SWINBURNE UNIVERSITY OF TECHNOLOGY

School of Science, Computing and Engineering Technologies

COS30082:

Applied Machine Learning

Assignment 1 Report

Submitted by:

Vu Thanh Phong

Lecturer:
Mr. Minh Hoang

Date of submission:
29/6/2025

I. Introduction:

This assignment focuses on a multi-class bird image classification task, where the goal is to accurately categorize bird images into one of 200 distinct species. The project involves key steps such as data preprocessing, model training, transfer learning, and performance evaluation to effectively tackle the challenges of fine-grained visual recognition.

II. Dataset:

In this Bird Image Classification assignment, the dataset CUB-200-2011 , available on Kaggle, was chosen for training purposes. This dataset comprises 11,788 images spanning 200 different bird species, making it a suitable choice for fine-grained image classification tasks.

III. Dataset Loading:

```
[3]  import kagglehub

     # Download latest version
     path = kagglehub.dataset_download("wenewone/cub2002011")

     print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/cub2002011
```

Figure 1: Download CUB-200-2011 dataset from Kaggle

This code block demonstrates how to download the CUB-200-2011 bird dataset from Kaggle using the kagglehub library. By specifying the dataset identifier "wenewone/cub2002011," the script fetches the latest version of the dataset and stores the local path, which is then printed. This setup is essential for accessing the dataset files for further preprocessing and model training.

```
[4]  dataset_dir = "/kaggle/input/cub2002011"
     img_dir = os.path.join(dataset_dir, "CUB_200_2011/images")

     # Load images.txt (image_id and file path)
     images_df = pd.read_csv(os.path.join(dataset_dir, "CUB_200_2011/images.txt"), sep=" ", names=["img_id", "filename"])
     images_df["filename_only"] = images_df["filename"].apply(lambda x: os.path.basename(x))

     # Load image_class_labels.txt
     labels_df = pd.read_csv(os.path.join(dataset_dir, "CUB_200_2011/image_class_labels.txt"), sep=" ", names=["img_id", "class_label"])
     labels_df["class_label"] -= 1  # Make 0-based

     # Load classes.txt (class_label to class_name)
     classes_df = pd.read_csv(os.path.join(dataset_dir, "CUB_200_2011/classes.txt"), sep=" ", names=["class_label", "raw_class_name"])
     classes_df["class_label"] -= 1
     classes_df["class_name"] = classes_df["raw_class_name"].apply(lambda x: x.split('.', 1)[-1])

     # Merge all
     df = images_df.merge(labels_df, on="img_id").merge(classes_df, on="class_label")
     df = df[["filename_only", "class_name", "class_label"]]
     df.rename(columns={"filename_only": "filename"}, inplace=True)

     # Convert class_label to string for Keras
     df["class_label"] = df["class_label"].astype(str)
```

Figure 2: CUB-200-2011 data preprocessing

The block of code above handles preprocessing of the dataset metadata. It loads three text files: images.txt (mapping image IDs to file paths), image_class_labels.txt (linking image IDs to class labels), and classes.txt (mapping class labels to bird species names). These are read into pandas DataFrames and merged into a single DataFrame df, which includes image filenames and their corresponding class labels. The labels are also adjusted to be zero-indexed and converted to strings for compatibility with Keras-based models.

```
[7]  # Split into train (70%) and temp (30%)
     train_df, temp_df = train_test_split(df, test_size=0.3, stratify=df["class_label"], random_state=42)

     # Split temp into validation (15%) and test (15%)
     val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df["class_label"], random_state=42)

     print(f"Train size: {len(train_df)}, Validation size: {len(val_df)}, Test size: {len(test_df)}")

     Train size: 8251, Validation size: 1768, Test size: 1769
```

Figure 3: Splitting train, validation and test dataset

The code above uses train_test_split() function to get the train, validation and test dataset.

```
[29] # Data augmentation for training
     train_datagen = ImageDataGenerator(
         rescale=1./255,
         shear_range=0.3,
         zoom_range=0.3,
         rotation_range=40,
         width_shift_range=0.3,
         height_shift_range=0.3,
         horizontal_flip=True,
         fill_mode='nearest'
     )

     # No augmentation for val/test
     val_test_datagen = ImageDataGenerator(rescale=1./255)
```

Figure 4: Applying augmentation for training images

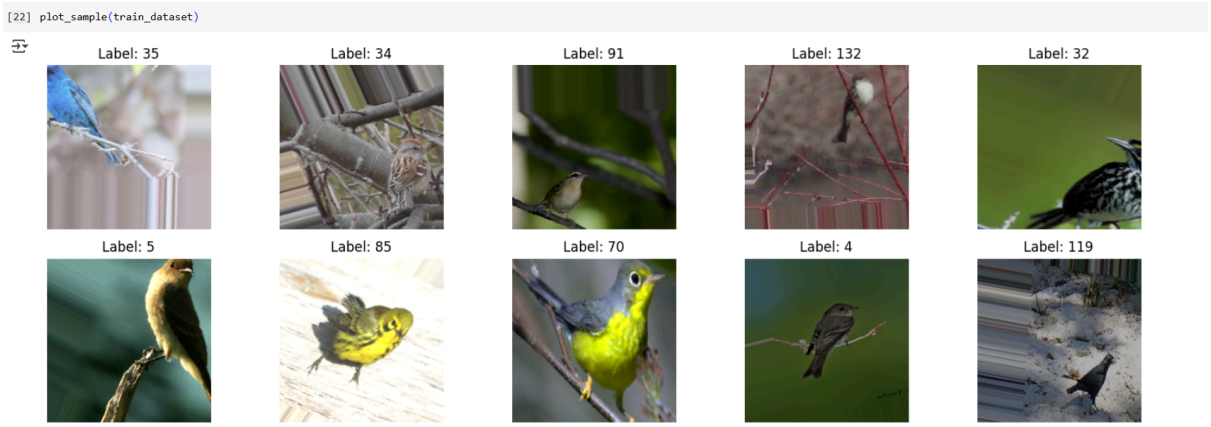

Figure 5: Plot 10 samples from the training dataset

IV. ResNet50:

ResNet50 is a deep convolutional neural network consisting of 50 layers, designed to solve the degradation problem in deep networks using residual connections. These connections allow the network to learn identity mappings more easily, enabling very deep architectures to be trained effectively. ResNet50, pre-trained on the ImageNet dataset, is widely used in transfer learning tasks due to its powerful feature extraction capabilities, especially for image classification problems.

```
[23] def build_resnet50_model(input_shape=(224, 224, 3), num_classes=200, freeze_base=True):
         base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)

         # Freeze the base model's layers
         if freeze_base:
             for layer in base_model.layers:
                 layer.trainable = False
         else:
             for layer in base_model.layers[-20:]:
                 layer.trainable = True

         x = base_model.output
         x = GlobalAveragePooling2D()(x)
         x = Dense(1024, activation='relu', kernel_regularizer=l2(0.001))(x)
         x = BatchNormalization()(x)
         x = Dropout(0.5)(x)
         predictions = Dense(num_classes, activation = 'softmax')(x)
         model = Model(inputs = base_model.input, outputs = predictions)

         return model
```

Figure 6: Function to build ResNet-50

The function build_resnet50_model builds a custom image classification model using the ResNet50 architecture as a base. It takes the input image shape, number of classes, and an option to freeze the base layers. If freeze_base=True, all layers in the ResNet50 base are frozen (non-trainable). Otherwise, only the last 20 layers are set as trainable (fine-tuned). After the base model, a global average pooling layer is added to reduce spatial dimensions, followed by a dense (fully connected) layer with ReLU activation and L2 regularization. Batch normalization and dropout are applied to prevent overfitting. Finally, a softmax output layer is used for multi-class classification, and the full model is returned.

V. Model training:

```
[ ]  history1 = model.fit(
         train_dataset,
         steps_per_epoch=train_dataset.samples // batch_size,
         epochs=100,
         validation_data=val_dataset,
         validation_steps=val_dataset.samples // batch_size,
         verbose=1,
         callbacks=[early_stop, reduce_lr]
     )
```

Figure 7: Function to build ResNet-50

This code trains the model using the Keras fit() function for 100 epochs with specified training and validation datasets. It calculates the number of steps

per epoch based on the dataset size and batch size. Two callbacks are used: EarlyStopping to stop training when validation performance stops improving, and ReduceLROnPlateau to lower the learning rate when progress stalls. The training process is logged with verbose = 1, and the training history is stored in history1 for later evaluation.

VI. Model Evaluation:

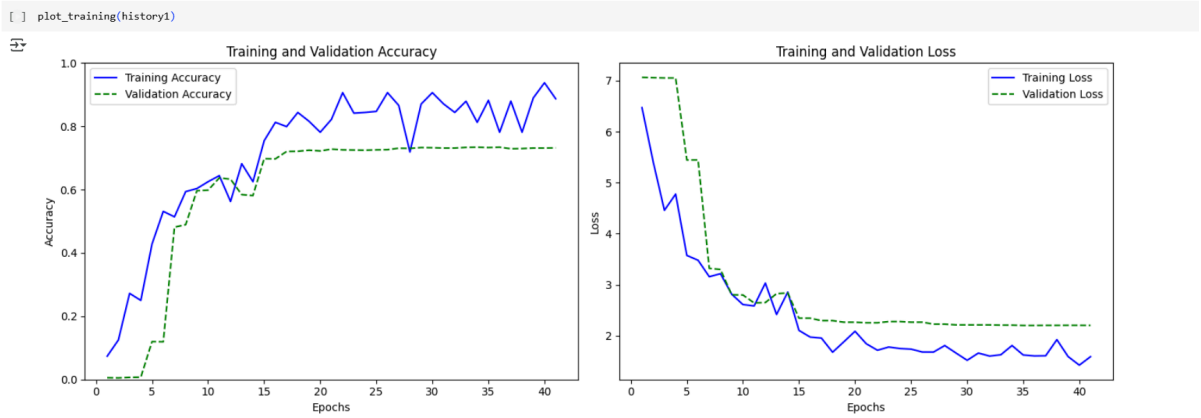

Figure 8: Model training evaluation

Training performance:

- Loss Reduction: The training loss consistently decreased from approximately 6.48 to 0.61 over 40 epochs, reflecting the model's ability to learn and minimize prediction errors on the training data.
- Accuracy Increase: The training accuracy improved significantly from around 7.64% at the beginning to approximately 95.62% by epoch 40. This sharp rise demonstrates effective learning and strong classification performance on the training set.

Validation performance:

- Loss Trends: The validation loss dropped rapidly in the early epochs, reaching around 2.14 by epoch 15. It then remained relatively stable, indicating consistent performance on the validation data.
- Accuracy Improvement: Validation accuracy increased steadily from 0.0% at the start to a peak of approximately 70.5% by the end of training. This shows that the model achieved a reasonable level of generalization to unseen data.

```
[ ]  y_pred = model.predict(test_dataset)
     y_true = test_dataset.classes
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 16s 199ms/step
```

```
[ ]  y_pred_labels = np.argmax(y_pred, axis=1)
```

```
[ ]  top1_accuracy = accuracy_score(y_true, y_pred_labels)
     print(f"Top-1 Accuracy: {top1_accuracy * 100:.2f}%")
```

```
Top-1 Accuracy: 74.73%
```

Figure 9: Performing test on test_dataset and get top-1 accuracy

This code calculates the Top-1 Accuracy, which measures the percentage of test samples where the predicted label exactly matches the true label. Using accuracy_score() from sklearn.metrics, it compares y_pred_labels with y_true and prints the result. The model achieved a Top-1 Accuracy of 74.73%, indicating that it correctly predicted the exact class for nearly three-quarters of the test images.

VII. Model Testing:

```
[ ]  for i, name in enumerate(predicted_class_names):
         print(f"Image {i} ({test_filenames[i]}): Predicted class: {name}")
```

```
Image 0 (Acadian_Flycatcher.jpg): Predicted class: 36
Image 1 (American_Crow.jpg): Predicted class: 26
Image 2 (Anna_Hummingbird.jpg): Predicted class: 67
Image 3 (Baird_Sparrow.jpg): Predicted class: 112
Image 4 (Bank_Swallow.jpg): Predicted class: 134
Image 5 (Belted_Kingfisher.jpg): Predicted class: 78
Image 6 (Black_and_white_Warbler.jpg): Predicted class: 158
Image 7 (Black_footed_Albatross.jpg): Predicted class: 7
Image 8 (Black_throated_Sparrow.jpg): Predicted class: 113
Image 9 (Blue_Grosbeak.jpg): Predicted class: 53
Image 10 (Least_Auklet.jpg): Predicted class: 6
Image 11 (Bohemian_Waxwing.jpg): Predicted class: 185
Image 12 (Pine_Warbler.jpg): Predicted class: 155
Image 13 (House_Sparrow.jpg): Predicted class: 117
```

Figure 9: Performing test on random bird images

Out of 14 randomly selected bird images, the model correctly predicted the exact class for 7 images, demonstrating a good level of accuracy on unseen data. For several of the incorrect predictions, such as Image 1, Image 9, and Image 10, the predicted class was numerically close to the actual class, indicating that the model is often able to identify visually similar bird species even if the prediction isn't perfect. This suggests the model has learned useful features for fine-grained classification, though there is still room for improvement in distinguishing between closely related classes.

VIII. Conclusion:

The model achieved a Top-1 accuracy of 74.73%, correctly classifying 7 out of 14 randomly selected test images. Several misclassifications were close to the actual labels, indicating the model's strong feature extraction capabilities. Overall, the results demonstrate that transfer learning with ResNet50 is effective for fine-grained bird classification, though further improvements could be made through fine-tuning, data augmentation, or ensembling techniques.