August 4, 2024

C964: Computer Science Capstone

Michael Lawrence

002680987

MLaw101@wgu.edu

Task 2 parts A, B, C and D

Part A: Letter of Transmittal

08/04/2024

Chief Editing Officer John Sparrow

The International Birder

555 Birder Lane, Dallas, TX

Dear Mr. Sparrow,

I am writing to offer a possible solution to the recent loss of our Lead Bird Identifier (LBI) and the current struggle to find a replacement.

With the news of Mr. Fowl leaving the company and no person with his skill level to replace him, I have formulated a possible solution to the problem that will allow almost anyone with slight computer knowledge to fulfill the task of bird identification.

Many advancements have been made in the way of machine learning in the past few years, because of this, and my knowledge of software engineering, I propose to design and deploy an in-house machine learning model with a simple user interface that will allow for any of the staff to simply input an image of an unclassified bird and receive that birds classification as output. I hypothesize that a classification accuracy of at least 80% can be reached with the model's first rendition. This is more than accurate enough for the initial classification that we will require to get our operations up and running again.

This product will benefit our publication in many ways. The task of bird identification will be accurate and quick. Money will be saved that would normally be allocated to retaining a bird identifier. The application will be simple and reliable enough to be run by any of the staff including interns. While not in the scope of this proposal, I feel that it is worth mentioning that once a

reliable model has been created, a mobile application front end for the model could be created and marketed to our readers for in-field identification.

The base of the model will be trained using a free and open source dataset of over 84,000 images of 524 different bird species, giving it a broad spectrum of data to start with, at no cost. The dataset to be used is a public domain (CC0 1.0 Universal License) dataset so no ethical issues are to be worried about. As new information is gained the model can be retrained to be updated with new species using our own user's submitted photos. Other free and open-source technologies will be used in order to minimize cost.

Due to my past as a software engineer at Google and my current employment as our head of web development, I will be able to create this application without the need to acquire new employees. All that I ask in return is a raise of 10% to my base yearly salary, bringing me to $100/hr.

Approximate costs for development include:

- Google Colab Pro+ cloud-based ML development environment subscription for one user, $50/month
- Anvil.works Business subscription $100/month
- MacBook Pro for development $4000

A possible timeline for the project is as follows:

- Planning and design: 2 weeks
- Development: 4 weeks
- Documentation: 1 week
- Deployment: 1 week
- Maintenance and updates: Ongoing

I sincerely hope you and the rest of the board take the time to consider this solution as I strongly believe it will save the company money and time as well as give us a forward path of growth.

Sincerely,

Michael Lawrence

Michael Lawrence, Head of Web Development, The International Birder

Part B: Project Proposal Plan

Project Summary

Problem:

The company is currently facing a significant challenge due to the recent departure of our Lead Bird Identifier (LBI), Mr. Fowl. His expertise in bird identification has been a crucial part of our operations, and finding a replacement with a similar skill set has proven difficult. The absence of a qualified individual to fill this role poses a risk to the accuracy and efficiency of our bird identification processes, which are vital to our publication's quality and reputation.

Without a suitable replacement, the company may experience delays and inaccuracies in bird classification, potentially leading to financial losses. The challenge is further compounded by the specialized nature of bird identification, which typically requires extensive training and experience, making it difficult to quickly onboard a new expert.

An innovative and cost-effective solution is needed to ensure accurate bird identification while minimizing the impact on our resources. This situation presents an opportunity to leverage advancements in technology and machine learning to create a sustainable and scalable solution that can be easily managed by our existing staff.

Solution:

The solution I propose is the development of a robust image classification machine-learning model that specializes in bird species identification. The model will be based on the DenseNet121 model(Huang, G., Liu, et al. (2017)) which itself is a type of 121 layer densely connected deep convolutional neural network (CNN) that relies on supervised learning and is specifically designed for image classification tasks.

To develop and train this model, we will leverage the Google Colab cloud-based machine learning development environment as well as the Python programming language and the TensorFlow machine learning libraries. This environment will give us access to all necessary libraries in a simple and easy-to-share format, as well as access to powerful hardware, like the NVIDIA A100 GPU, that would otherwise be outside of the budget for our small publication. The model will be trained on a dataset containing more than 84000 images of birds in 524 different categories.

As a front end for this ML model, a simple web application will be created using the Anvil.Works platform. Anvil is a platform that allows for creating full-stack web applications using Python and their in-house cloud-based editor. Anvil can provide cloud hosting of our application which further reduces the need for the purchase of expensive hardware. Anvil also provides user authentication which will provide security from various forms of attacks.

To summarize, the deliverables of this project will include a cloud-hosted web front end for an in-house written and trained image classification model that is specialized in taking an unlabeled/non-categorized image of a bird as input and outputting the species of that bird.

Data Summary

The data for training this ML model will be sourced from Kaggle.com (https://www.kaggle.com/datasets/gpiosenka/100-bird-species). This public domain (CC0 1.0 Universal License) dataset is split into three sets. A training set, a validation set, and a testing set. The set is comprised of 525 categories, 84635 training images, 2625 testing images, and 2625 validation images of size 224x224x3. All images are in ".jpg" format. The model will also accept ".jpeg," ".png," ".bmp," and ".gif" (Animated gifs are truncated to the first frame).

Within the dataset, there is one subdirectory/category that is present in all parent directories, that being a directory labeled "LOONEY BIRDS." This directory is a prank, or "Easter egg" within the dataset containing images of human beings. They will be removed programmatically before training to ensure the model is trained to look only for features pertaining to birds. All previous to further mention of the categories within the dataset will reference 524 categories.

All bird images in the dataset are cropped so that at least 50% of the 224x224x3 image contains a bird. This dataset will give a robust jumping-off point for a strong base of training.

A notable downside to this dataset is that it is heavily focused on male-gendered birds at an average rate of 80% male and 20% female. This could cause issues with training as male birds tend to have much brighter and different colors when compared to their female counterparts.

When the maintenance phase of the project begins, a new dataset can be formed in a similar fashion to further the training of the model, and to help with the unbalanced genders of the set, from sources like The Cornell Lab of Ornithology – Macaulay Library, Google Images, and iNaturalist.com. A program could be started to have our readers send in photos of birds they find in the wild.

As mentioned above, the dataset we are beginning with was created in the public domain (CC0 1.0 Universal License), so no ethical concerns need to be raised with the use of the set.

Implementation

The project will be implemented using the industry standard Agile methodology, allowing iterative development and continuous feedback. Agile will be well suited to this project due to its flexibility and ability to adapt quickly to unseen changes or roadblocks. Frequent communication will be had with stakeholders so progress can be monitored and concerns managed.

The phases of development will be as follows:

- Planning and Design
    - We will collaborate with stakeholders to define in further detail their requirements for the scope of the project.
    - We will design the architecture of the DenseNet121 model as well as wire-frame the design of the web front-end.
    - Accounts will be created and necessary purchases made for Google Colab as well as Anvil.works.
- Data Acquisition
    - The data will be downloaded and thoroughly analyzed for consistency and accuracy.
- Model Development
    - The model will be developed and trained using the training and validation datasets. This will take place in a Google Colab Notebook.
    - The model's training will be evaluated against the unseen-to-it, test dataset and any adjustments will be made to either, architecture, hyperparameters, training methodology, or all of the above.
- User Interface Development
    - Using Anvil.works we will develop the front-end application UI, focusing on simplicity and ease of use.
    - The functionality will be added for uploading an image, sending it to the Colab Notebook, utilizing the model to classify the results, sending those results back to the front end, and displaying results to the user.
- Testing
    - Test all aspects of the completed application to ensure features work as expected and scope requirements are met.
    - Conduct user acceptance training utilizing TIB staff members. Record their feedback.
- Documentation
    - Create user guides that are comprehensive and include instructions for using the application, best practices for achieving accurate predictions, and troubleshooting tips.
    - Create technical documentation that outlines the model's architecture, training process, integration with the user interface, and resource cost.

- Deployment
    - The Anvil.works application will be published on the web and made accessible to all necessary staff members.
    - Conduct a training seminar for the staff to bring everyone up to speed on the new application and ensure they are all familiar with its use and operation.
- Maintenance
    - Monitor use, accuracy, and functionality and provide support to users where needed.
    - Continue to retrain the model when new data becomes available

Timeline

Phase of Development	Sprint Duration	Projected start date (MM/DD/YY)	Anticipated end date (MM/DD/YY)
Planning and design	2 weeks	08/19/24	09/02/24
Data Acquisition	1 week	09/02/24	09/09/24
Model Development	2 weeks	09/09/24	09/30/24
UI Development	2 weeks	09/30/24	10/14/24
Testing	1 week	10/14/24	10/21/24
Documentation	1 week	10/28/24	11/04/24
Deployment	1 week	11/04/24	11/11/24
Maintenance	Ongoing	11/11/24	Ongoing

Evaluation Plan

The verification methods used at each stage of development will be listed below:

- Planning And Design
    - A thorough review of the gathered requirements and design will be conducted with stakeholders to ensure all needs are accurately met, the UI layout is approved, and the model design meets specifications.
- Data Acquisition
    - Verify the quality and integrity of the dataset by scanning for missing or corrupted files. Address issues as needed.
    - Create visualizations that show data is being correctly labeled and that the image augmentation pipeline is working as intended.
- Model Development
    - Generate a diagram of the model itself for inspection.
    - Create visualizations in the form of bar and line graphs that demonstrate the model's accuracy curves.
    - Create a confusion matrix to visualize the performance of the training and check for imbalances.
- User Interface Development
    - Conduct usability tests with non-technical staff members to ensure the UI is considered intuitive and easy to use.
    - Perform integration testing to verify that the front-end UI properly communicates with the back-end, ensuring image uploads and classification results are processed smoothly.
- Testing
    - Conduct functional testing to verify that all features work as intended and that error handling is in place and accurate.
    - Test multiple different image file types and images that are not within the scope of training and monitor results and errors.
    - After any modification or update, regression testing will be conducted to ensure that no prior functionality was disrupted.

Upon completion of the project, the validation process be conducted to ensure all deliverables meet the scope and the standards set by the stakeholders. These tests will include User Acceptance Testing (UAT), where key stakeholders and a sample of end-users interact with the application,

conduct classifications, and provide feedback. This UAT will ensure that the application is not only functional but also aligns with user expectations. The feedback will be collected and assessed, and any necessary adjustments will be made before final deployment.

Performance validation will be carried out to assess the application's reliability and effectiveness. Included will be in-depth accuracy testing of the DenseNet121 model utilizing the test dataset provided in the overarching dataset. This set was not used during training and will provide useful insights into the generalization of the ML model. In addition to the test dataset, images of birds will be gathered from Google Images and tested on the model to analyze its real-world proficiency.

All documentation will be reviewed and proofread to confirm its completeness and accuracy, ensuring end-users can operate the application and have information to troubleshoot where necessary.

Finally, the application will be checked for compliance with data protection regulations and company policies.

Hardware and Software Costs

Hardware

MacBook Pro for Development: $4,000

Software

Google Colab Pro+ Subscription: $50/month

Anvil.works Subscription: $100/month (Business Plan)

Kaggle.com: Free

Total Hardware and Software Costs: **$4,150 initial cost** + **$150/month ongoing**

Labor Time and Costs

Planning and Design: 40 hours

Cost: $100/hour * 40 hours = $4,000

Data Acquisition and Preprocessing: 20 hours

Cost: $100/hour * 20 hours = $2,000

Model Development: 120 hours

Cost: $100/hour * 120 hours = $12,000

User Interface Development: 80 hours

Cost: $100/hour * 80 hours = $8,000

Testing and Validation: 40 hours

Cost: $100/hour * 40 hours = $4,000

Documentation: 20 hours

Cost: $100/hour * 20 hours = $2,000

Deployment: 20 hours

Cost: $100/hour * 20 hours = $2,000

Maintenance and Updates: Estimated at 10 hours/month

Cost: $100/hour * 10 hours/month = $1,000/month

Total Estimated Labor Costs: **$34,000 initial cost** + **$1,000/month ongoing**

Part D: Post-implementation Report

Solution Summary

The International Birder faced a significant challenge following the departure of the Lead Bird Identifier (LBI), a crucial role for a bird-watching publication. The specialized expertise required for this position made it difficult to find a suitable replacement quickly, putting the efficiency of operations at risk. Without a replacement, the company was at risk of delays, and inaccuracies, threatening its reputation.

To address this issue, a machine learning solution was developed, centered around a DenseNet121 image classification model. This model was trained on a dataset of 524 bird species containing over 84,000 images, and integrated into a user-friendly web application hosted on Anvil.works. The application allows staff members to upload images of birds and receive accurate species classifications without needing specialized expertise. The solution leverages cloud-based resources from Google Colab for model training and is designed to be scalable, efficient, and easy to use.

The application directly addresses the problem by automating the bird identification process, replacing the need for a human Lead Bird Identifier. With its robust training, the ML solution ensures that bird species are classified correctly. The intuitive user interface allows any staff member, regardless of their technical background, to use the system effectively. This not only solves the immediate challenge of identifying bird species but also reduces operational costs by eliminating the need for specialized human expertise.

Data Summary

The data used in the project was a public domain (CC0 1.0 Universal License) set of over 84,000 images divided into 524 classes. It was pulled from Kaggle.com (https://www.kaggle.com/datasets/gpiosenka/100-bird-species). The dataset was comprised of three smaller sets, a train set, a valid set, and a test set. Using TensorFlow(TF) Keras's image_dataset_from_directory, these directories, that followed the structure (/train/CLASSIFICATION/001.jpg), were created into TensorFlow datasets. The class names were pulled from the training dataset and saved to the variable "class_names" for ease of access later.

The training dataset was split into two datasets, one of the raw training data, and one with data augmentation applied to help prevent under and overfitting. The augmentation methods chosen were simple, practical methods. All data was shuffled during dataset creation as well as at the start of each epoch of training. In an effort to maintain efficiency during training the "train," and "valid," datasets were cached and prefetched. The test dataset was only prefetched.

Below is an example of the code used to create the datasets as well as images pulled from the created datasets.

raw_training_dataset = image_dataset_from_directory(

```
path_to_training_data,
image_size=image_size,
batch_size=batch_size,
label_mode='categorical',
seed=seed,
shuffle=True
)

class_names = raw_training_dataset.class_names

data_augmentation_pipeline = Sequential([
RandomFlip("horizontal_and_vertical"),
RandomRotation(0.2),
RandomZoom(0.2),
RandomContrast(0.2)
])

training_dataset = raw_training_dataset.map(
lambda x, y: (data_augmentation_pipeline(x, training=True), y)
).cache().prefetch(buffer_size=tf.data.AUTOTUNE)

valid_dataset = image_dataset_from_directory(
path_to_validating_data,
image_size=image_size,
batch_size=batch_size,
label_mode='categorical',
seed=seed,
shuffle=True
).cache().prefetch(buffer_size=tf.data.AUTOTUNE)

test_dataset = image_dataset_from_directory(
path_to_test_data,
image_size=image_size,
batch_size=batch_size,
label_mode='categorical',
seed=seed,
shuffle=True
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Raw Training Dataset:

Augmented Training Dataset:

Validation Dataset:

Test Dataset:

Machine Learning

The method chosen for this project was the creation of a DenseNet121 which was first written about in "Densely Connected Convolutional Networks"(Huang, G., Liu, et al. (2017)). Utilizing densely connected convolutional layers, each layer receives input from all proceeding layers, which allows for efficient feature propagation and reduces the problem of vanishing gradients. The model was chosen due to its specialized nature for high-accuracy image classification tasks.

For the development of this algorithm, the open-source machine learning library TensorFlow was used for ease of development and training. The model architecture was defined by stacking dense blocks, transition layers, and convolutional layers. The model starts with an initial convolutional layer to extract basic features and downsample the input. It then passes through a series of dense blocks where each layer connects to every other layer in a feed-forward fashion, which helps in efficient feature reuse. Between dense blocks, transition layers downsample the feature maps and reduce the number of filters. After the last dense block, the model applies global average pooling to reduce the spatial dimensions, followed by a dropout layer for regularization. Finally, the model ends with a dense layer that outputs the classification probabilities for each class using softmax activation.

The model was trained by taking the augmented “train_dataset” and validating it against the unaugmented “valid” dataset. Utilizing Google Colab’s cloud-based GPU resources, in particular the powerful NVIDIA A100 GPU, training time was around one hour and twenty minutes for 25 epochs. During training callbacks were made which adjusted the model’s learning rate to minimize the error between its predictions and maintain a steady climb in validation accuracy. Early stopping and checkpointing techniques were employed to prevent overfitting and to save the best-performing version of the model. This resulted in an approximate 79% validation accuracy. And when evaluated against the unseen “test” dataset, the model reached an accuracy of over 81%.

Validation

Before training a small subset of each dataset was visualized to show that the class labels were accurately displayed. This includes showing the effects of the data augmentation that was applied to the training dataset to assist the model with generalization.

```
def visualize_dataset(dataset, class_names, num_images=4):

plt.figure(figsize=(10, 10))
for images, and labels in the dataset.take(1):
for i in range(num_images):
ax = plt.subplot(1, num_images, i + 1)
plt.imshow(images[i].numpy().astype("uint8"))
plt.title(class_names[labels[i].numpy().argmax()])
plt.axis("off")
```

```
plt.show()

print("Raw Training Dataset:")
visualize_dataset(raw_training_dataset, class_names)

print("Augmented Training Dataset:")
visualize_dataset(training_dataset, class_names)

print("Validation Dataset:")
visualize_dataset(valid_dataset, class_names)

print("Test Dataset:")
visualize_dataset(test_dataset, class_names)
```

Augmented Training Dataset:

Validation Dataset:

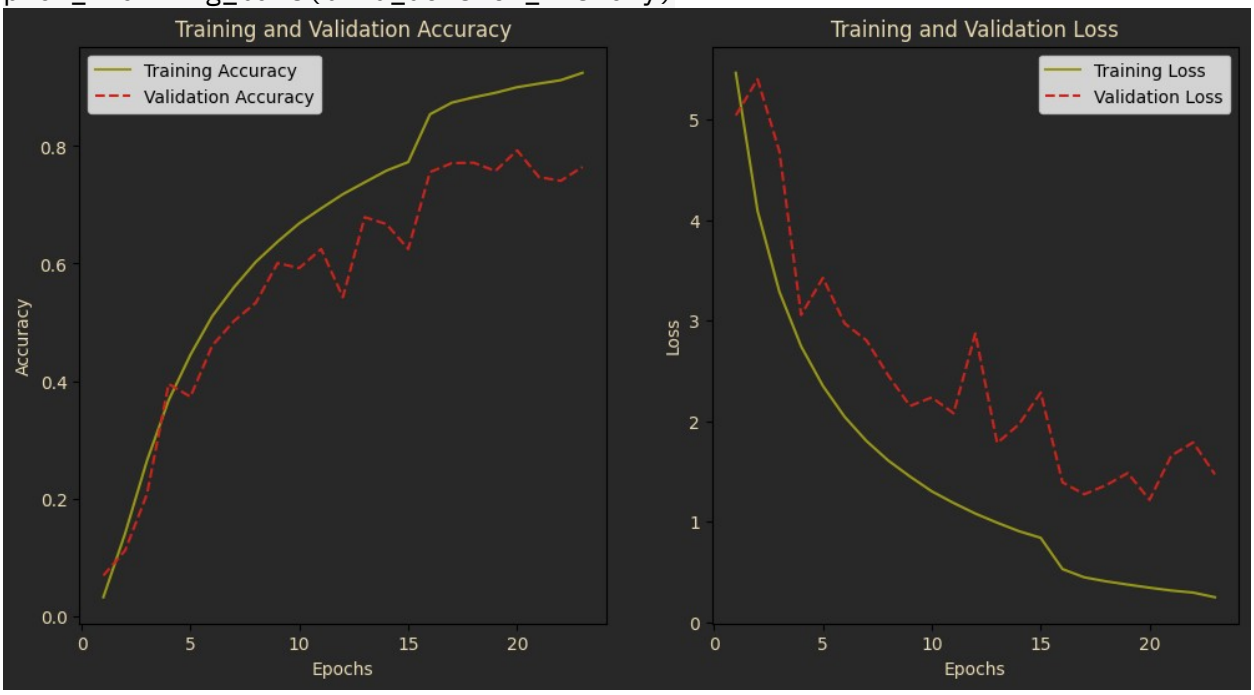
Test Dataset:

During training the model was shown images from the augmented dataset to create predictions and those predictions were checked against the validation dataset. The model went through 25 epochs and peaked at a validation accuracy of 79.27%. The training and validation accuracy were displayed as bar and line graphs for ease of analysis.

```
bird_densnet_history = bird_densenet.fit(

training_dataset,
epochs=25,
callbacks=[model_checkpoint, reduce_lr, early_stopping],
validation_data=valid_dataset,
batch_size=batch_size,
shuffle=True
)


def plot_training_lines(history):

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.figure(figsize=(12, 6), facecolor='#282828')

plt.subplot(1, 2, 1, facecolor='#282828')
plt.plot(epochs, acc, color='#98971a', label='Training Accuracy')
plt.plot(epochs, val_acc, color='#cc241d', linestyle='--',
label='Validation Accuracy')
plt.title('Training and Validation Accuracy', color='#ebdbb2')
plt.xlabel('Epochs', color='#ebdbb2')
plt.ylabel('Accuracy', color='#ebdbb2')
plt.xticks(color='#ebdbb2')
plt.yticks(color='#ebdbb2')
```

```
plt.legend()

plt.subplot(1, 2, 2, facecolor='#282828')
plt.plot(epochs, loss, color='#98971a', label='Training Loss')
plt.plot(epochs, val_loss, color='#cc241d', linestyle='--',
label='Validation Loss')
plt.title('Training and Validation Loss', color='#ebdbb2')
plt.xlabel('Epochs', color='#ebdbb2')
plt.ylabel('Loss', color='#ebdbb2')
plt.xticks(color='#ebdbb2')
plt.yticks(color='#ebdbb2')
plt.legend()

plt.show()

def plot_training_bars(history):
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)
width = 0.35

plt.figure(figsize=(12, 6), facecolor='#282828')

plt.subplot(1, 2, 1, facecolor='#282828')
plt.bar(np.array(epochs) - width/2, acc, width, label='Training Accuracy',
color='#98971a')
plt.bar(np.array(epochs) + width/2, val_acc, width, label='Validation
Accuracy', color='#cc241d')
plt.title('Training and Validation Accuracy', color='#ebdbb2')
plt.xlabel('Epochs', color='#ebdbb2')
plt.ylabel('Accuracy', color='#ebdbb2')
plt.xticks(epochs, color='#ebdbb2')
plt.yticks(color='#ebdbb2')
plt.legend()

plt.subplot(1, 2, 2, facecolor='#282828')
plt.bar(np.array(epochs) - width/2, loss, width, label='Training Loss',
color='#98971a')
plt.bar(np.array(epochs) + width/2, val_loss, width, label='Validation
Loss', color='#cc241d')
plt.title('Training and Validation Loss', color='#ebdbb2')
plt.xlabel('Epochs', color='#ebdbb2')
```

```
plt.ylabel('Loss', color='#ebdbb2')
plt.xticks(epochs, color='#ebdbb2')
plt.yticks(color='#ebdbb2')
plt.legend()

plt.show()

plot_training_lines(bird_densnet_history)
plot_training_bars(bird_densnet_history)
```



After training, the model was evaluated on a holdout test set that was not used during the training process. The model's accuracy was measured by comparing its predictions against the actual species labels in this test set. Accuracy was calculated as the percentage of correct predictions out of the total number of predictions.

```
trained_model =
load_model('/content/models/bird_classification_densenet.keras')
```

```
test_loss, test_accuracy = trained_model.evaluate(test_dataset)
```

```
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
41/41 ━━━━━━━━━━━━━━━━━━━━ 27s 278ms/step - accuracy: 0.8117 - loss:
1.0371
Test Loss: 1.0263651609420776, Test Accuracy: 0.8152672052383423
```

As shown in the code snippets above, the model shows a greater than 80% accuracy when evaluated against the test data that was not used to train the model.

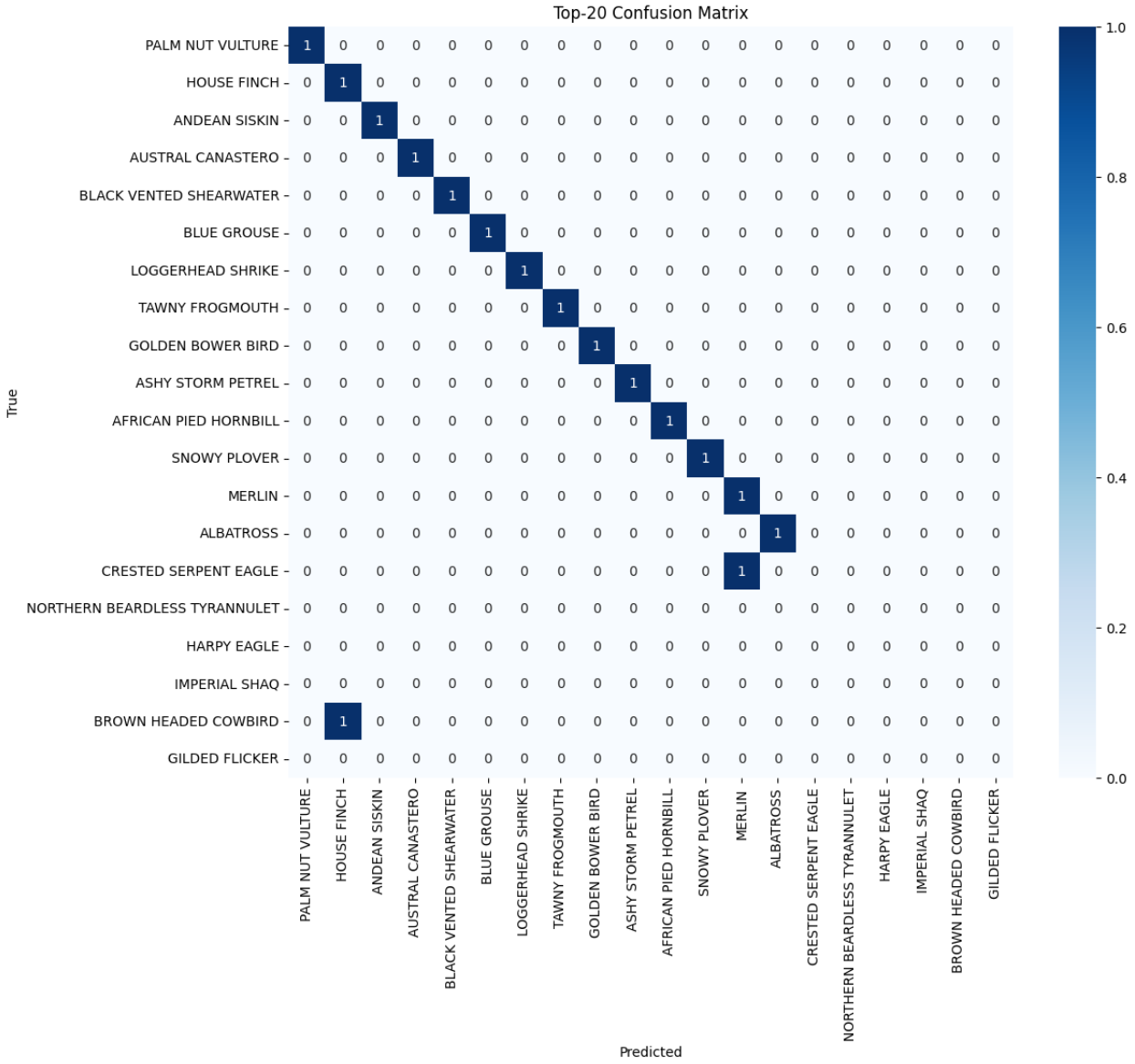
Also, a confusion matrix was created to show the Top-20 miss-classifications when evaluated against the validation dataset.

```
trained_model.evaluate(valid_dataset)
```

```
true_labels = np.concatenate([y for x, y in valid_dataset], axis=0)
predictions = trained_model.predict(valid_dataset)
predicted_labels = np.argmax(predictions, axis=-1)
```

```
true_labels = np.argmax(true_labels, axis=-1)

cm = confusion_matrix(true_labels, predicted_labels)

misclassifications = np.sum(cm, axis=1) - np.diag(cm)

top_n = 20
top_n_indices = np.argsort(misclassifications)[-top_n:]

cm_top_n = cm[top_n_indices][:, top_n_indices]
class_names_top_n = [class_names[i] for i in top_n_indices]

plt.figure(figsize=(12, 10))
sns.heatmap(cm_top_n, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names_top_n, yticklabels=class_names_top_n)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title(f'Top-{top_n} Confusion Matrix')
plt.show()
```

```
41/41 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.7890 - loss: 1.2286
41/41 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step
```



As well as using the `.evaluate()` method, a script was written to show, on a small subset of photos from the test set, how well the model makes predictions.

```
image_paths = [
'/content/test/BALD EAGLE/3.jpg',
'/content/test/AMERICAN FLAMINGO/4.jpg',
'/content/test/COMMON FIRECREST/2.jpg',
'/content/test/CALIFORNIA GULL/4.jpg',
'/content/test/CROW/5.jpg'

]
```

```
images = np.vstack([preprocess_image(img_path, target_size) for img_path
in image_paths])
predictions = trained_model.predict(images)
decoded_preds = decode_predictions(predictions, class_names, top=3)

for img_path, decoded in zip(image_paths, decoded_preds):
print(f"Image: {img_path}")
for class_name, probability in decoded:
print(f" Predicted class: {class_name}, Probability: {probability:.4f}")

for img_path, decoded in zip(image_paths, decoded_preds):
classes = [class_name for class_name, _ in decoded]
probabilities = [probability for _, probability in decoded]

plt.figure(figsize=(8, 4))
plt.barh(classes, probabilities, color='#98971a')
plt.xlabel('Probability')
plt.title(f"Predictions for {img_path.split('/')[-2]}")
plt.gca().invert_yaxis()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Image: /content/test/BALD EAGLE/3.jpg
  Predicted class: BALD EAGLE, Probability: 0.8917
  Predicted class: WHITE TAILED TROPIC, Probability: 0.0946
  Predicted class: ALBATROSS, Probability: 0.0091
Image: /content/test/AMERICAN FLAMINGO/4.jpg
  Predicted class: AMERICAN FLAMINGO, Probability: 0.9993
  Predicted class: SCARLET IBIS, Probability: 0.0007
  Predicted class: BALD IBIS, Probability: 0.0000
Image: /content/test/COMMON FIRECREST/2.jpg
  Predicted class: COMMON FIRECREST, Probability: 1.0000
  Predicted class: D-ARNAUDS BARBET, Probability: 0.0000
  Predicted class: CAPE MAY WARBLER, Probability: 0.0000
Image: /content/test/CALIFORNIA GULL/4.jpg
  Predicted class: CALIFORNIA GULL, Probability: 0.9633
  Predicted class: NORTHERN FULMAR, Probability: 0.0356
  Predicted class: FAIRY TERN, Probability: 0.0006
Image: /content/test/CROW/5.jpg
  Predicted class: HAMERKOP, Probability: 0.6567
  Predicted class: BLACK VULTURE, Probability: 0.1995
  Predicted class: CROW, Probability: 0.0333
```

User Guide

Here will be presented a guide for testing the application. Files/directories provided are:

- The “External_test_images” Directory and its contents of 4 images for testing.
- The “Model” directory which contains “bird_classification_densenet.keras”
- “ApplicationMikeLawrenceCapstone.ipynb” Is a local copy of the Google Colab Notebook used in this guide. The code is copy-pasted from the next file listed. This is for brevity and ease of evaluation.
- “MainMikeLawrenceCapstone.ipynb” Is the main development Google Colab Notebook
- The “capstone-bird-classifier-densenet” Directory is a clone of the source code for the Anvil.Works Web Application. The public repository can be found at: “https://github.com/MikeLawCodes/capstone-bird-classifier-densenet.git”

1. Open an up-to-date web browser such as Google Chrome, Mozilla Firefox, Microsoft Edge
2. Open this link to the Google Colab Notebook: “https://colab.research.google.com/drive/1FRDPpVFKE0x7UjzcfApw9V06TOpXVZU5?usp=sharing” Or upload the “ApplicationMikeLawrenceCapstone.ipynb” file provided.
3. Connect to a runtime of your choice. The free CPU runtime will work, though other runtimes will increase performance.
4. Each cell is labeled with a comment indicating its number for reference here. Run cells 1 through 6.
5. When cell 6 finishes running, a warning will appear. Click cancel, the runtime does not need to be restarted.
6. Run cells 7 through 12.
7. After cell 12 completes, navigate to the sidebar and click on the file icon to open the file browser
8. In the top left of the file browser will be an upload file button. Click this button and navigate to where the `bird_classification_densenet.keras` file is located on your computer and select it for upload. This is the model that has been trained.
9. When the file is finished being uploaded, run cells 13 through 16. Cell 16 will run predictions on 5 images from the “test” dataset that was created earlier.

10. Run cells 17 and 18.
    1. Cell 17 creates a function that can be called by the Anvil front-end that takes an uploaded image from the web page, and returns a prediction to be displayed on the web page.
    2. Cell 18 calls a function from the Anvil API to keep the Colab notebook running while using it as a back-end for the application.
11. Click the link in the text box above cell 17.
12. When presented with a redirect warning, click on the https://educated-oval-iberian-lynx.anvil.app link. This will navigate to the Anvil web application. Ensure this is in a new tab so that the Colab file continues to run.
13. You will now be presented with two buttons. Click on the “Upload Image” button.
14. Navigate to one of the 4 image files within the “External_test_images” directory. These are images taken from Google Images and were not part of the original dataset.
15. Upload the chosen file. There will be a display of the uploaded file between the two buttons which will confirm it was uploaded.
16. Click the "Classify" button and wait. A result will appear in the bottom outlined box informing the user of the classification of the bird. Each image has its classification as its file name. Compare the result to the file name to confirm the result.

Reference Page

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).