss: 1.1084 - accuracy:
0.7900
```

[19]:
```
[1.1083741188049316, 0.7900000214576721]
```

8.1 Model Evaluation

Accuracy did not improve on this version of the model, and our model appeared to be overfitting more when comparing the loss values to the simple model.

[20]:
```
plot_loss(comp_history)
```



8.2 Prediction Evaluation

Exploring the individual class predictions, there were significant improvements, for instance the Frill Back Pigeon was much better classified, but other birds became much worse including the Bald Eagle and Bald Ibis. While the additional layers seemed to have helped the mode learn new information, they also resulted in it unlearning relevant features for other birds.

[21]:
```
comp_preds = pred_eval(comp_model)
comp_correct = count_correct(comp_preds)

correct_overall['Complex'] = comp_correct['Correct']

correct_overall.sort_values('Complex')
```

```
4/4 [==============================] - 0s 23ms/step
```

[21]:

y_true	Simple	Complex
BALD EAGLE	3	1
BALD IBIS	4	2
EURASIAN GOLDEN ORIOLE	4	2

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

8.3 Images of Incorrect Classifications

The Bald Eagle was particularly confusing, as it was very unclear what the model was using to make its predictions, while others made more sense since there were some color or pattern similarities such as the Flame Tanager and Eurasian Golden Oriole

[22]:
```
show_class('BALD EAGLE', comp_preds)

show_class('BALD IBIS', comp_preds)

show_class('EURASIAN GOLDEN ORIOLE', comp_preds)
```

```
Incorrectly labeled BALD EAGLE
```


```
Correctly labeled BALD EAGLE
```

Incorrectly labeled BALD IBIS

Correctly labeled BALD IBIS

Incorrectly labeled EURASIAN GOLDEN ORIOLE

Correctly labeled EURASIAN GOLDEN ORIOLE

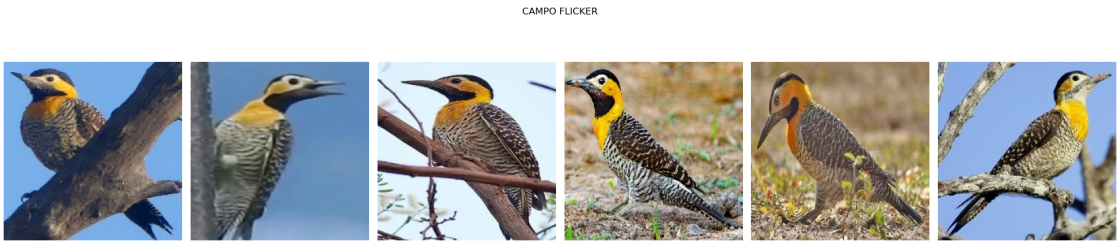
EURASIAN GOLDEN ORIOLE

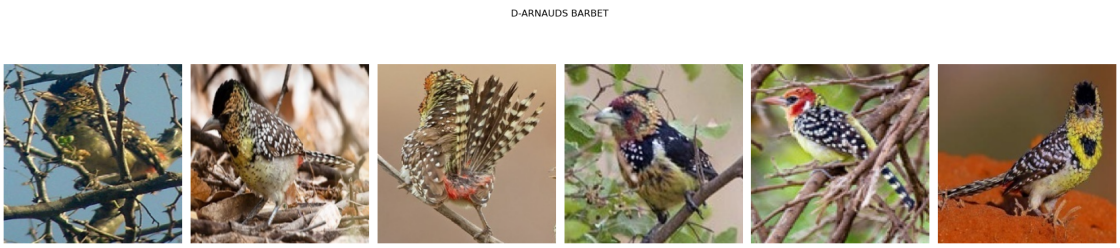
EURASIAN GOLDEN ORIOLE

[70]:
```
plot_birds(['ALEXANDRINE PARAKEET', 'FLAME TANAGER', 'CAMPO FLICKER',␣
 ↪'COPPERSMITH BARBET', 'D-ARNAUDS BARBET'], 6)
```

ALEXANDRINE PARAKEET

FLAME TANAGER





8.4 Images of Correct Classifications

Looking at two of the birds that improved, we were once again seeing very distinctive patterns and colors. The Distinct yellow head on the Campo Flicker and the Green and Red colors on the Barbet seemed like they were likely the reasons for the improved predictions.

[71]:
```
show_class('CAMPO FLICKER', comp_preds)

show_class('COPPERSMITH BARBET', comp_preds)
```

```
Correctly labeled CAMPO FLICKER
```

```
Correctly labeled COPPERSMITH BARBET
```

9 Grayscale Model Testing

To validate our assumption that color was playing a significant role in the predictions, we re-trained our model on grayscale images to see which of the birds were still being well classified.

[24]:
```
grey_train = []
for i,j in train_data:
    img = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)
    grey_train.append([img, j])
print(len(grey_train))

grey_val = []
for i,j in val_data:
    img = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)
    grey_val.append([img, j])
print(len(grey_val))
```

```
2000
100
```

[25]:
```
# prep greyscale data
# Shuffle data to randomize batches
np.random.seed(rs)
np.random.shuffle(grey_train)
np.random.shuffle(grey_val)

X_grey_train = []
y_grey_train = []
for x, y in grey_train:
    X_grey_train.append(x)
    y_grey_train.append(y)

X_grey_train = np.array(X_grey_train)
```

```
y_grey_train = np.array(y_grey_train)

y_grey_train_vect = lb.fit_transform(y_grey_train)

X_grey_val = []
y_grey_val = []
for x, y in grey_val:
    X_grey_val.append(x)
    y_grey_val.append(y)

X_grey_val = np.array(X_grey_val)
y_grey_val = np.array(y_grey_val)

y_grey_val_vect = lb.fit_transform(y_grey_val)
```

[26]:
```
keras.backend.clear_session()
model_setup()

grey_model = Sequential()

grey_model.add(Conv2D(32, 3, activation='leaky_relu', input_shape=(224, 224,␣
↪1)))

grey_model.add(MaxPooling2D(2))

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

grey_model.add(MaxPooling2D(2))

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

grey_model.add(MaxPooling2D(2))

grey_model.add(Flatten())

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

grey_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      320

 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: 451476 (1.72 MB)
Trainable params: 451476 (1.72 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[27]:
```
# Fitting the greyscale model

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

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

grey_history = grey_model.fit(x = X_grey_train/255, y= y_grey_train_vect,␣
 ↪batch_size=128, validation_data= (X_grey_val/255, y_grey_val_vect), verbose␣
 ↪= 0, epochs = 100,callbacks=[stopping, checkpoint])

#print(grey_history.history.keys())
max_val = max(grey_history.history['val_accuracy'])
max_train = max(grey_history.history['accuracy'])

print(f'The maximum traing accuracy was {round(max_train,3)} \nThe maximum␣
 ↪validation accuracy was {round(max_val,3)} ')
```

```
The maximum traing accuracy was 1.0
The maximum validation accuracy was 0.57
```

9.1 Prediction Evaluation

As expected, some of the species with very distinct colors were now being very poorly classified. Other birds that had very distinctive feather shapes such as the California Quail, or very particular patterns to their coloring like the Cuban Trogon, were still being failry well calssified.

[28]:
```
grey_preds = pred_eval(grey_model, X = X_grey_val, y=y_grey_val)
grey_correct = count_correct(grey_preds)

correct_overall['Gray'] = grey_correct['Correct']

correct_overall.sort_values('Gray')
```

```
4/4 [==============================] - 0s 33ms/step
```

[28]:

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

10 Progressive Resizing Model

Clearly color is a very important part of identifying birds, even for human observers, so we couldn't drop color from our model and still expect it to perform well. However, we wanted to see if we could move the model away from focusing so much on the very distinct features, and help it learn more general features that might help it learn thigns about the other species.

We employed progressive resizing, a method to train the model on lower quality images, and progressively adding in new layers trained on higher quality images. The goal is to preven the later layers in the model from being influenced by noise in the higher quality images, or in our case the distinctive patterning the model was learning.

We chose to use 3 levels of resizing: 56x56, 112x112, and 224x224.

10.1 56x56

The base of our model started with the complex model designed in the previous step. We then increased the number of filters to 128, 64, and 32 on the 3 convolutional layers in order to pull out more features from the lower quality images.

We also added an additional fully connected layer ahead of the softmax layer to provide additional feature differentiation on the entirety of the image.

To combat overfitting, we added Dropout layer with a rate of 20% after the last MaxPooling layer. And because we introduced the dropout, we also removed our early stopping parameter and allowed each model to run for 100 epochs.

[29]:
```
# 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()
```

[30]:
```
# 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)
_________________________________________________________________
```

[31]:
```
# 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 371ms/step - loss: 3.0073 -
accuracy: 0.0500 - val_loss: 2.9782 - val_accuracy: 0.0400
Epoch 2/100
16/16 [==============================] - 6s 353ms/step - loss: 2.9645 -
accuracy: 0.1075 - val_loss: 3.0627 - val_accuracy: 0.1000
Epoch 3/100
16/16 [==============================] - 6s 356ms/step - loss: 2.6709 -
accuracy: 0.1970 - val_loss: 2.3822 - val_accuracy: 0.2700
Epoch 4/100
16/16 [==============================] - 6s 366ms/step - loss: 2.4312 -
accuracy: 0.2665 - val_loss: 2.1243 - val_accuracy: 0.3100
Epoch 5/100
16/16 [==============================] - 6s 363ms/step - loss: 2.0374 -
accuracy: 0.3850 - val_loss: 1.6841 - val_accuracy: 0.4800
Epoch 6/100
16/16 [==============================] - 6s 357ms/step - loss: 1.8221 -
accuracy: 0.4540 - val_loss: 1.7722 - val_accuracy: 0.4500
```

```
Epoch 7/100
16/16 [==============================] - 6s 348ms/step - loss: 1.6115 -
accuracy: 0.5155 - val_loss: 1.4120 - val_accuracy: 0.5300
Epoch 8/100
16/16 [==============================] - 5s 315ms/step - loss: 1.4363 -
accuracy: 0.5615 - val_loss: 1.2717 - val_accuracy: 0.5800
Epoch 9/100
16/16 [==============================] - 5s 328ms/step - loss: 1.2937 -
accuracy: 0.6090 - val_loss: 1.0209 - val_accuracy: 0.6200
Epoch 10/100
16/16 [==============================] - 6s 347ms/step - loss: 1.1387 -
accuracy: 0.6580 - val_loss: 1.0289 - val_accuracy: 0.6600
Epoch 11/100
16/16 [==============================] - 5s 340ms/step - loss: 1.0393 -
accuracy: 0.6810 - val_loss: 0.9779 - val_accuracy: 0.6700
Epoch 12/100
16/16 [==============================] - 6s 363ms/step - loss: 0.8897 -
accuracy: 0.7210 - val_loss: 0.9631 - val_accuracy: 0.6900
Epoch 13/100
16/16 [==============================] - 6s 374ms/step - loss: 0.8545 -
accuracy: 0.7380 - val_loss: 0.8576 - val_accuracy: 0.7400
Epoch 14/100
16/16 [==============================] - 5s 319ms/step - loss: 0.7242 -
accuracy: 0.7700 - val_loss: 0.7977 - val_accuracy: 0.7300
Epoch 15/100
16/16 [==============================] - 6s 360ms/step - loss: 0.6338 -
accuracy: 0.8045 - val_loss: 0.7574 - val_accuracy: 0.7200
Epoch 16/100
16/16 [==============================] - 6s 353ms/step - loss: 0.5712 -
accuracy: 0.8190 - val_loss: 0.8890 - val_accuracy: 0.7300
Epoch 17/100
16/16 [==============================] - 5s 341ms/step - loss: 0.5249 -
accuracy: 0.8300 - val_loss: 0.8332 - val_accuracy: 0.7700
Epoch 18/100
16/16 [==============================] - 6s 363ms/step - loss: 0.4698 -
accuracy: 0.8585 - val_loss: 1.0475 - val_accuracy: 0.7000
Epoch 19/100
16/16 [==============================] - 6s 358ms/step - loss: 0.4022 -
accuracy: 0.8735 - val_loss: 0.5625 - val_accuracy: 0.7600
Epoch 20/100
16/16 [==============================] - 5s 341ms/step - loss: 0.4051 -
accuracy: 0.8640 - val_loss: 0.7539 - val_accuracy: 0.7900
Epoch 21/100
16/16 [==============================] - 6s 346ms/step - loss: 0.3025 -
accuracy: 0.9060 - val_loss: 0.6055 - val_accuracy: 0.8400
Epoch 22/100
16/16 [==============================] - 5s 329ms/step - loss: 0.2986 -
accuracy: 0.9060 - val_loss: 0.6363 - val_accuracy: 0.8200
```

```
Epoch 23/100
16/16 [==============================] - 6s 366ms/step - loss: 0.3117 -
accuracy: 0.9000 - val_loss: 0.6322 - val_accuracy: 0.8000
Epoch 24/100
16/16 [==============================] - 6s 345ms/step - loss: 0.2161 -
accuracy: 0.9320 - val_loss: 0.6556 - val_accuracy: 0.8200
Epoch 25/100
16/16 [==============================] - 5s 337ms/step - loss: 0.1859 -
accuracy: 0.9385 - val_loss: 0.7339 - val_accuracy: 0.8200
Epoch 26/100
16/16 [==============================] - 5s 321ms/step - loss: 0.2578 -
accuracy: 0.9155 - val_loss: 0.5532 - val_accuracy: 0.8100
Epoch 27/100
16/16 [==============================] - 6s 360ms/step - loss: 0.1730 -
accuracy: 0.9425 - val_loss: 0.9375 - val_accuracy: 0.7200
Epoch 28/100
16/16 [==============================] - 5s 309ms/step - loss: 0.1168 -
accuracy: 0.9655 - val_loss: 0.5640 - val_accuracy: 0.8400
Epoch 29/100
16/16 [==============================] - 5s 319ms/step - loss: 0.1975 -
accuracy: 0.9400 - val_loss: 0.5186 - val_accuracy: 0.8500
Epoch 30/100
16/16 [==============================] - 6s 355ms/step - loss: 0.0973 -
accuracy: 0.9725 - val_loss: 0.5338 - val_accuracy: 0.8400
Epoch 31/100
16/16 [==============================] - 5s 343ms/step - loss: 0.1288 -
accuracy: 0.9580 - val_loss: 0.6253 - val_accuracy: 0.7900
Epoch 32/100
16/16 [==============================] - 6s 362ms/step - loss: 0.1275 -
accuracy: 0.9590 - val_loss: 0.5819 - val_accuracy: 0.8500
Epoch 33/100
16/16 [==============================] - 6s 360ms/step - loss: 0.1259 -
accuracy: 0.9635 - val_loss: 0.8129 - val_accuracy: 0.8100
Epoch 34/100
16/16 [==============================] - 5s 336ms/step - loss: 0.0964 -
accuracy: 0.9695 - val_loss: 0.8701 - val_accuracy: 0.8000
Epoch 35/100
16/16 [==============================] - 6s 357ms/step - loss: 0.0967 -
accuracy: 0.9685 - val_loss: 0.7481 - val_accuracy: 0.8000
Epoch 36/100
16/16 [==============================] - 6s 344ms/step - loss: 0.1018 -
accuracy: 0.9690 - val_loss: 0.8254 - val_accuracy: 0.7900
Epoch 37/100
16/16 [==============================] - 5s 326ms/step - loss: 0.1399 -
accuracy: 0.9630 - val_loss: 0.8799 - val_accuracy: 0.7600
Epoch 38/100
16/16 [==============================] - 5s 340ms/step - loss: 0.0854 -
accuracy: 0.9775 - val_loss: 0.4658 - val_accuracy: 0.8300
```

```
Epoch 39/100
16/16 [==============================] - 5s 325ms/step - loss: 0.0808 -
accuracy: 0.9775 - val_loss: 0.6500 - val_accuracy: 0.8400
Epoch 40/100
16/16 [==============================] - 5s 311ms/step - loss: 0.1341 -
accuracy: 0.9570 - val_loss: 0.5405 - val_accuracy: 0.8500
Epoch 41/100
16/16 [==============================] - 5s 331ms/step - loss: 0.0440 -
accuracy: 0.9865 - val_loss: 0.6677 - val_accuracy: 0.8400
Epoch 42/100
16/16 [==============================] - 5s 333ms/step - loss: 0.0533 -
accuracy: 0.9850 - val_loss: 0.6049 - val_accuracy: 0.8300
Epoch 43/100
16/16 [==============================] - 5s 330ms/step - loss: 0.0900 -
accuracy: 0.9745 - val_loss: 0.5626 - val_accuracy: 0.8600
Epoch 44/100
16/16 [==============================] - 6s 359ms/step - loss: 0.0583 -
accuracy: 0.9845 - val_loss: 1.0961 - val_accuracy: 0.7700
Epoch 45/100
16/16 [==============================] - 6s 372ms/step - loss: 0.1031 -
accuracy: 0.9685 - val_loss: 0.6527 - val_accuracy: 0.8300
Epoch 46/100
16/16 [==============================] - 6s 367ms/step - loss: 0.0549 -
accuracy: 0.9840 - val_loss: 0.5795 - val_accuracy: 0.8300
Epoch 47/100
16/16 [==============================] - 6s 375ms/step - loss: 0.0628 -
accuracy: 0.9795 - val_loss: 0.5320 - val_accuracy: 0.8600
Epoch 48/100
16/16 [==============================] - 6s 345ms/step - loss: 0.0340 -
accuracy: 0.9880 - val_loss: 0.7774 - val_accuracy: 0.7900
Epoch 49/100
16/16 [==============================] - 6s 370ms/step - loss: 0.0741 -
accuracy: 0.9780 - val_loss: 0.6108 - val_accuracy: 0.8500
Epoch 50/100
16/16 [==============================] - 6s 371ms/step - loss: 0.0643 -
accuracy: 0.9830 - val_loss: 0.5499 - val_accuracy: 0.8200
Epoch 51/100
16/16 [==============================] - 5s 318ms/step - loss: 0.0429 -
accuracy: 0.9840 - val_loss: 0.8869 - val_accuracy: 0.8100
Epoch 52/100
16/16 [==============================] - 5s 328ms/step - loss: 0.0433 -
accuracy: 0.9890 - val_loss: 0.5851 - val_accuracy: 0.8800
Epoch 53/100
16/16 [==============================] - 5s 331ms/step - loss: 0.0394 -
accuracy: 0.9855 - val_loss: 0.5607 - val_accuracy: 0.8500
Epoch 54/100
16/16 [==============================] - 5s 320ms/step - loss: 0.0497 -
accuracy: 0.9810 - val_loss: 0.6130 - val_accuracy: 0.8500
```

```
Epoch 55/100
16/16 [==============================] - 6s 384ms/step - loss: 0.0486 -
accuracy: 0.9830 - val_loss: 0.6125 - val_accuracy: 0.8700
Epoch 56/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0347 -
accuracy: 0.9910 - val_loss: 0.5728 - val_accuracy: 0.8800
Epoch 57/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0819 -
accuracy: 0.9805 - val_loss: 0.6926 - val_accuracy: 0.8200
Epoch 58/100
16/16 [==============================] - 6s 369ms/step - loss: 0.0174 -
accuracy: 0.9950 - val_loss: 0.8175 - val_accuracy: 0.8200
Epoch 59/100
16/16 [==============================] - 6s 400ms/step - loss: 0.0701 -
accuracy: 0.9775 - val_loss: 0.4770 - val_accuracy: 0.8700
Epoch 60/100
16/16 [==============================] - 6s 357ms/step - loss: 0.0334 -
accuracy: 0.9895 - val_loss: 0.5404 - val_accuracy: 0.8800
Epoch 61/100
16/16 [==============================] - 6s 362ms/step - loss: 0.0612 -
accuracy: 0.9810 - val_loss: 0.8738 - val_accuracy: 0.8200
Epoch 62/100
16/16 [==============================] - 6s 368ms/step - loss: 0.0546 -
accuracy: 0.9845 - val_loss: 0.5868 - val_accuracy: 0.8200
Epoch 63/100
16/16 [==============================] - 6s 356ms/step - loss: 0.0175 -
accuracy: 0.9950 - val_loss: 0.5928 - val_accuracy: 0.8300
Epoch 64/100
16/16 [==============================] - 5s 336ms/step - loss: 0.0745 -
accuracy: 0.9795 - val_loss: 0.9999 - val_accuracy: 0.7800
Epoch 65/100
16/16 [==============================] - 5s 345ms/step - loss: 0.0319 -
accuracy: 0.9905 - val_loss: 0.5728 - val_accuracy: 0.8600
Epoch 66/100
16/16 [==============================] - 6s 349ms/step - loss: 0.0410 -
accuracy: 0.9860 - val_loss: 0.7975 - val_accuracy: 0.7900
Epoch 67/100
16/16 [==============================] - 6s 372ms/step - loss: 0.0312 -
accuracy: 0.9905 - val_loss: 0.9044 - val_accuracy: 0.7800
Epoch 68/100
16/16 [==============================] - 6s 350ms/step - loss: 0.0546 -
accuracy: 0.9830 - val_loss: 0.8682 - val_accuracy: 0.8200
Epoch 69/100
16/16 [==============================] - 6s 369ms/step - loss: 0.0196 -
accuracy: 0.9965 - val_loss: 0.7950 - val_accuracy: 0.8200
Epoch 70/100
16/16 [==============================] - 6s 361ms/step - loss: 0.0546 -
accuracy: 0.9850 - val_loss: 0.9876 - val_accuracy: 0.7900
```

```
Epoch 71/100
16/16 [==============================] - 5s 330ms/step - loss: 0.0401 -
accuracy: 0.9870 - val_loss: 0.8698 - val_accuracy: 0.8000
Epoch 72/100
16/16 [==============================] - 6s 366ms/step - loss: 0.0237 -
accuracy: 0.9905 - val_loss: 1.1388 - val_accuracy: 0.7400
Epoch 73/100
16/16 [==============================] - 5s 332ms/step - loss: 0.0456 -
accuracy: 0.9880 - val_loss: 1.1686 - val_accuracy: 0.8000
Epoch 74/100
16/16 [==============================] - 6s 380ms/step - loss: 0.1286 -
accuracy: 0.9685 - val_loss: 0.8205 - val_accuracy: 0.8400
Epoch 75/100
16/16 [==============================] - 6s 377ms/step - loss: 0.0120 -
accuracy: 0.9960 - val_loss: 0.6297 - val_accuracy: 0.8600
Epoch 76/100
16/16 [==============================] - 5s 326ms/step - loss: 0.0244 -
accuracy: 0.9930 - val_loss: 0.6523 - val_accuracy: 0.8400
Epoch 77/100
16/16 [==============================] - 6s 347ms/step - loss: 0.0079 -
accuracy: 0.9985 - val_loss: 0.6554 - val_accuracy: 0.8500
Epoch 78/100
16/16 [==============================] - 6s 372ms/step - loss: 0.0439 -
accuracy: 0.9895 - val_loss: 0.8199 - val_accuracy: 0.8500
Epoch 79/100
16/16 [==============================] - 6s 370ms/step - loss: 0.0216 -
accuracy: 0.9935 - val_loss: 0.7087 - val_accuracy: 0.8300
Epoch 80/100
16/16 [==============================] - 6s 379ms/step - loss: 0.0206 -
accuracy: 0.9950 - val_loss: 0.8979 - val_accuracy: 0.8400
Epoch 81/100
16/16 [==============================] - 6s 373ms/step - loss: 0.0569 -
accuracy: 0.9840 - val_loss: 0.9490 - val_accuracy: 0.8200
Epoch 82/100
16/16 [==============================] - 6s 344ms/step - loss: 0.0300 -
accuracy: 0.9915 - val_loss: 0.8002 - val_accuracy: 0.8300
Epoch 83/100
16/16 [==============================] - 6s 353ms/step - loss: 0.0151 -
accuracy: 0.9955 - val_loss: 1.4066 - val_accuracy: 0.7100
Epoch 84/100
16/16 [==============================] - 6s 350ms/step - loss: 0.0483 -
accuracy: 0.9905 - val_loss: 0.7865 - val_accuracy: 0.8600
Epoch 85/100
16/16 [==============================] - 6s 355ms/step - loss: 0.0344 -
accuracy: 0.9925 - val_loss: 0.9383 - val_accuracy: 0.8200
Epoch 86/100
16/16 [==============================] - 6s 355ms/step - loss: 0.0416 -
accuracy: 0.9885 - val_loss: 0.8063 - val_accuracy: 0.8900
```

```
Epoch 87/100
16/16 [==============================] - 5s 334ms/step - loss: 0.0278 -
accuracy: 0.9905 - val_loss: 0.7575 - val_accuracy: 0.8500
Epoch 88/100
16/16 [==============================] - 5s 343ms/step - loss: 0.0262 -
accuracy: 0.9905 - val_loss: 0.8464 - val_accuracy: 0.8500
Epoch 89/100
16/16 [==============================] - 6s 346ms/step - loss: 0.0237 -
accuracy: 0.9930 - val_loss: 0.5249 - val_accuracy: 0.8900
Epoch 90/100
16/16 [==============================] - 6s 364ms/step - loss: 0.0516 -
accuracy: 0.9860 - val_loss: 0.6892 - val_accuracy: 0.7900
Epoch 91/100
16/16 [==============================] - 5s 323ms/step - loss: 0.0188 -
accuracy: 0.9955 - val_loss: 0.7217 - val_accuracy: 0.8400
Epoch 92/100
16/16 [==============================] - 6s 354ms/step - loss: 0.0247 -
accuracy: 0.9925 - val_loss: 0.9539 - val_accuracy: 0.8800
Epoch 93/100
16/16 [==============================] - 5s 341ms/step - loss: 0.0236 -
accuracy: 0.9940 - val_loss: 1.3058 - val_accuracy: 0.8000
Epoch 94/100
16/16 [==============================] - 5s 337ms/step - loss: 0.0428 -
accuracy: 0.9870 - val_loss: 1.3770 - val_accuracy: 0.7800
Epoch 95/100
16/16 [==============================] - 5s 342ms/step - loss: 0.0153 -
accuracy: 0.9960 - val_loss: 0.8698 - val_accuracy: 0.8300
Epoch 96/100
16/16 [==============================] - 6s 354ms/step - loss: 0.0079 -
accuracy: 0.9985 - val_loss: 1.5114 - val_accuracy: 0.8200
Epoch 97/100
16/16 [==============================] - 6s 351ms/step - loss: 0.0394 -
accuracy: 0.9855 - val_loss: 1.1668 - val_accuracy: 0.7800
Epoch 98/100
16/16 [==============================] - 6s 349ms/step - loss: 0.0063 -
accuracy: 0.9980 - val_loss: 0.8333 - val_accuracy: 0.8400
Epoch 99/100
16/16 [==============================] - 6s 364ms/step - loss: 0.0632 -
accuracy: 0.9795 - val_loss: 0.7232 - val_accuracy: 0.8100
Epoch 100/100
16/16 [==============================] - 6s 360ms/step - loss: 0.0272 -
accuracy: 0.9930 - val_loss: 0.8502 - val_accuracy: 0.8600
```

[32]:
```
model_56 = keras.saving.load_model('model_56.keras')

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

```
4/4 [==============================] - 0s 21ms/step - loss: 0.8063 - accuracy:
```

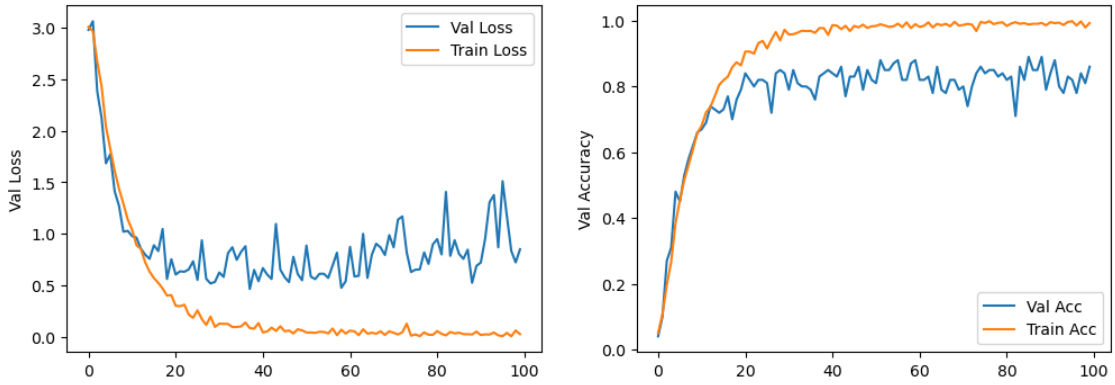
```
0.8900
```

[32]:
```
[0.8062862157821655, 0.8899999856948853]
```

10.1.1 Model Evaluation

Downscaling the images has given us a significant boost in accuracy and improved our loss as well. We've also seen an improvement in overfitting, although our training model is still performing better than our validation.

[33]:
```
plot_loss(history_56)
```



10.1.2 Prediction Evaluations

This new version of the model has provided signficant improvement on our classes as well. Most classes improved, and none got worse.

[34]:
```
preds_56 = pred_eval(model_56, X= X_val_56)
correct_56 = count_correct(preds_56)

correct_overall['56'] = correct_56['Correct']

correct_overall.sort_values('56')
```

```
4/4 [==============================] - 0s 18ms/step
```

[34]:

y_true	Simple	Complex	Gray	56
D-ARNAUDS BARBET	3	3	2	3
CEDAR WAXWING	3	3	3	3
ALEXANDRINE PARAKEET	5	5	1	4
FRILL BACK PIGEON	2	4	3	4
BALD EAGLE	3	1	2	4
BLUE HERON	2	3	3	4
BOBOLINK	4	3	3	4

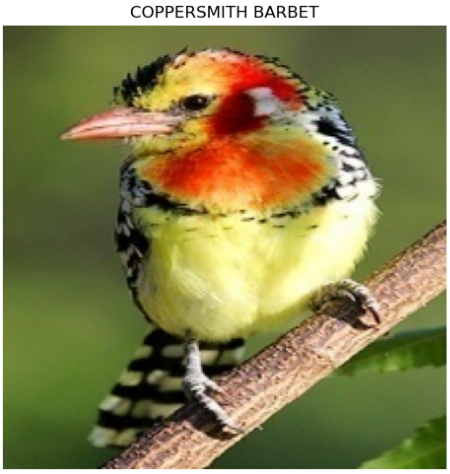
GAMBELS QUAIL	5	5	4	4
COPPERSMITH BARBET	4	5	1	4
FLAME TANAGER	3	4	1	5
EURASIAN GOLDEN ORIOLE	4	2	1	5
EURASIAN BULLFINCH	5	5	4	5
CAMPO FLICKER	4	5	2	5
CALIFORNIA QUAIL	5	5	5	5
BORNEAN PHEASANT	5	5	5	5
BALD IBIS	4	2	4	5
ASIAN CRESTED IBIS	5	4	4	5
ARARIPE MANAKIN	5	5	3	5
CUBAN TROGON	5	5	4	5
GREAT KISKADEE	5	5	2	5

10.1.3 Images of Misclassified Birds

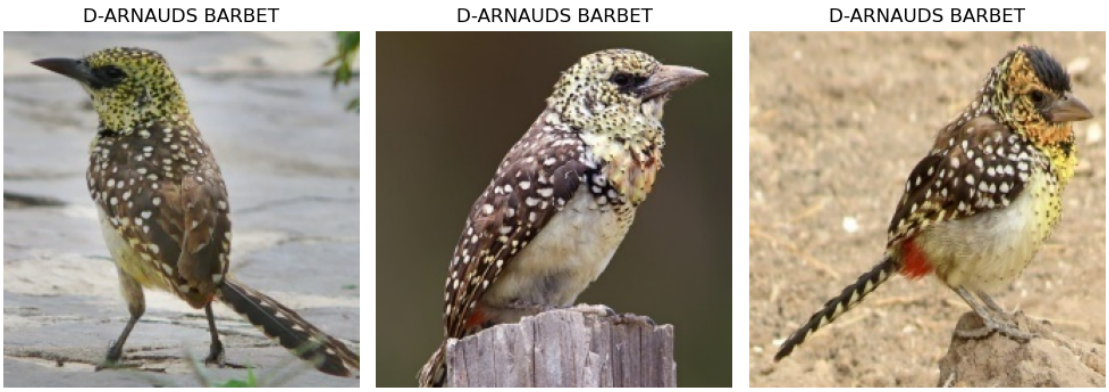
Looking at the two birds that got misclassified the most in this model, the misclassifications do make some sense. For isntance, the D-Arnauds Barbet that was misclassified as a Coppersmith Barbet seems like a mistake a human could make looking at the bird from that angle.

[72]:
```
show_class('D-ARNAUDS BARBET', preds_56)

show_class('CEDAR WAXWING', preds_56)
```

```
Incorrectly labeled D-ARNAUDS BARBET
```




```
Correctly labeled D-ARNAUDS BARBET
```



Incorrectly labeled CEDAR WAXWING

Correctly labeled CEDAR WAXWING

[74]:
```
plot_birds(['COPPERSMITH BARBET'], 5)
```

10.2 112x112

The next step is to add a new layer as the new input that takes 112x112 images, performs a convolution and then uses MaxPooling to downscale the iamges to 56x56 again.

To avoid picking up too much noise, we reduced the filter size on this model.

The weights of the trained 56x56 model are locked, so that they are not influenced by the features that the 112x112 model pulls out.

[36]:
```
# 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()
```

[37]:
```
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)
_________________________________________________________________
```

[38]:
```
model_56 = keras.saving.load_model('model_56.keras')

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

```python
# 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)
_________________________________________________________________
```

[39]:
```
# 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 [==============================] - 7s 399ms/step - loss: 5.7147 -
accuracy: 0.0975 - val_loss: 2.9342 - val_accuracy: 0.1600
Epoch 2/100
16/16 [==============================] - 6s 407ms/step - loss: 2.4164 -
accuracy: 0.3020 - val_loss: 1.9593 - val_accuracy: 0.4500
Epoch 3/100
16/16 [==============================] - 7s 424ms/step - loss: 1.4529 -
accuracy: 0.5570 - val_loss: 1.2689 - val_accuracy: 0.6100
Epoch 4/100
16/16 [==============================] - 6s 401ms/step - loss: 0.7909 -
accuracy: 0.7830 - val_loss: 0.7977 - val_accuracy: 0.8100
Epoch 5/100
16/16 [==============================] - 6s 402ms/step - loss: 0.3816 -
accuracy: 0.8805 - val_loss: 0.9181 - val_accuracy: 0.7600
Epoch 6/100
16/16 [==============================] - 6s 378ms/step - loss: 0.3239 -
accuracy: 0.9010 - val_loss: 0.8537 - val_accuracy: 0.8200
Epoch 7/100
16/16 [==============================] - 6s 393ms/step - loss: 0.0903 -
accuracy: 0.9720 - val_loss: 0.7386 - val_accuracy: 0.8100
Epoch 8/100
16/16 [==============================] - 6s 391ms/step - loss: 0.1576 -
```

```
accuracy: 0.9485 - val_loss: 1.2543 - val_accuracy: 0.7500
Epoch 9/100
16/16 [==============================] - 7s 414ms/step - loss: 0.1581 -
accuracy: 0.9490 - val_loss: 0.7301 - val_accuracy: 0.8100
Epoch 10/100
16/16 [==============================] - 7s 438ms/step - loss: 0.0880 -
accuracy: 0.9685 - val_loss: 1.1752 - val_accuracy: 0.7100
Epoch 11/100
16/16 [==============================] - 6s 388ms/step - loss: 0.1692 -
accuracy: 0.9560 - val_loss: 0.7033 - val_accuracy: 0.8500
Epoch 12/100
16/16 [==============================] - 6s 404ms/step - loss: 0.0601 -
accuracy: 0.9810 - val_loss: 0.6913 - val_accuracy: 0.8300
Epoch 13/100
16/16 [==============================] - 6s 377ms/step - loss: 0.1538 -
accuracy: 0.9535 - val_loss: 0.7629 - val_accuracy: 0.8500
Epoch 14/100
16/16 [==============================] - 7s 415ms/step - loss: 0.0369 -
accuracy: 0.9885 - val_loss: 0.6887 - val_accuracy: 0.8500
Epoch 15/100
16/16 [==============================] - 6s 399ms/step - loss: 0.0666 -
accuracy: 0.9770 - val_loss: 0.9296 - val_accuracy: 0.8100
Epoch 16/100
16/16 [==============================] - 6s 392ms/step - loss: 0.0560 -
accuracy: 0.9855 - val_loss: 0.7425 - val_accuracy: 0.8400
Epoch 17/100
16/16 [==============================] - 7s 418ms/step - loss: 0.0890 -
accuracy: 0.9745 - val_loss: 0.7354 - val_accuracy: 0.8500
Epoch 18/100
16/16 [==============================] - 6s 394ms/step - loss: 0.0232 -
accuracy: 0.9925 - val_loss: 0.7206 - val_accuracy: 0.8500
Epoch 19/100
16/16 [==============================] - 6s 386ms/step - loss: 0.1220 -
accuracy: 0.9655 - val_loss: 0.7502 - val_accuracy: 0.8700
Epoch 20/100
16/16 [==============================] - 6s 403ms/step - loss: 0.0263 -
accuracy: 0.9915 - val_loss: 0.9847 - val_accuracy: 0.7800
Epoch 21/100
16/16 [==============================] - 7s 429ms/step - loss: 0.0318 -
accuracy: 0.9915 - val_loss: 0.7351 - val_accuracy: 0.8100
Epoch 22/100
16/16 [==============================] - 7s 445ms/step - loss: 0.0438 -
accuracy: 0.9855 - val_loss: 0.9144 - val_accuracy: 0.8200
Epoch 23/100
16/16 [==============================] - 7s 416ms/step - loss: 0.0847 -
accuracy: 0.9790 - val_loss: 0.7420 - val_accuracy: 0.8600
Epoch 24/100
16/16 [==============================] - 7s 420ms/step - loss: 0.0527 -
```

```
accuracy: 0.9815 - val_loss: 0.7549 - val_accuracy: 0.8700
Epoch 25/100
16/16 [==============================] - 7s 420ms/step - loss: 0.0247 -
accuracy: 0.9930 - val_loss: 0.9089 - val_accuracy: 0.8200
Epoch 26/100
16/16 [==============================] - 7s 420ms/step - loss: 0.0379 -
accuracy: 0.9890 - val_loss: 1.2677 - val_accuracy: 0.7800
Epoch 27/100
16/16 [==============================] - 7s 417ms/step - loss: 0.0532 -
accuracy: 0.9845 - val_loss: 0.7381 - val_accuracy: 0.8700
Epoch 28/100
16/16 [==============================] - 7s 431ms/step - loss: 0.0246 -
accuracy: 0.9930 - val_loss: 0.7069 - val_accuracy: 0.8500
Epoch 29/100
16/16 [==============================] - 6s 399ms/step - loss: 0.0408 -
accuracy: 0.9875 - val_loss: 0.8909 - val_accuracy: 0.8400
Epoch 30/100
16/16 [==============================] - 6s 373ms/step - loss: 0.0273 -
accuracy: 0.9900 - val_loss: 0.8256 - val_accuracy: 0.8400
Epoch 31/100
16/16 [==============================] - 6s 402ms/step - loss: 0.0322 -
accuracy: 0.9880 - val_loss: 1.0253 - val_accuracy: 0.8100
Epoch 32/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0308 -
accuracy: 0.9925 - val_loss: 0.7972 - val_accuracy: 0.8700
Epoch 33/100
16/16 [==============================] - 6s 386ms/step - loss: 0.0330 -
accuracy: 0.9905 - val_loss: 0.7983 - val_accuracy: 0.8500
Epoch 34/100
16/16 [==============================] - 6s 393ms/step - loss: 0.0345 -
accuracy: 0.9895 - val_loss: 0.8541 - val_accuracy: 0.8600
Epoch 35/100
16/16 [==============================] - 6s 396ms/step - loss: 0.0243 -
accuracy: 0.9910 - val_loss: 0.6249 - val_accuracy: 0.8800
Epoch 36/100
16/16 [==============================] - 6s 402ms/step - loss: 0.0304 -
accuracy: 0.9895 - val_loss: 0.6499 - val_accuracy: 0.8800
Epoch 37/100
16/16 [==============================] - 7s 421ms/step - loss: 0.0354 -
accuracy: 0.9885 - val_loss: 0.6321 - val_accuracy: 0.9000
Epoch 38/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0194 -
accuracy: 0.9950 - val_loss: 0.7407 - val_accuracy: 0.8400
Epoch 39/100
16/16 [==============================] - 7s 423ms/step - loss: 0.0162 -
accuracy: 0.9955 - val_loss: 0.7784 - val_accuracy: 0.8500
Epoch 40/100
16/16 [==============================] - 7s 407ms/step - loss: 0.0198 -
```

```
accuracy: 0.9930 - val_loss: 0.8878 - val_accuracy: 0.8300
Epoch 41/100
16/16 [==============================] - 7s 415ms/step - loss: 0.0323 -
accuracy: 0.9880 - val_loss: 0.7291 - val_accuracy: 0.8300
Epoch 42/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0164 -
accuracy: 0.9970 - val_loss: 0.6561 - val_accuracy: 0.8800
Epoch 43/100
16/16 [==============================] - 6s 370ms/step - loss: 0.0177 -
accuracy: 0.9915 - val_loss: 0.8353 - val_accuracy: 0.8400
Epoch 44/100
16/16 [==============================] - 6s 406ms/step - loss: 0.0303 -
accuracy: 0.9910 - val_loss: 0.7385 - val_accuracy: 0.8700
Epoch 45/100
16/16 [==============================] - 6s 393ms/step - loss: 0.0202 -
accuracy: 0.9940 - val_loss: 0.7862 - val_accuracy: 0.8600
Epoch 46/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0205 -
accuracy: 0.9930 - val_loss: 0.7050 - val_accuracy: 0.8500
Epoch 47/100
16/16 [==============================] - 6s 402ms/step - loss: 0.0167 -
accuracy: 0.9955 - val_loss: 0.6365 - val_accuracy: 0.8400
Epoch 48/100
16/16 [==============================] - 6s 386ms/step - loss: 0.0253 -
accuracy: 0.9900 - val_loss: 0.7171 - val_accuracy: 0.8900
Epoch 49/100
16/16 [==============================] - 7s 420ms/step - loss: 0.0333 -
accuracy: 0.9910 - val_loss: 0.6262 - val_accuracy: 0.8700
Epoch 50/100
16/16 [==============================] - 6s 393ms/step - loss: 0.0168 -
accuracy: 0.9930 - val_loss: 0.7055 - val_accuracy: 0.8800
Epoch 51/100
16/16 [==============================] - 7s 419ms/step - loss: 0.0181 -
accuracy: 0.9945 - val_loss: 0.8988 - val_accuracy: 0.8300
Epoch 52/100
16/16 [==============================] - 7s 421ms/step - loss: 0.0254 -
accuracy: 0.9915 - val_loss: 0.8322 - val_accuracy: 0.8800
Epoch 53/100
16/16 [==============================] - 6s 404ms/step - loss: 0.0121 -
accuracy: 0.9955 - val_loss: 1.6361 - val_accuracy: 0.7700
Epoch 54/100
16/16 [==============================] - 6s 403ms/step - loss: 0.0477 -
accuracy: 0.9915 - val_loss: 0.7997 - val_accuracy: 0.8500
Epoch 55/100
16/16 [==============================] - 6s 381ms/step - loss: 0.0106 -
accuracy: 0.9965 - val_loss: 0.7126 - val_accuracy: 0.8500
Epoch 56/100
16/16 [==============================] - 7s 428ms/step - loss: 0.0268 -
```

```
accuracy: 0.9910 - val_loss: 0.9490 - val_accuracy: 0.8700
Epoch 57/100
16/16 [==============================] - 7s 418ms/step - loss: 0.0060 -
accuracy: 0.9985 - val_loss: 0.8021 - val_accuracy: 0.8700
Epoch 58/100
16/16 [==============================] - 6s 395ms/step - loss: 0.0222 -
accuracy: 0.9950 - val_loss: 0.7769 - val_accuracy: 0.8500
Epoch 59/100
16/16 [==============================] - 6s 404ms/step - loss: 0.0203 -
accuracy: 0.9920 - val_loss: 0.9962 - val_accuracy: 0.8600
Epoch 60/100
16/16 [==============================] - 6s 389ms/step - loss: 0.0137 -
accuracy: 0.9945 - val_loss: 0.7484 - val_accuracy: 0.8700
Epoch 61/100
16/16 [==============================] - 6s 388ms/step - loss: 0.0146 -
accuracy: 0.9950 - val_loss: 0.7449 - val_accuracy: 0.8700
Epoch 62/100
16/16 [==============================] - 6s 407ms/step - loss: 0.0192 -
accuracy: 0.9940 - val_loss: 0.9904 - val_accuracy: 0.8400
Epoch 63/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0185 -
accuracy: 0.9925 - val_loss: 0.6566 - val_accuracy: 0.8600
Epoch 64/100
16/16 [==============================] - 7s 419ms/step - loss: 0.0210 -
accuracy: 0.9920 - val_loss: 0.8935 - val_accuracy: 0.8600
Epoch 65/100
16/16 [==============================] - 6s 364ms/step - loss: 0.0352 -
accuracy: 0.9895 - val_loss: 0.6954 - val_accuracy: 0.8800
Epoch 66/100
16/16 [==============================] - 6s 376ms/step - loss: 0.0175 -
accuracy: 0.9940 - val_loss: 0.7045 - val_accuracy: 0.8600
Epoch 67/100
16/16 [==============================] - 6s 371ms/step - loss: 0.0113 -
accuracy: 0.9970 - val_loss: 0.9715 - val_accuracy: 0.8700
Epoch 68/100
16/16 [==============================] - 6s 382ms/step - loss: 0.0188 -
accuracy: 0.9950 - val_loss: 0.8638 - val_accuracy: 0.8300
Epoch 69/100
16/16 [==============================] - 6s 399ms/step - loss: 0.0176 -
accuracy: 0.9945 - val_loss: 0.7477 - val_accuracy: 0.8800
Epoch 70/100
16/16 [==============================] - 6s 395ms/step - loss: 0.0099 -
accuracy: 0.9975 - val_loss: 0.6996 - val_accuracy: 0.9000
Epoch 71/100
16/16 [==============================] - 6s 378ms/step - loss: 0.0122 -
accuracy: 0.9960 - val_loss: 0.9165 - val_accuracy: 0.8600
Epoch 72/100
16/16 [==============================] - 7s 425ms/step - loss: 0.0110 -
```

```
accuracy: 0.9960 - val_loss: 1.1637 - val_accuracy: 0.7700
Epoch 73/100
16/16 [==============================] - 6s 376ms/step - loss: 0.0228 -
accuracy: 0.9920 - val_loss: 1.0175 - val_accuracy: 0.8000
Epoch 74/100
16/16 [==============================] - 6s 395ms/step - loss: 0.0154 -
accuracy: 0.9950 - val_loss: 0.8724 - val_accuracy: 0.8700
Epoch 75/100
16/16 [==============================] - 6s 392ms/step - loss: 0.0127 -
accuracy: 0.9955 - val_loss: 0.7081 - val_accuracy: 0.9000
Epoch 76/100
16/16 [==============================] - 6s 377ms/step - loss: 0.0102 -
accuracy: 0.9970 - val_loss: 0.8194 - val_accuracy: 0.8600
Epoch 77/100
16/16 [==============================] - 6s 398ms/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 [==============================] - 6s 376ms/step - loss: 0.0109 -
accuracy: 0.9955 - val_loss: 1.1724 - val_accuracy: 0.8300
Epoch 80/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0316 -
accuracy: 0.9915 - val_loss: 0.8092 - val_accuracy: 0.8700
Epoch 81/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0053 -
accuracy: 0.9975 - val_loss: 0.8868 - val_accuracy: 0.8700
Epoch 82/100
16/16 [==============================] - 7s 412ms/step - loss: 0.0221 -
accuracy: 0.9920 - val_loss: 0.7205 - val_accuracy: 0.8600
Epoch 83/100
16/16 [==============================] - 6s 380ms/step - loss: 0.0060 -
accuracy: 0.9975 - val_loss: 1.0194 - val_accuracy: 0.8700
Epoch 84/100
16/16 [==============================] - 6s 374ms/step - loss: 0.0110 -
accuracy: 0.9960 - val_loss: 0.9122 - val_accuracy: 0.8800
Epoch 85/100
16/16 [==============================] - 6s 369ms/step - loss: 0.0055 -
accuracy: 0.9990 - val_loss: 0.7957 - val_accuracy: 0.8600
Epoch 86/100
16/16 [==============================] - 6s 358ms/step - loss: 0.0266 -
accuracy: 0.9930 - val_loss: 0.8322 - val_accuracy: 0.8500
Epoch 87/100
16/16 [==============================] - 6s 364ms/step - loss: 0.0194 -
accuracy: 0.9930 - val_loss: 0.7897 - val_accuracy: 0.8900
Epoch 88/100
16/16 [==============================] - 6s 357ms/step - loss: 0.0152 -
```

```
accuracy: 0.9945 - val_loss: 0.9141 - val_accuracy: 0.8200
Epoch 89/100
16/16 [==============================] - 6s 380ms/step - loss: 0.0221 -
accuracy: 0.9930 - val_loss: 0.7707 - val_accuracy: 0.8600
Epoch 90/100
16/16 [==============================] - 7s 411ms/step - loss: 0.0193 -
accuracy: 0.9950 - val_loss: 0.9370 - val_accuracy: 0.8500
Epoch 91/100
16/16 [==============================] - 6s 402ms/step - loss: 0.0294 -
accuracy: 0.9920 - val_loss: 0.9675 - val_accuracy: 0.8400
Epoch 92/100
16/16 [==============================] - 7s 415ms/step - loss: 0.0246 -
accuracy: 0.9925 - val_loss: 0.7766 - val_accuracy: 0.8900
Epoch 93/100
16/16 [==============================] - 6s 377ms/step - loss: 0.0157 -
accuracy: 0.9960 - val_loss: 0.7358 - val_accuracy: 0.8600
Epoch 94/100
16/16 [==============================] - 6s 397ms/step - loss: 0.0231 -
accuracy: 0.9910 - val_loss: 1.0152 - val_accuracy: 0.8400
Epoch 95/100
16/16 [==============================] - 7s 426ms/step - loss: 0.0151 -
accuracy: 0.9940 - val_loss: 0.8413 - val_accuracy: 0.8800
Epoch 96/100
16/16 [==============================] - 6s 379ms/step - loss: 0.0115 -
accuracy: 0.9960 - val_loss: 0.7504 - val_accuracy: 0.8700
Epoch 97/100
16/16 [==============================] - 6s 398ms/step - loss: 0.0300 -
accuracy: 0.9915 - val_loss: 0.8409 - val_accuracy: 0.8800
Epoch 98/100
16/16 [==============================] - 6s 372ms/step - loss: 0.0091 -
accuracy: 0.9965 - val_loss: 0.7519 - val_accuracy: 0.8700
Epoch 99/100
16/16 [==============================] - 6s 353ms/step - loss: 0.0083 -
accuracy: 0.9985 - val_loss: 0.7649 - val_accuracy: 0.8700
Epoch 100/100
16/16 [==============================] - 6s 368ms/step - loss: 0.0261 -
accuracy: 0.9930 - val_loss: 0.9930 - val_accuracy: 0.8600
```

[40]:
```
model_112 = keras.saving.load_model('model_112.keras')

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

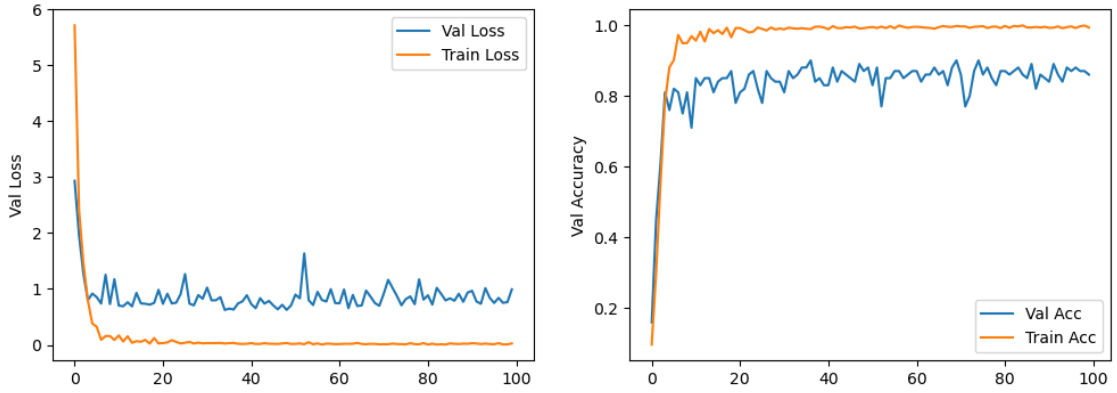
```
4/4 [==============================] - 0s 24ms/step - loss: 0.6321 - accuracy:
0.9000
```

[40]:
```
[0.632105827331543, 0.8999999761581421]
```

10.2.1 Model Evaluation

This new layer improved the model again, increasing both accuracy and decreasing loss. It also appears that this new version is less prone to overfitting since it's only updating the weights for a single layer.

[41]:
```
plot_loss(history_112)
```



10.2.2 Prediction Evaluation

There is one calss that had a decrease in correct predictions, Campo Flicker, but in all other cases teh predictions either stayed the same or improved.

[42]:
```
preds_112 = pred_eval(model_112, X= X_val_112)
correct_112 = count_correct(preds_112)

correct_overall['112'] = correct_112['Correct']

correct_overall.sort_values('112')
```

```
4/4 [==============================] - 0s 23ms/step
```

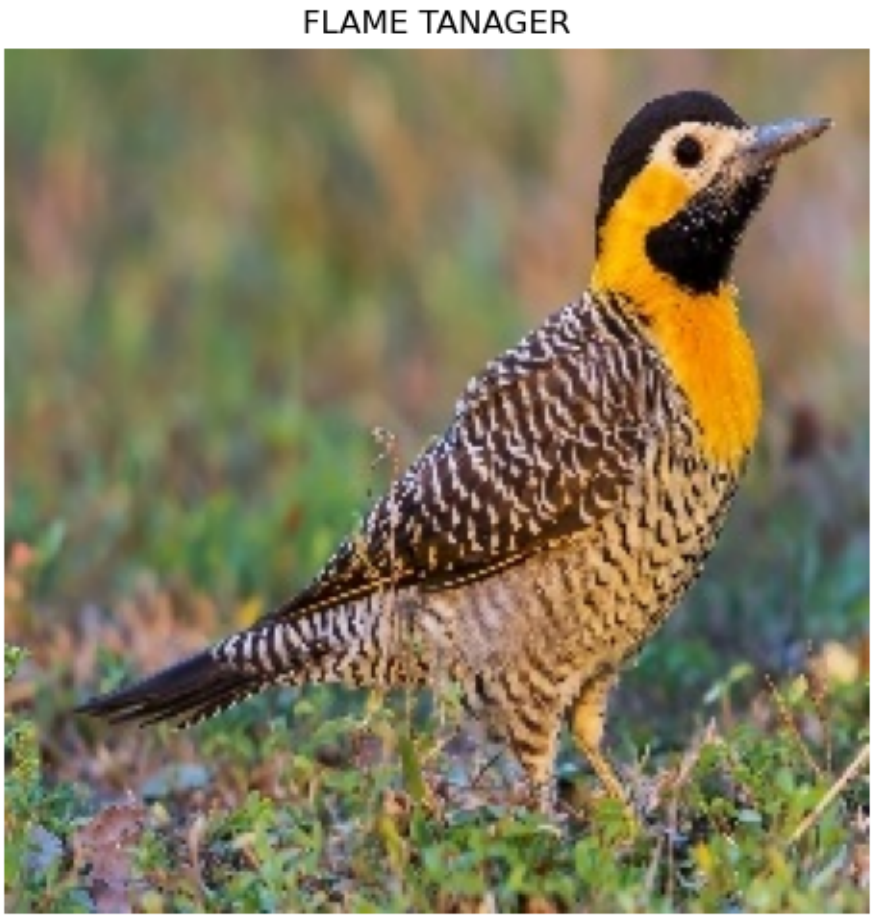
[42]:

y_true	Simple	Complex	Gray	56	112
D-ARNAUDS BARBET	3	3	2	3	3
CEDAR WAXWING	3	3	3	3	3
ALEXANDRINE PARAKEET	5	5	1	4	4
COPPERSMITH BARBET	4	5	1	4	4
GAMBELS QUAIL	5	5	4	4	4
BOBOLINK	4	3	3	4	4
CAMPO FLICKER	4	5	2	5	4
BLUE HERON	2	3	3	4	4
BORNEAN PHEASANT	5	5	5	5	5
CALIFORNIA QUAIL	5	5	5	5	5

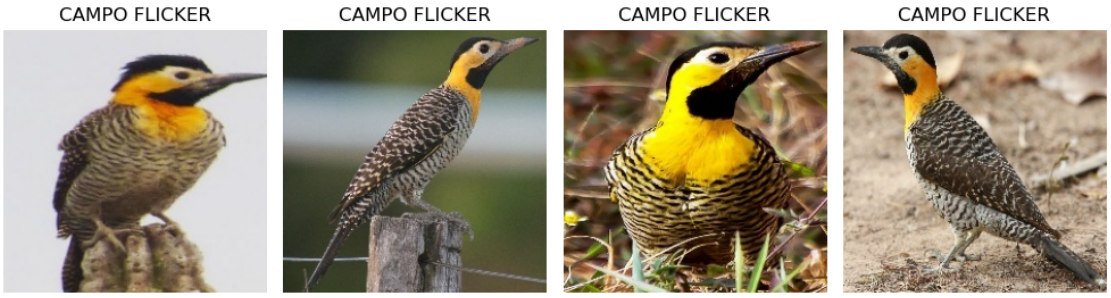
BALD IBIS	4	2	4	5	5
BALD EAGLE	3	1	2	4	5
ASIAN CRESTED IBIS	5	4	4	5	5
CUBAN TROGON	5	5	4	5	5
ARARIPE MANAKIN	5	5	3	5	5
EURASIAN BULLFINCH	5	5	4	5	5
EURASIAN GOLDEN ORIOLE	4	2	1	5	5
FLAME TANAGER	3	4	1	5	5
FRILL BACK PIGEON	2	4	3	4	5
GREAT KISKADEE	5	5	2	5	5

[75]:
```
show_class('CAMPO FLICKER', preds_112)
```

```
Incorrectly labeled CAMPO FLICKER
```




```
Correctly labeled CAMPO FLICKER
```



10.3 224x224

For the final step, we added one CNN layer with a 224x224 input size, again using MaxPooling to scale the output down to 112x112.

[43]:
```
model_setup()

model = Sequential()

model.add(Conv2D(16, 3, padding='same', activation='leaky_relu',␣
 ↪input_shape=(224, 224, 3)))

model.add(MaxPooling2D(2))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 224, 224, 16)      448

 max_pooling2d_4 (MaxPoolin  (None, 112, 112, 16)      0
 g2D)

=================================================================
Total params: 448 (1.75 KB)
Trainable params: 448 (1.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[44]:
```
model_112 = keras.saving.load_model('model_112.keras')

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

```python
# 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[2:]:
    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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 224, 224, 16)      448

 max_pooling2d_4 (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: 239444 (935.33 KB)
Trainable params: 448 (1.75 KB)
Non-trainable params: 238996 (933.58 KB)
_________________________________________________________________
```

[45]:
```
# 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 [==============================] - 8s 450ms/step - loss: 5.2142 -
accuracy: 0.5275 - val_loss: 0.8083 - val_accuracy: 0.8200
Epoch 2/100
16/16 [==============================] - 7s 436ms/step - loss: 0.1594 -
accuracy: 0.9510 - val_loss: 0.5972 - val_accuracy: 0.8300
Epoch 3/100
16/16 [==============================] - 7s 444ms/step - loss: 0.0723 -
accuracy: 0.9755 - val_loss: 0.6432 - val_accuracy: 0.8400
Epoch 4/100
16/16 [==============================] - 7s 463ms/step - loss: 0.0828 -
accuracy: 0.9740 - val_loss: 0.6589 - val_accuracy: 0.8100
Epoch 5/100
16/16 [==============================] - 7s 436ms/step - loss: 0.0686 -
accuracy: 0.9760 - val_loss: 0.5700 - val_accuracy: 0.8400
Epoch 6/100
16/16 [==============================] - 7s 456ms/step - loss: 0.0327 -
accuracy: 0.9900 - val_loss: 0.5767 - val_accuracy: 0.8300
Epoch 7/100
16/16 [==============================] - 7s 449ms/step - loss: 0.0373 -
```

```
accuracy: 0.9855 - val_loss: 0.5726 - val_accuracy: 0.8600
Epoch 8/100
16/16 [==============================] - 7s 432ms/step - loss: 0.0437 -
accuracy: 0.9830 - val_loss: 0.5785 - val_accuracy: 0.8600
Epoch 9/100
16/16 [==============================] - 7s 434ms/step - loss: 0.0372 -
accuracy: 0.9880 - val_loss: 0.5932 - val_accuracy: 0.8500
Epoch 10/100
16/16 [==============================] - 7s 453ms/step - loss: 0.0348 -
accuracy: 0.9890 - val_loss: 0.6088 - val_accuracy: 0.8400
Epoch 11/100
16/16 [==============================] - 7s 466ms/step - loss: 0.0323 -
accuracy: 0.9885 - val_loss: 0.5889 - val_accuracy: 0.8400
Epoch 12/100
16/16 [==============================] - 7s 438ms/step - loss: 0.0448 -
accuracy: 0.9870 - val_loss: 0.6153 - val_accuracy: 0.8500
Epoch 13/100
16/16 [==============================] - 7s 463ms/step - loss: 0.0286 -
accuracy: 0.9910 - val_loss: 0.6230 - val_accuracy: 0.8500
Epoch 14/100
16/16 [==============================] - 7s 426ms/step - loss: 0.0226 -
accuracy: 0.9910 - val_loss: 0.6306 - val_accuracy: 0.8500
Epoch 15/100
16/16 [==============================] - 7s 438ms/step - loss: 0.0369 -
accuracy: 0.9880 - val_loss: 0.7169 - val_accuracy: 0.7900
Epoch 16/100
16/16 [==============================] - 7s 446ms/step - loss: 0.0333 -
accuracy: 0.9890 - val_loss: 0.5862 - val_accuracy: 0.8800
Epoch 17/100
16/16 [==============================] - 7s 434ms/step - loss: 0.0280 -
accuracy: 0.9905 - val_loss: 0.6182 - val_accuracy: 0.8400
Epoch 18/100
16/16 [==============================] - 7s 446ms/step - loss: 0.0316 -
accuracy: 0.9905 - val_loss: 0.7347 - val_accuracy: 0.8100
Epoch 19/100
16/16 [==============================] - 7s 436ms/step - loss: 0.0271 -
accuracy: 0.9900 - val_loss: 0.6610 - val_accuracy: 0.8600
Epoch 20/100
16/16 [==============================] - 7s 445ms/step - loss: 0.0302 -
accuracy: 0.9905 - val_loss: 0.7393 - val_accuracy: 0.8400
Epoch 21/100
16/16 [==============================] - 6s 396ms/step - loss: 0.0366 -
accuracy: 0.9875 - val_loss: 0.5631 - val_accuracy: 0.8700
Epoch 22/100
16/16 [==============================] - 7s 446ms/step - loss: 0.0302 -
accuracy: 0.9915 - val_loss: 0.6316 - val_accuracy: 0.8500
Epoch 23/100
16/16 [==============================] - 7s 451ms/step - loss: 0.0265 -
```

```
accuracy: 0.9940 - val_loss: 0.8046 - val_accuracy: 0.8100
Epoch 24/100
16/16 [==============================] - 8s 508ms/step - loss: 0.0385 -
accuracy: 0.9885 - val_loss: 0.6549 - val_accuracy: 0.8500
Epoch 25/100
16/16 [==============================] - 7s 450ms/step - loss: 0.0259 -
accuracy: 0.9890 - val_loss: 0.6940 - val_accuracy: 0.8600
Epoch 26/100
16/16 [==============================] - 7s 440ms/step - loss: 0.0261 -
accuracy: 0.9920 - val_loss: 0.7696 - val_accuracy: 0.8300
Epoch 27/100
16/16 [==============================] - 7s 424ms/step - loss: 0.0267 -
accuracy: 0.9910 - val_loss: 0.6462 - val_accuracy: 0.8500
Epoch 28/100
16/16 [==============================] - 7s 418ms/step - loss: 0.0221 -
accuracy: 0.9940 - val_loss: 0.6448 - val_accuracy: 0.8500
Epoch 29/100
16/16 [==============================] - 7s 432ms/step - loss: 0.0305 -
accuracy: 0.9900 - val_loss: 0.6597 - val_accuracy: 0.8500
Epoch 30/100
16/16 [==============================] - 8s 490ms/step - loss: 0.0193 -
accuracy: 0.9945 - val_loss: 0.6576 - val_accuracy: 0.8600
Epoch 31/100
16/16 [==============================] - 8s 480ms/step - loss: 0.0302 -
accuracy: 0.9910 - val_loss: 0.8538 - val_accuracy: 0.8500
Epoch 32/100
16/16 [==============================] - 7s 440ms/step - loss: 0.0231 -
accuracy: 0.9925 - val_loss: 0.7767 - val_accuracy: 0.8500
Epoch 33/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0307 -
accuracy: 0.9905 - val_loss: 0.6836 - val_accuracy: 0.8500
Epoch 34/100
16/16 [==============================] - 7s 430ms/step - loss: 0.0199 -
accuracy: 0.9945 - val_loss: 0.6996 - val_accuracy: 0.8600
Epoch 35/100
16/16 [==============================] - 7s 425ms/step - loss: 0.0254 -
accuracy: 0.9915 - val_loss: 0.6552 - val_accuracy: 0.8700
Epoch 36/100
16/16 [==============================] - 7s 435ms/step - loss: 0.0241 -
accuracy: 0.9910 - val_loss: 0.6394 - val_accuracy: 0.8700
Epoch 37/100
16/16 [==============================] - 7s 435ms/step - loss: 0.0281 -
accuracy: 0.9900 - val_loss: 0.6483 - val_accuracy: 0.8700
Epoch 38/100
16/16 [==============================] - 7s 429ms/step - loss: 0.0140 -
accuracy: 0.9960 - val_loss: 0.6689 - val_accuracy: 0.8700
Epoch 39/100
16/16 [==============================] - 7s 431ms/step - loss: 0.0280 -
```

```
accuracy: 0.9910 - val_loss: 0.6746 - val_accuracy: 0.8600
Epoch 40/100
16/16 [==============================] - 7s 442ms/step - loss: 0.0216 -
accuracy: 0.9930 - val_loss: 0.6968 - val_accuracy: 0.8700
Epoch 41/100
16/16 [==============================] - 7s 445ms/step - loss: 0.0191 -
accuracy: 0.9935 - val_loss: 0.7638 - val_accuracy: 0.8500
Epoch 42/100
16/16 [==============================] - 7s 412ms/step - loss: 0.0280 -
accuracy: 0.9925 - val_loss: 0.7182 - val_accuracy: 0.8500
Epoch 43/100
16/16 [==============================] - 7s 428ms/step - loss: 0.0214 -
accuracy: 0.9910 - val_loss: 0.7625 - val_accuracy: 0.8500
Epoch 44/100
16/16 [==============================] - 7s 433ms/step - loss: 0.0270 -
accuracy: 0.9915 - val_loss: 0.7040 - val_accuracy: 0.8800
Epoch 45/100
16/16 [==============================] - 7s 460ms/step - loss: 0.0269 -
accuracy: 0.9915 - val_loss: 0.7576 - val_accuracy: 0.8200
Epoch 46/100
16/16 [==============================] - 8s 477ms/step - loss: 0.0155 -
accuracy: 0.9925 - val_loss: 0.7339 - val_accuracy: 0.8300
Epoch 47/100
16/16 [==============================] - 7s 451ms/step - loss: 0.0202 -
accuracy: 0.9955 - val_loss: 0.6794 - val_accuracy: 0.8800
Epoch 48/100
16/16 [==============================] - 7s 437ms/step - loss: 0.0200 -
accuracy: 0.9930 - val_loss: 0.7837 - val_accuracy: 0.8500
Epoch 49/100
16/16 [==============================] - 7s 460ms/step - loss: 0.0245 -
accuracy: 0.9930 - val_loss: 0.7183 - val_accuracy: 0.8500
Epoch 50/100
16/16 [==============================] - 7s 424ms/step - loss: 0.0223 -
accuracy: 0.9925 - val_loss: 0.7721 - val_accuracy: 0.8700
Epoch 51/100
16/16 [==============================] - 7s 431ms/step - loss: 0.0189 -
accuracy: 0.9940 - val_loss: 0.7547 - val_accuracy: 0.8200
Epoch 52/100
16/16 [==============================] - 7s 447ms/step - loss: 0.0370 -
accuracy: 0.9910 - val_loss: 0.9500 - val_accuracy: 0.8500
Epoch 53/100
16/16 [==============================] - 7s 408ms/step - loss: 0.0229 -
accuracy: 0.9935 - val_loss: 0.8540 - val_accuracy: 0.7900
Epoch 54/100
16/16 [==============================] - 7s 428ms/step - loss: 0.0168 -
accuracy: 0.9935 - val_loss: 0.8260 - val_accuracy: 0.8500
Epoch 55/100
16/16 [==============================] - 7s 450ms/step - loss: 0.0183 -
```

```
accuracy: 0.9930 - val_loss: 0.7188 - val_accuracy: 0.8700
Epoch 56/100
16/16 [==============================] - 8s 477ms/step - loss: 0.0214 -
accuracy: 0.9930 - val_loss: 0.9207 - val_accuracy: 0.8600
Epoch 57/100
16/16 [==============================] - 8s 472ms/step - loss: 0.0146 -
accuracy: 0.9940 - val_loss: 0.7824 - val_accuracy: 0.8300
Epoch 58/100
16/16 [==============================] - 7s 463ms/step - loss: 0.0200 -
accuracy: 0.9930 - val_loss: 0.8971 - val_accuracy: 0.8500
Epoch 59/100
16/16 [==============================] - 7s 430ms/step - loss: 0.0191 -
accuracy: 0.9925 - val_loss: 0.8315 - val_accuracy: 0.8400
Epoch 60/100
16/16 [==============================] - 8s 489ms/step - loss: 0.0181 -
accuracy: 0.9935 - val_loss: 0.7383 - val_accuracy: 0.8500
Epoch 61/100
16/16 [==============================] - 7s 439ms/step - loss: 0.0187 -
accuracy: 0.9920 - val_loss: 0.8193 - val_accuracy: 0.8500
Epoch 62/100
16/16 [==============================] - 7s 449ms/step - loss: 0.0170 -
accuracy: 0.9945 - val_loss: 0.8378 - val_accuracy: 0.8500
Epoch 63/100
16/16 [==============================] - 7s 457ms/step - loss: 0.0185 -
accuracy: 0.9940 - val_loss: 0.7692 - val_accuracy: 0.8600
Epoch 64/100
16/16 [==============================] - 7s 415ms/step - loss: 0.0117 -
accuracy: 0.9950 - val_loss: 0.8670 - val_accuracy: 0.8200
Epoch 65/100
16/16 [==============================] - 7s 430ms/step - loss: 0.0311 -
accuracy: 0.9895 - val_loss: 0.8169 - val_accuracy: 0.8400
Epoch 66/100
16/16 [==============================] - 7s 417ms/step - loss: 0.0138 -
accuracy: 0.9950 - val_loss: 0.7159 - val_accuracy: 0.8500
Epoch 67/100
16/16 [==============================] - 7s 445ms/step - loss: 0.0163 -
accuracy: 0.9940 - val_loss: 0.7961 - val_accuracy: 0.8500
Epoch 68/100
16/16 [==============================] - 7s 447ms/step - loss: 0.0228 -
accuracy: 0.9915 - val_loss: 0.7404 - val_accuracy: 0.8500
Epoch 69/100
16/16 [==============================] - 7s 441ms/step - loss: 0.0111 -
accuracy: 0.9975 - val_loss: 0.7267 - val_accuracy: 0.8600
Epoch 70/100
16/16 [==============================] - 7s 442ms/step - loss: 0.0104 -
accuracy: 0.9960 - val_loss: 0.8317 - val_accuracy: 0.8700
Epoch 71/100
16/16 [==============================] - 7s 444ms/step - loss: 0.0195 -
```

```
accuracy: 0.9925 - val_loss: 0.7670 - val_accuracy: 0.8600
Epoch 72/100
16/16 [==============================] - 7s 453ms/step - loss: 0.0136 -
accuracy: 0.9960 - val_loss: 0.8815 - val_accuracy: 0.8400
Epoch 73/100
16/16 [==============================] - 7s 418ms/step - loss: 0.0246 -
accuracy: 0.9905 - val_loss: 0.7983 - val_accuracy: 0.8700
Epoch 74/100
16/16 [==============================] - 7s 432ms/step - loss: 0.0135 -
accuracy: 0.9960 - val_loss: 0.8060 - val_accuracy: 0.8500
Epoch 75/100
16/16 [==============================] - 7s 434ms/step - loss: 0.0163 -
accuracy: 0.9930 - val_loss: 0.7346 - val_accuracy: 0.8700
Epoch 76/100
16/16 [==============================] - 7s 418ms/step - loss: 0.0221 -
accuracy: 0.9910 - val_loss: 0.7863 - val_accuracy: 0.8500
Epoch 77/100
16/16 [==============================] - 7s 449ms/step - loss: 0.0186 -
accuracy: 0.9940 - val_loss: 0.8490 - val_accuracy: 0.8500
Epoch 78/100
16/16 [==============================] - 7s 434ms/step - loss: 0.0226 -
accuracy: 0.9925 - val_loss: 0.7730 - val_accuracy: 0.8700
Epoch 79/100
16/16 [==============================] - 7s 446ms/step - loss: 0.0134 -
accuracy: 0.9955 - val_loss: 0.7914 - val_accuracy: 0.8300
Epoch 80/100
16/16 [==============================] - 7s 423ms/step - loss: 0.0239 -
accuracy: 0.9925 - val_loss: 0.8118 - val_accuracy: 0.8600
Epoch 81/100
16/16 [==============================] - 7s 432ms/step - loss: 0.0192 -
accuracy: 0.9930 - val_loss: 0.8567 - val_accuracy: 0.8500
Epoch 82/100
16/16 [==============================] - 7s 425ms/step - loss: 0.0171 -
accuracy: 0.9950 - val_loss: 0.8722 - val_accuracy: 0.8700
Epoch 83/100
16/16 [==============================] - 7s 434ms/step - loss: 0.0129 -
accuracy: 0.9955 - val_loss: 0.9083 - val_accuracy: 0.8400
Epoch 84/100
16/16 [==============================] - 7s 423ms/step - loss: 0.0212 -
accuracy: 0.9935 - val_loss: 0.8678 - val_accuracy: 0.8700
Epoch 85/100
16/16 [==============================] - 7s 411ms/step - loss: 0.0185 -
accuracy: 0.9940 - val_loss: 0.9037 - val_accuracy: 0.8400
Epoch 86/100
16/16 [==============================] - 7s 430ms/step - loss: 0.0095 -
accuracy: 0.9960 - val_loss: 0.8887 - val_accuracy: 0.8400
Epoch 87/100
16/16 [==============================] - 6s 407ms/step - loss: 0.0497 -
```

```
accuracy: 0.9850 - val_loss: 0.7350 - val_accuracy: 0.8800
Epoch 88/100
16/16 [==============================] - 7s 414ms/step - loss: 0.0210 -
accuracy: 0.9920 - val_loss: 0.7959 - val_accuracy: 0.8400
Epoch 89/100
16/16 [==============================] - 7s 425ms/step - loss: 0.0273 -
accuracy: 0.9930 - val_loss: 0.8523 - val_accuracy: 0.8600
Epoch 90/100
16/16 [==============================] - 7s 412ms/step - loss: 0.0324 -
accuracy: 0.9910 - val_loss: 0.8888 - val_accuracy: 0.8600
Epoch 91/100
16/16 [==============================] - 7s 438ms/step - loss: 0.0274 -
accuracy: 0.9935 - val_loss: 0.8642 - val_accuracy: 0.8100
Epoch 92/100
16/16 [==============================] - 7s 452ms/step - loss: 0.0304 -
accuracy: 0.9910 - val_loss: 0.7511 - val_accuracy: 0.8800
Epoch 93/100
16/16 [==============================] - 7s 464ms/step - loss: 0.0206 -
accuracy: 0.9955 - val_loss: 0.7381 - val_accuracy: 0.8800
Epoch 94/100
16/16 [==============================] - 7s 458ms/step - loss: 0.0213 -
accuracy: 0.9915 - val_loss: 0.9245 - val_accuracy: 0.8200
Epoch 95/100
16/16 [==============================] - 8s 483ms/step - loss: 0.0235 -
accuracy: 0.9920 - val_loss: 0.7996 - val_accuracy: 0.8700
Epoch 96/100
16/16 [==============================] - 7s 465ms/step - loss: 0.0218 -
accuracy: 0.9935 - val_loss: 0.7321 - val_accuracy: 0.8700
Epoch 97/100
16/16 [==============================] - 8s 496ms/step - loss: 0.0201 -
accuracy: 0.9920 - val_loss: 0.7754 - val_accuracy: 0.8600
Epoch 98/100
16/16 [==============================] - 8s 503ms/step - loss: 0.0159 -
accuracy: 0.9940 - val_loss: 0.7936 - val_accuracy: 0.8600
Epoch 99/100
16/16 [==============================] - 8s 470ms/step - loss: 0.0157 -
accuracy: 0.9945 - val_loss: 0.7142 - val_accuracy: 0.8400
Epoch 100/100
16/16 [==============================] - 7s 409ms/step - loss: 0.0121 -
accuracy: 0.9955 - val_loss: 0.8066 - val_accuracy: 0.8500
```

[46]:
```
model_224 = keras.saving.load_model('model_224.keras')

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

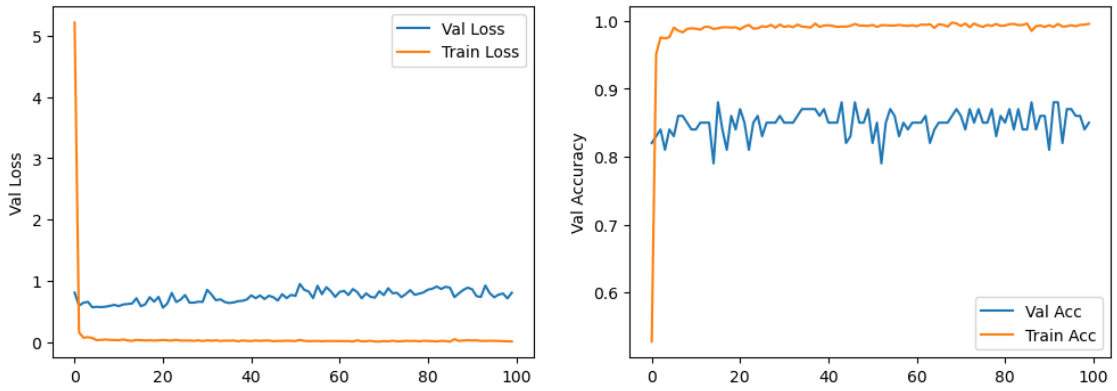
```
4/4 [==============================] - 0s 29ms/step - loss: 0.5862 - accuracy:
0.8800
```

[46]: [0.586241602897644, 0.8799999952316284]

10.3.1 Model Evaluation

Unfortunately, the new layer decreased the accuracy that we'd gained in the previous model. However, the loss also decreased slightly, indicating that there were some features that had been helpful in getting the model closer to the ground truth even if it ultimately predicted more birds incorrectly.

[47]:
```
plot_loss(history_224)
```



10.3.2 Prediction Evaluation

The decrease in accuracy appears to be an issue of Alexandrine Parakeet, Bald Ibis, and Bald Eagle. Blue Heron has actually been better predicted, and others have remained the same.

[48]:
```
preds_224 = pred_eval(model_224)
correct_224 = count_correct(preds_224)

correct_overall['224'] = correct_224['Correct']

correct_overall.sort_values('224')
```

```
4/4 [==============================] - 0s 27ms/step
```

[48]:

y_true	Simple	Complex	Gray	56	112	224
ALEXANDRINE PARAKEET	5	5	1	4	4	3
D-ARNAUDS BARBET	3	3	2	3	3	3
CEDAR WAXWING	3	3	3	3	3	3
COPPERSMITH BARBET	4	5	1	4	4	4
GAMBELS QUAIL	5	5	4	4	4	4
BOBOLINK	4	3	3	4	4	4
CAMPO FLICKER	4	5	2	5	4	4
BALD IBIS	4	2	4	5	5	4

BALD EAGLE	3	1	2	4	5	4
BORNEAN PHEASANT	5	5	5	5	5	5
CALIFORNIA QUAIL	5	5	5	5	5	5
ASIAN CRESTED IBIS	5	4	4	5	5	5
CUBAN TROGON	5	5	4	5	5	5
ARARIPE MANAKIN	5	5	3	5	5	5
EURASIAN BULLFINCH	5	5	4	5	5	5
EURASIAN GOLDEN ORIOLE	4	2	1	5	5	5
FLAME TANAGER	3	4	1	5	5	5
FRILL BACK PIGEON	2	4	3	4	5	5
BLUE HERON	2	3	3	4	4	5
GREAT KISKADEE	5	5	2	5	5	5

10.3.3 Images of Misclassified Birds

Examining the birds that perfomred worse, it does appear that the model learned some odd features from the 224x224 images. While there are some similarities in some cases, some of the misclassifications indicate that model has picked out irrelavent features for its predictions.

[76]:
```
show_class('BALD IBIS', preds_224)

show_class('BALD EAGLE', preds_224)

show_class('ALEXANDRINE PARAKEET', preds_224)
```

```
Incorrectly labeled BALD IBIS
```

EURASIAN BULLFINCH

Correctly labeled BALD IBIS

Incorrectly labeled BALD EAGLE

ASIAN CRESTED IBIS

Correctly labeled BALD EAGLE

BALD EAGLE

BALD EAGLE

BALD EAGLE

BALD EAGLE

Incorrectly labeled ALEXANDRINE PARAKEET

Correctly labeled ALEXANDRINE PARAKEET

[77]:
```
plot_birds(['EURASIAN BULLFINCH', 'ASIAN CRESTED IBIS'], 4)
```

11 Add more Images

In reviewing the images that were poorly calssified across all three models, 2 in particular stood out: D-ARNAUDS BARBET and CEDAR WAXWING. To see if we could improve the version of the mode with the 224x224 layer, we decided to add 50 additional images of each of those birds to the training dataset.

We then retrained the 224x224 model with the new data, and also unlocked teh weights of the 112x112 layer to see if we could get gains in both accuracy and loss by giving both layers more information.

[50]:
```
'''Original code to load additional data'''
# 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 = ['CEDAR WAXWING', 'D-ARNAUDS BARBET']

# # 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])
```

[50]:
```
'Original code to load additional data'
```

[51]:
```
# Load additional images
new_train_data = pickle.load(open('./saved_data/bird_new_train_data.pkl', 'rb'))
```

[52]:
```
# Add images to training dataset
for item in new_train_data:
    train_data.append(item)

len(train_data)
```

[52]:
```
2100
```

[53]:
```
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)
```

[54]:
```
model_setup()

model = Sequential()

model.add(Conv2D(16, 3, padding='same', activation='leaky_relu',␣
 ↪input_shape=(224, 224, 3)))

model.add(MaxPooling2D(2))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 224, 224, 16)      448

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

```
 g2D)

=================================================================
Total params: 448 (1.75 KB)
Trainable params: 448 (1.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[55]:
```
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_7 (Conv2D)           (None, 224, 224, 16)      448

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

[56]:
```
# Set callbacks

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

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

history_new_img = 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 [==============================] - 9s 514ms/step - loss: 2.6104 -
accuracy: 0.7510 - val_loss: 0.8052 - val_accuracy: 0.7600
Epoch 2/100
17/17 [==============================] - 9s 525ms/step - loss: 0.2933 -
accuracy: 0.9314 - val_loss: 0.6742 - val_accuracy: 0.8600
Epoch 3/100
```

```
17/17 [==============================] - 9s 515ms/step - loss: 0.3987 -
accuracy: 0.9214 - val_loss: 0.6869 - val_accuracy: 0.8600
Epoch 4/100
17/17 [==============================] - 8s 499ms/step - loss: 0.1782 -
accuracy: 0.9662 - val_loss: 0.8370 - val_accuracy: 0.8000
Epoch 5/100
17/17 [==============================] - 8s 490ms/step - loss: 0.2818 -
accuracy: 0.9400 - val_loss: 0.8272 - val_accuracy: 0.8100
Epoch 6/100
17/17 [==============================] - 9s 514ms/step - loss: 0.2012 -
accuracy: 0.9524 - val_loss: 0.6672 - val_accuracy: 0.8300
Epoch 7/100
17/17 [==============================] - 8s 498ms/step - loss: 0.2840 -
accuracy: 0.9390 - val_loss: 0.6120 - val_accuracy: 0.8400
Epoch 8/100
17/17 [==============================] - 9s 509ms/step - loss: 0.1831 -
accuracy: 0.9676 - val_loss: 0.5355 - val_accuracy: 0.8700
Epoch 9/100
17/17 [==============================] - 9s 502ms/step - loss: 0.3231 -
accuracy: 0.9367 - val_loss: 0.7098 - val_accuracy: 0.8100
Epoch 10/100
17/17 [==============================] - 8s 495ms/step - loss: 0.1707 -
accuracy: 0.9676 - val_loss: 0.5343 - val_accuracy: 0.8600
Epoch 11/100
17/17 [==============================] - 8s 479ms/step - loss: 0.1629 -
accuracy: 0.9662 - val_loss: 0.7085 - val_accuracy: 0.8200
Epoch 12/100
17/17 [==============================] - 8s 494ms/step - loss: 0.1667 -
accuracy: 0.9648 - val_loss: 0.5188 - val_accuracy: 0.8800
Epoch 13/100
17/17 [==============================] - 8s 456ms/step - loss: 0.1605 -
accuracy: 0.9648 - val_loss: 1.0141 - val_accuracy: 0.7400
Epoch 14/100
17/17 [==============================] - 8s 459ms/step - loss: 0.1675 -
accuracy: 0.9605 - val_loss: 0.7397 - val_accuracy: 0.8300
Epoch 15/100
17/17 [==============================] - 8s 452ms/step - loss: 0.1556 -
accuracy: 0.9681 - val_loss: 0.5570 - val_accuracy: 0.8700
Epoch 16/100
17/17 [==============================] - 8s 493ms/step - loss: 0.1421 -
accuracy: 0.9648 - val_loss: 0.4875 - val_accuracy: 0.8800
Epoch 17/100
17/17 [==============================] - 9s 511ms/step - loss: 0.1550 -
accuracy: 0.9619 - val_loss: 0.9594 - val_accuracy: 0.7900
Epoch 18/100
17/17 [==============================] - 10s 582ms/step - loss: 0.1498 -
accuracy: 0.9729 - val_loss: 0.5257 - val_accuracy: 0.8800
Epoch 19/100
```

```
17/17 [==============================] - 10s 564ms/step - loss: 0.1321 -
accuracy: 0.9724 - val_loss: 0.6193 - val_accuracy: 0.7800
Epoch 20/100
17/17 [==============================] - 10s 603ms/step - loss: 0.1623 -
accuracy: 0.9652 - val_loss: 0.6383 - val_accuracy: 0.8100
Epoch 21/100
17/17 [==============================] - 12s 699ms/step - loss: 0.1296 -
accuracy: 0.9710 - val_loss: 0.8379 - val_accuracy: 0.7300
Epoch 22/100
17/17 [==============================] - 12s 729ms/step - loss: 0.1173 -
accuracy: 0.9752 - val_loss: 0.5071 - val_accuracy: 0.9000
Epoch 23/100
17/17 [==============================] - 12s 727ms/step - loss: 0.1447 -
accuracy: 0.9671 - val_loss: 0.4682 - val_accuracy: 0.8800
Epoch 24/100
17/17 [==============================] - 13s 762ms/step - loss: 0.1544 -
accuracy: 0.9681 - val_loss: 0.4962 - val_accuracy: 0.8600
Epoch 25/100
17/17 [==============================] - 14s 803ms/step - loss: 0.1122 -
accuracy: 0.9781 - val_loss: 0.5114 - val_accuracy: 0.8700
Epoch 26/100
17/17 [==============================] - 14s 843ms/step - loss: 0.1350 -
accuracy: 0.9671 - val_loss: 0.5648 - val_accuracy: 0.8500
Epoch 27/100
17/17 [==============================] - 15s 895ms/step - loss: 0.1174 -
accuracy: 0.9767 - val_loss: 0.4854 - val_accuracy: 0.8300
Epoch 28/100
17/17 [==============================] - 12s 665ms/step - loss: 0.1300 -
accuracy: 0.9714 - val_loss: 0.7926 - val_accuracy: 0.8100
Epoch 29/100
17/17 [==============================] - 10s 584ms/step - loss: 0.1474 -
accuracy: 0.9671 - val_loss: 0.5334 - val_accuracy: 0.8400
Epoch 30/100
17/17 [==============================] - 10s 569ms/step - loss: 0.1320 -
accuracy: 0.9748 - val_loss: 0.5035 - val_accuracy: 0.8600
Epoch 31/100
17/17 [==============================] - 10s 602ms/step - loss: 0.1295 -
accuracy: 0.9700 - val_loss: 0.4346 - val_accuracy: 0.8600
Epoch 32/100
17/17 [==============================] - 10s 583ms/step - loss: 0.1314 -
accuracy: 0.9733 - val_loss: 0.5117 - val_accuracy: 0.8100
Epoch 33/100
17/17 [==============================] - 11s 627ms/step - loss: 0.1214 -
accuracy: 0.9771 - val_loss: 0.5171 - val_accuracy: 0.8200
Epoch 34/100
17/17 [==============================] - 11s 629ms/step - loss: 0.1314 -
accuracy: 0.9719 - val_loss: 0.6363 - val_accuracy: 0.8100
Epoch 35/100
```

```
17/17 [==============================] - 10s 614ms/step - loss: 0.1182 -
accuracy: 0.9724 - val_loss: 0.6835 - val_accuracy: 0.8300
Epoch 36/100
17/17 [==============================] - 11s 628ms/step - loss: 0.1378 -
accuracy: 0.9681 - val_loss: 0.4659 - val_accuracy: 0.8500
Epoch 37/100
17/17 [==============================] - 11s 660ms/step - loss: 0.1074 -
accuracy: 0.9786 - val_loss: 0.4925 - val_accuracy: 0.8600
Epoch 38/100
17/17 [==============================] - 12s 695ms/step - loss: 0.1343 -
accuracy: 0.9681 - val_loss: 0.6771 - val_accuracy: 0.7900
Epoch 39/100
17/17 [==============================] - 13s 753ms/step - loss: 0.1243 -
accuracy: 0.9705 - val_loss: 0.5387 - val_accuracy: 0.8300
Epoch 40/100
17/17 [==============================] - 14s 799ms/step - loss: 0.1009 -
accuracy: 0.9795 - val_loss: 0.5832 - val_accuracy: 0.8200
Epoch 41/100
17/17 [==============================] - 14s 827ms/step - loss: 0.1399 -
accuracy: 0.9686 - val_loss: 0.5021 - val_accuracy: 0.8800
Epoch 42/100
17/17 [==============================] - 16s 924ms/step - loss: 0.1090 -
accuracy: 0.9786 - val_loss: 0.4680 - val_accuracy: 0.8700
Epoch 43/100
17/17 [==============================] - 12s 705ms/step - loss: 0.1244 -
accuracy: 0.9729 - val_loss: 0.4754 - val_accuracy: 0.8600
Epoch 44/100
17/17 [==============================] - 10s 600ms/step - loss: 0.1164 -
accuracy: 0.9733 - val_loss: 0.4621 - val_accuracy: 0.8800
Epoch 45/100
17/17 [==============================] - 9s 559ms/step - loss: 0.1176 -
accuracy: 0.9786 - val_loss: 0.4526 - val_accuracy: 0.8800
Epoch 46/100
17/17 [==============================] - 10s 589ms/step - loss: 0.1133 -
accuracy: 0.9743 - val_loss: 0.4139 - val_accuracy: 0.8900
Epoch 47/100
17/17 [==============================] - 9s 509ms/step - loss: 0.1019 -
accuracy: 0.9781 - val_loss: 0.4719 - val_accuracy: 0.8600
Epoch 48/100
17/17 [==============================] - 9s 530ms/step - loss: 0.1067 -
accuracy: 0.9757 - val_loss: 0.5151 - val_accuracy: 0.8800
Epoch 49/100
17/17 [==============================] - 9s 560ms/step - loss: 0.1045 -
accuracy: 0.9786 - val_loss: 0.4561 - val_accuracy: 0.8800
Epoch 50/100
17/17 [==============================] - 10s 591ms/step - loss: 0.1013 -
accuracy: 0.9781 - val_loss: 0.5773 - val_accuracy: 0.7900
Epoch 51/100
```

```
17/17 [==============================] - 11s 631ms/step - loss: 0.1046 -
accuracy: 0.9795 - val_loss: 0.5113 - val_accuracy: 0.8400
Epoch 52/100
17/17 [==============================] - 11s 667ms/step - loss: 0.1130 -
accuracy: 0.9743 - val_loss: 0.5217 - val_accuracy: 0.8700
Epoch 53/100
17/17 [==============================] - 12s 697ms/step - loss: 0.1075 -
accuracy: 0.9767 - val_loss: 0.3924 - val_accuracy: 0.9100
Epoch 54/100
17/17 [==============================] - 12s 735ms/step - loss: 0.1082 -
accuracy: 0.9786 - val_loss: 0.5457 - val_accuracy: 0.8500
Epoch 55/100
17/17 [==============================] - 13s 760ms/step - loss: 0.1037 -
accuracy: 0.9800 - val_loss: 0.4220 - val_accuracy: 0.8600
Epoch 56/100
17/17 [==============================] - 13s 741ms/step - loss: 0.1098 -
accuracy: 0.9743 - val_loss: 0.4596 - val_accuracy: 0.8900
Epoch 57/100
17/17 [==============================] - 14s 832ms/step - loss: 0.0985 -
accuracy: 0.9800 - val_loss: 0.4688 - val_accuracy: 0.8700
Epoch 58/100
17/17 [==============================] - 11s 635ms/step - loss: 0.0958 -
accuracy: 0.9738 - val_loss: 0.4527 - val_accuracy: 0.8900
Epoch 59/100
17/17 [==============================] - 9s 546ms/step - loss: 0.0930 -
accuracy: 0.9810 - val_loss: 0.7293 - val_accuracy: 0.8400
Epoch 60/100
17/17 [==============================] - 10s 607ms/step - loss: 0.1165 -
accuracy: 0.9776 - val_loss: 0.4675 - val_accuracy: 0.8600
Epoch 61/100
17/17 [==============================] - 10s 571ms/step - loss: 0.0992 -
accuracy: 0.9786 - val_loss: 0.5680 - val_accuracy: 0.8500
Epoch 62/100
17/17 [==============================] - 10s 566ms/step - loss: 0.1042 -
accuracy: 0.9743 - val_loss: 0.4508 - val_accuracy: 0.8600
Epoch 63/100
17/17 [==============================] - 10s 567ms/step - loss: 0.1190 -
accuracy: 0.9710 - val_loss: 0.5155 - val_accuracy: 0.8600
Epoch 64/100
17/17 [==============================] - 10s 578ms/step - loss: 0.1006 -
accuracy: 0.9776 - val_loss: 0.4338 - val_accuracy: 0.8700
Epoch 65/100
17/17 [==============================] - 10s 585ms/step - loss: 0.0876 -
accuracy: 0.9805 - val_loss: 0.3614 - val_accuracy: 0.8800
Epoch 66/100
17/17 [==============================] - 11s 662ms/step - loss: 0.0993 -
accuracy: 0.9790 - val_loss: 0.4341 - val_accuracy: 0.8500
Epoch 67/100
```

```
17/17 [==============================] - 11s 652ms/step - loss: 0.0928 -
accuracy: 0.9795 - val_loss: 0.4649 - val_accuracy: 0.8500
Epoch 68/100
17/17 [==============================] - 12s 700ms/step - loss: 0.1130 -
accuracy: 0.9748 - val_loss: 0.4475 - val_accuracy: 0.8600
Epoch 69/100
17/17 [==============================] - 12s 726ms/step - loss: 0.1052 -
accuracy: 0.9810 - val_loss: 0.4453 - val_accuracy: 0.8900
Epoch 70/100
17/17 [==============================] - 13s 788ms/step - loss: 0.0929 -
accuracy: 0.9776 - val_loss: 0.6032 - val_accuracy: 0.8100
Epoch 71/100
17/17 [==============================] - 14s 825ms/step - loss: 0.1066 -
accuracy: 0.9762 - val_loss: 0.4937 - val_accuracy: 0.8600
Epoch 72/100
17/17 [==============================] - 13s 737ms/step - loss: 0.0884 -
accuracy: 0.9786 - val_loss: 0.4788 - val_accuracy: 0.8500
Epoch 73/100
17/17 [==============================] - 10s 575ms/step - loss: 0.0960 -
accuracy: 0.9771 - val_loss: 0.5376 - val_accuracy: 0.8700
Epoch 74/100
17/17 [==============================] - 10s 599ms/step - loss: 0.0939 -
accuracy: 0.9800 - val_loss: 0.4772 - val_accuracy: 0.8700
Epoch 75/100
17/17 [==============================] - 10s 586ms/step - loss: 0.0962 -
accuracy: 0.9776 - val_loss: 0.4820 - val_accuracy: 0.8600
Epoch 76/100
17/17 [==============================] - 10s 583ms/step - loss: 0.0932 -
accuracy: 0.9805 - val_loss: 0.4210 - val_accuracy: 0.8700
Epoch 77/100
17/17 [==============================] - 10s 580ms/step - loss: 0.0924 -
accuracy: 0.9824 - val_loss: 0.4497 - val_accuracy: 0.8600
Epoch 78/100
17/17 [==============================] - 10s 589ms/step - loss: 0.1088 -
accuracy: 0.9757 - val_loss: 0.5279 - val_accuracy: 0.8600
Epoch 79/100
17/17 [==============================] - 10s 602ms/step - loss: 0.0875 -
accuracy: 0.9833 - val_loss: 0.4201 - val_accuracy: 0.8800
Epoch 80/100
17/17 [==============================] - 11s 631ms/step - loss: 0.0999 -
accuracy: 0.9757 - val_loss: 0.5654 - val_accuracy: 0.8700
Epoch 81/100
17/17 [==============================] - 13s 766ms/step - loss: 0.0858 -
accuracy: 0.9838 - val_loss: 0.5098 - val_accuracy: 0.8200
Epoch 82/100
17/17 [==============================] - 12s 682ms/step - loss: 0.1067 -
accuracy: 0.9762 - val_loss: 0.5055 - val_accuracy: 0.8800
Epoch 83/100
```

```
17/17 [==============================] - 10s 615ms/step - loss: 0.1002 -
accuracy: 0.9752 - val_loss: 0.5254 - val_accuracy: 0.8600
Epoch 84/100
17/17 [==============================] - 11s 638ms/step - loss: 0.0924 -
accuracy: 0.9790 - val_loss: 0.6986 - val_accuracy: 0.7900
Epoch 85/100
17/17 [==============================] - 11s 642ms/step - loss: 0.0830 -
accuracy: 0.9824 - val_loss: 0.4269 - val_accuracy: 0.8900
Epoch 86/100
17/17 [==============================] - 11s 647ms/step - loss: 0.0822 -
accuracy: 0.9795 - val_loss: 0.4313 - val_accuracy: 0.8800
Epoch 87/100
17/17 [==============================] - 9s 506ms/step - loss: 0.0915 -
accuracy: 0.9824 - val_loss: 0.4221 - val_accuracy: 0.8500
Epoch 88/100
17/17 [==============================] - 9s 512ms/step - loss: 0.0713 -
accuracy: 0.9848 - val_loss: 0.4498 - val_accuracy: 0.8700
Epoch 89/100
17/17 [==============================] - 9s 518ms/step - loss: 0.0820 -
accuracy: 0.9843 - val_loss: 0.4782 - val_accuracy: 0.8500
Epoch 90/100
17/17 [==============================] - 9s 516ms/step - loss: 0.0778 -
accuracy: 0.9843 - val_loss: 0.6499 - val_accuracy: 0.7800
Epoch 91/100
17/17 [==============================] - 9s 555ms/step - loss: 0.0817 -
accuracy: 0.9824 - val_loss: 0.4590 - val_accuracy: 0.8800
Epoch 92/100
17/17 [==============================] - 10s 558ms/step - loss: 0.0882 -
accuracy: 0.9819 - val_loss: 0.5335 - val_accuracy: 0.8300
Epoch 93/100
17/17 [==============================] - 9s 554ms/step - loss: 0.0951 -
accuracy: 0.9795 - val_loss: 0.4268 - val_accuracy: 0.8800
Epoch 94/100
17/17 [==============================] - 9s 544ms/step - loss: 0.0851 -
accuracy: 0.9795 - val_loss: 0.5520 - val_accuracy: 0.8500
Epoch 95/100
17/17 [==============================] - 10s 568ms/step - loss: 0.0793 -
accuracy: 0.9848 - val_loss: 0.5293 - val_accuracy: 0.8600
Epoch 96/100
17/17 [==============================] - 10s 596ms/step - loss: 0.0994 -
accuracy: 0.9762 - val_loss: 0.4923 - val_accuracy: 0.8600
Epoch 97/100
17/17 [==============================] - 11s 623ms/step - loss: 0.1015 -
accuracy: 0.9757 - val_loss: 0.4794 - val_accuracy: 0.8100
Epoch 98/100
17/17 [==============================] - 11s 657ms/step - loss: 0.0914 -
accuracy: 0.9805 - val_loss: 0.4800 - val_accuracy: 0.8800
Epoch 99/100
```

```
17/17 [==============================] - 11s 640ms/step - loss: 0.0898 -
accuracy: 0.9781 - val_loss: 0.4278 - val_accuracy: 0.8300
Epoch 100/100
17/17 [==============================] - 11s 646ms/step - loss: 0.0751 -
accuracy: 0.9852 - val_loss: 0.5544 - val_accuracy: 0.8300
```

11.1 Model Evaluation

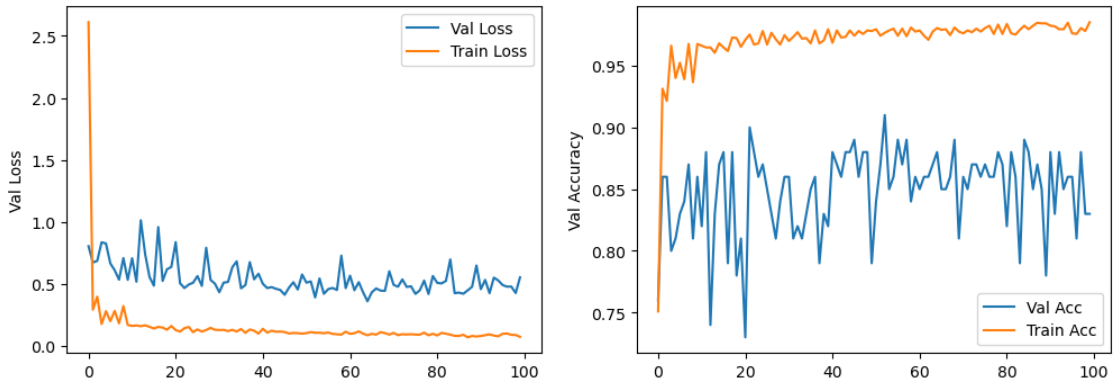
Adding the extra images did in fact increase accuracy and decrease loss. The mdoel was now performing at 91% with a loss of .39, our best scores yet.

[57]:
```
mod_new_img = keras.saving.load_model('model_new_img.keras')

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

```
4/4 [==============================] - 1s 47ms/step - loss: 0.3924 - accuracy:
0.9100
```

[57]:
```
[0.3924221098423004, 0.9100000262260437]
```

[58]:
```
plot_loss(history_new_img)
```



11.2 Prediction Evaluations

Looking at the new predictions, these additional images and training seemed to have a net positive on the model. While the Barbet did not improve, the Waxwing did. And wiht one exception (the Blue Heron), all our classes either improved or remained the same from the 224x224 model.

[59]:
```
new_img_preds = pred_eval(mod_new_img)
new_img_correct = count_correct(new_img_preds)

correct_overall['new_img'] = new_img_correct['Correct']

correct_overall.sort_values(by='new_img')
```

```
4/4 [==============================] - 0s 42ms/step
```

[59]:

y_true	Simple	Complex	Gray	56	112	224	new_img
D-ARNAUDS BARBET	3	3	2	3	3	3	3
ALEXANDRINE PARAKEET	5	5	1	4	4	3	4
BALD EAGLE	3	1	2	4	5	4	4
BLUE HERON	2	3	3	4	4	5	4
BOBOLINK	4	3	3	4	4	4	4
GAMBELS QUAIL	5	5	4	4	4	4	4
CEDAR WAXWING	3	3	3	3	3	3	4
COPPERSMITH BARBET	4	5	1	4	4	4	4
FRILL BACK PIGEON	2	4	3	4	5	5	5
FLAME TANAGER	3	4	1	5	5	5	5
EURASIAN GOLDEN ORIOLE	4	2	1	5	5	5	5
EURASIAN BULLFINCH	5	5	4	5	5	5	5
CAMPO FLICKER	4	5	2	5	4	4	5
CALIFORNIA QUAIL	5	5	5	5	5	5	5
BORNEAN PHEASANT	5	5	5	5	5	5	5
BALD IBIS	4	2	4	5	5	4	5
ASIAN CRESTED IBIS	5	4	4	5	5	5	5
ARARIPE MANAKIN	5	5	3	5	5	5	5
CUBAN TROGON	5	5	4	5	5	5	5
GREAT KISKADEE	5	5	2	5	5	5	5

12 Final Model Test

With an accuracy 10% points higher than our inital model, we decide to do a final evaluation on our test set.

[62]:
```
np.random.seed(rs)
np.random.shuffle(test_data)

X_test = []
y_test = []
for x, y in test_data:
    X_test.append(x)
    y_test.append(y)

X_test = np.array(X_test)
y_test = np.array(y_test)

y_test_vect = lb.fit_transform(y_test)
```

12.1 Results Evaluation

Unfortunately, the model did not perform as well on our test set as it did on the validation set and we achieved an accuracy of only 83%. While some classes did very well, and some that had been

consistently poor (D-Arnauds Barbet) did better, many classes saw significant decreases.

Blue Heron and Bobolink in particular has significant drops and became our most poorly predicted classes.

[63]:
```
mod_new_img.evaluate(X_test/255, y_test_vect)
```

```
1/4 [======>…] - ETA: 0s - loss: 0.9380 - accuracy:
0.81254/4 [==============================] - 0s 30ms/step - loss: 0.5899 -
accuracy: 0.8300
```

[63]:
```
[0.5899484157562256, 0.8299999833106995]
```

[64]:
```
final_preds = pred_eval(mod_new_img, X_test, y_test)
final_correct = count_correct(final_preds)

correct_overall['Test'] = final_correct['Correct']

correct_overall.sort_values('Test')
```

```
4/4 [==============================] - 0s 28ms/step
```

[64]:

y_true	Simple	Complex	Gray	56	112	224	new_img	Test
BLUE HERON	2	3	3	4	4	5	4	2
BOBOLINK	4	3	3	4	4	4	4	2
BALD IBIS	4	2	4	5	5	4	5	3
CALIFORNIA QUAIL	5	5	5	5	5	5	5	3
ALEXANDRINE PARAKEET	5	5	1	4	4	3	4	4
FRILL BACK PIGEON	2	4	3	4	5	5	5	4
EURASIAN BULLFINCH	5	5	4	5	5	5	5	4
COPPERSMITH BARBET	4	5	1	4	4	4	4	4
GAMBELS QUAIL	5	5	4	4	4	4	4	4
CAMPO FLICKER	4	5	2	5	4	4	5	4
GREAT KISKADEE	5	5	2	5	5	5	5	4
CEDAR WAXWING	3	3	3	3	3	3	4	5
BALD EAGLE	3	1	2	4	5	4	4	5
CUBAN TROGON	5	5	4	5	5	5	5	5
D-ARNAUDS BARBET	3	3	2	3	3	3	3	5
ASIAN CRESTED IBIS	5	4	4	5	5	5	5	5
EURASIAN GOLDEN ORIOLE	4	2	1	5	5	5	5	5
FLAME TANAGER	3	4	1	5	5	5	5	5
ARARIPE MANAKIN	5	5	3	5	5	5	5	5
BORNEAN PHEASANT	5	5	5	5	5	5	5	5

12.2 Class Evaluation

Exploring the two worst predicted classes, we noticed that one of the issues appeared to be distinct differences between the images in the validation and teh test set. For instance, the positions of

the blue heron in the test and validation set were very different, and the Bobolinks were in very different positions that showed very different parts of the coloration and patterning.

It appeared that we had overfit the validation in our model training, and while we had made the model slightly more flexible, we still trained it to recognize specific features that were not conistent across the validation and test set.

[79]:
```
show_class('BLUE HERON', final_preds, X_test)

show_class('BOBOLINK', final_preds, X_test)
```

```
Incorrectly labeled BLUE HERON
```


```
Correctly labeled BLUE HERON
```


```
Incorrectly labeled BOBOLINK
```

Correctly labeled BOBOLINK

[89]:
```
fig, axs = plt.subplots(ncols = 5, figsize= (15, 5))
axs = axs.ravel()

for idx, i in enumerate(X_val[y_val == 'BOBOLINK']):
    axs[idx].imshow(i)
    axs[idx].axis('off')

plt.show()

fig, axs = plt.subplots(ncols = 5, figsize= (15, 5))
axs = axs.ravel()
```

```
for idx, i in enumerate(X_val[y_val == 'BLUE HERON']):
    axs[idx].imshow(i)
    axs[idx].axis('off')

plt.show()
```