FINE GRAINED CLASSIFICATION

Data Loading and Pre-Processing

Instructions for Loading data:

1. This image file is unzipped to a folder (**"/content/Deep_Learning_coursework/bird"**) within the content drive - This makes the process faster than unzipping in the drive
2. The images will now be split to Test(50%), Train(40%) and Validation(10%) using stratified split. The images are moved to the folders according to this directory:

```
/content
└── Deep_Learning_coursework
......└── bird
..........└── CUB_200_2011
..................└── images
..................└── test_images
..................└── val_images
..................└── train_images
```

4. The images are loaded using the image_dataset_from_directory which is an inbuilt pre-processing function from Keras.
5. While extracting the data, the size of the image (height = 256, width = 256) and batch size (32) are adjusted

Implemented approach for fine-grained bird species classification:

- Transfer Learning tried with EfficientNetB4, ResNet, fixed on EfficientNetV2-S
- Fine Tuning done by unfreezing the last 50 layers so the model can learn from the dataset
- Model Accuracy currently at 90.62%, with precision at 93.52 and recall at 90.28
- Confusion Matrix shows that most images are perfectly classified

Other Approches which can improve model performance:

- DenseNet121, ViT, GANs may produce identical or improved results since they are perform well with datasets of a small size
- Ensemble or model stacking
- Model has to be implemented with the larger dataset
- Other optimizers like AdaGrad, SGD (Stochastic Gradient Descent), AdamW
- Hyperparameter tuning for optimizing learning rate, batch size, dropout rate

Code Implementation

Data Preprocessing:

• Images resized to 256×256, batch size 32
• Applied data augmentation (flipping, rotation, zoom, contrast adjustment)
• Used AUTOTUNE to prefetch batches for efficient training
• One-hot encoding applied to class labels (20 classes in total)

Model Selection & Training:
• Used EfficientNet, pre-trained on ImageNet
• Added GlobalAveragePooling2D, Dropout (0.3), and Dense (softmax) output layer for 20 classes
• Compiled with Adam optimizer and categorical cross-entropy loss
• Fine-tuned by unfreezing the last 50 layers

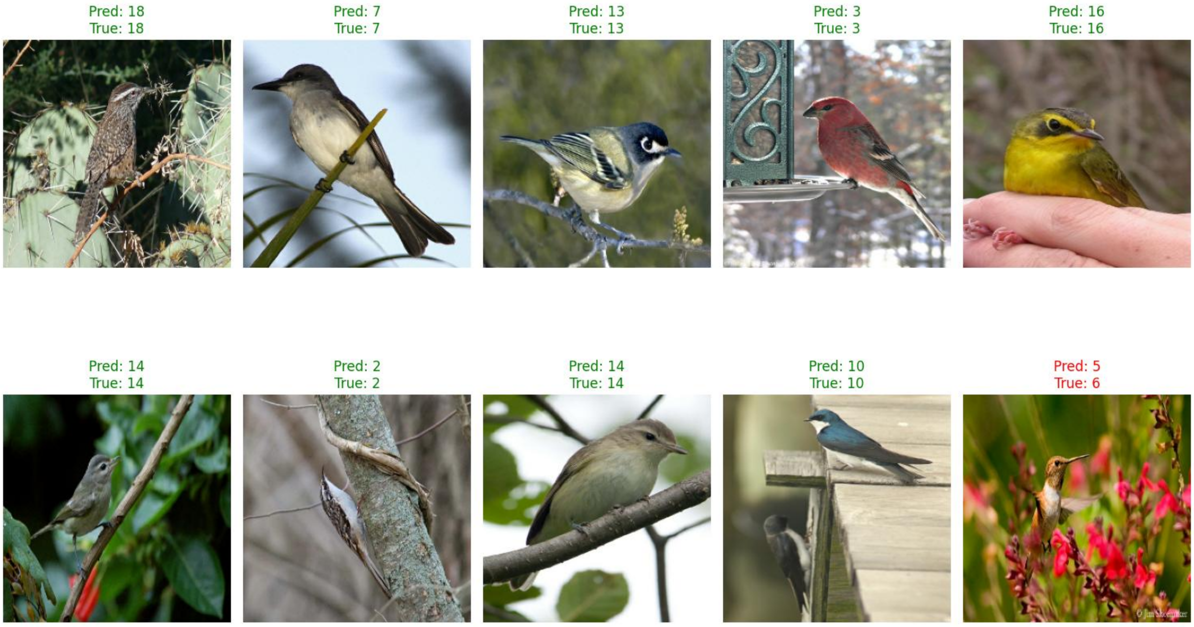
Training the Model:
• Trained on train_ds, validated on val_ds
• Metrics tracked: accuracy

Testing Data:
• Test images loaded and preprocessed similarly to training data
• Model predictions displayed with actual images
• Accuracy, Precision and Recall used for performance analysis
• Confusion matrix plotted for performance analysis

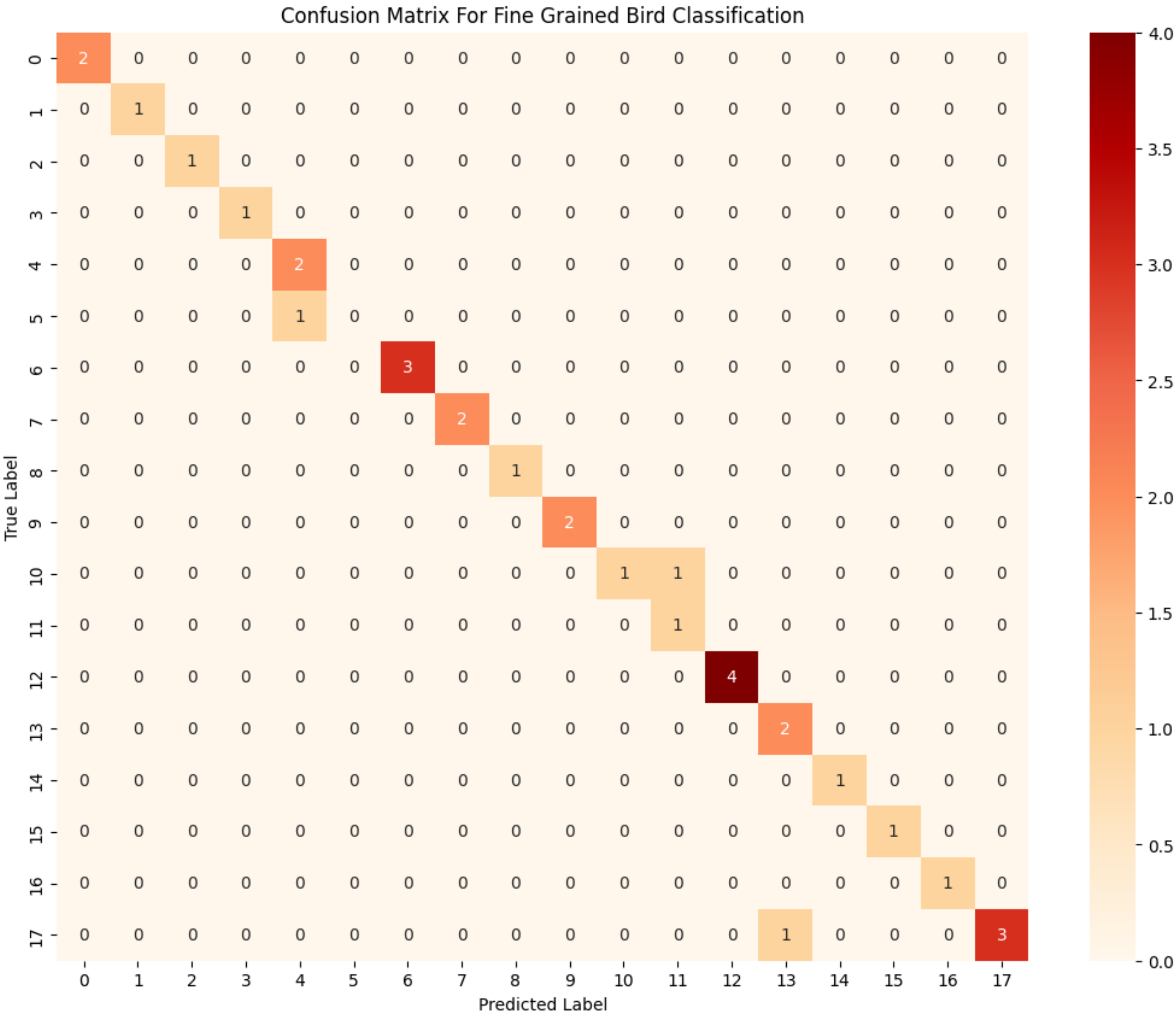
Performance Improvement:

1. Hyperparameter tuning (highest feasibility, major impact).
2. Ensemble/Stacking (if computational resources allow).
3. Trying AdamW/SGD with tuning (helps convergence and generalization).
4. DenseNet121, ViT, GANs (ViT is great but may require larger datasets or fine-tuning).
5. Larger Dataset (ideal but dataset availability is a constraint).

Model Results:



Pred: 18
True: 18
Pred: 7
True: 7
Pred: 13
True: 13
Pred: 3
True: 3
Pred: 16
True: 16
Pred: 14
True: 14
Pred: 2
True: 2
Pred: 14
True: 14
Pred: 10
True: 10
Pred: 5
True: 6



Confusion Matrix For Fine Grained Bird Classification
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
0: 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1: 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4: 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0
5: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
6: 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0
7: 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
8: 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
9: 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
10: 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
11: 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
12: 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0
13: 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0
14: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
15: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
16: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
17: 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 3
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

Performance Metrics

```python
# Computing and printing performance metrics
accuracy = accuracy_score(true_classes, predicted_classes)
print(f"Accuracy: {accuracy:.4f}")


precision = precision_score(true_classes, predicted_classes, average='macro',zero_division = 1)
print(f"Precision: {precision:.4f}")


recall = recall_score(true_classes, predicted_classes, average='macro')
print(f"Recall: {recall:.4f}")
```

```
Accuracy: 0.9062
Precision: 0.9352
Recall: 0.9028
```