Bird Classification Project

Prepared by – Aakash Maskara

Overview

This report documents the development, experimentation, evaluation, and reflection for two deep learning models designed for image classification tasks.

The assignment focused on:

1. Birds vs Squirrels Classifier : A 3-class classification problem
2. Birder (358-class Bird Type Classifier) : A large-scale fine-grained classification problem

The models were built using TensorFlow, with a shared preprocessing pipeline, standard evaluation metrics, and model saving conventions.

Dataset Insights

- Birds vs Squirrels Dataset :
    - Task : Binary species classification among birds, squirrels, and other animals
    - 3-class classification
    - Training : 1152 images (TFRecord : birds-vs-squirrels-train.tfrecords)
    - Validation : 288 images (TFRecord : birds-vs-squirrels-validation.tf records)
    - Balanced and relatively small dataset, useful for fast experimentation and baseline models
- Birder 358-class :
    - Task : Fine-grained classification of bird species across 358 unique classes
    - 358 distinct bird classes

    - Training : 7160 images (20 samples per class), (TFRecord : birds-20-eachOf-358.tfrecords)
    - Validation : 3580 images (10 samples per class), (TFRecord : birds-10-eachOf-358.tfrecords)
    - Large output space and small per-class sample size made it a difficult task

Data Insights

- The 358-class dataset is highly imbalanced per class with only 30 total images per class, making it prone to overfitting
- Birds vs Squirrels was better balanced and easier to generalize
- All data was pipelined using td.data.TFRecordDataset, with map, batch, prefetch, and repeat operations for performance

Observation

- Large models significantly increased training time, especially when base layers were unfrozen
- Overfitting was more visible in the 358-class task due to limited training data per class
- Some epochs showed higher validation accuracy than training accuracy, which was attributed to batch-wise variability and frozen base layers

Preprocessing & Feature Engineering

A shared preprocessing pipleline (preprocessDefinition.py) was used :

- Resized all images to 299×299 to maintain compatibility with models like Xception and InceptionV3
- Used tf.keras.applications.xception.preprocess_input() for normalization
- Data Augmentation (only for 358-class task) :
    - Random flip
    - Random zoom
    - Contrast and rotation transformations

This ensured generalization.

Model Architecture

Birds vs Squirrels Classifier

- Base Model : Xception (pretrained on ImageNet, frozen)
- Top Layers :
    - Global Average Pooling
    - Dropout (0.5)
    - Dense (256,ReLU)
    - Dense (3,Softmax)
- Loss : Sparse Categorical Crossentropy
- Metrics : Accuracy
- Optimizer : Adam

Birder 358-Class Classifier

- Base Model : InceptionV3 (pretrained on ImageNet)
- Top Layers :
    - Global Average Pooling
    - Dense (512,ReLU) → BatchNorm → Dropout(0.2)
    - Dense (256,ReLU) → BatchNorm → Dropout(0.2)
    - Dense (358,Softmax)
- Metrics :
    - Accuracy
    - Top-5, Top-10, Top-20 (SparseTopKCategoricalAccuracy)
- Two-Phase Training :
    - Phase 1 : Trained head layers (base frozen)
    - Phase 2 : Unfreezed top 30 layers of base and fine-tuned

Training & Optimization

Birds vs Squirrels

- Trained for 10 epochs with EarlyStopping
- Achieved rapid convergence

Birder (358-class)

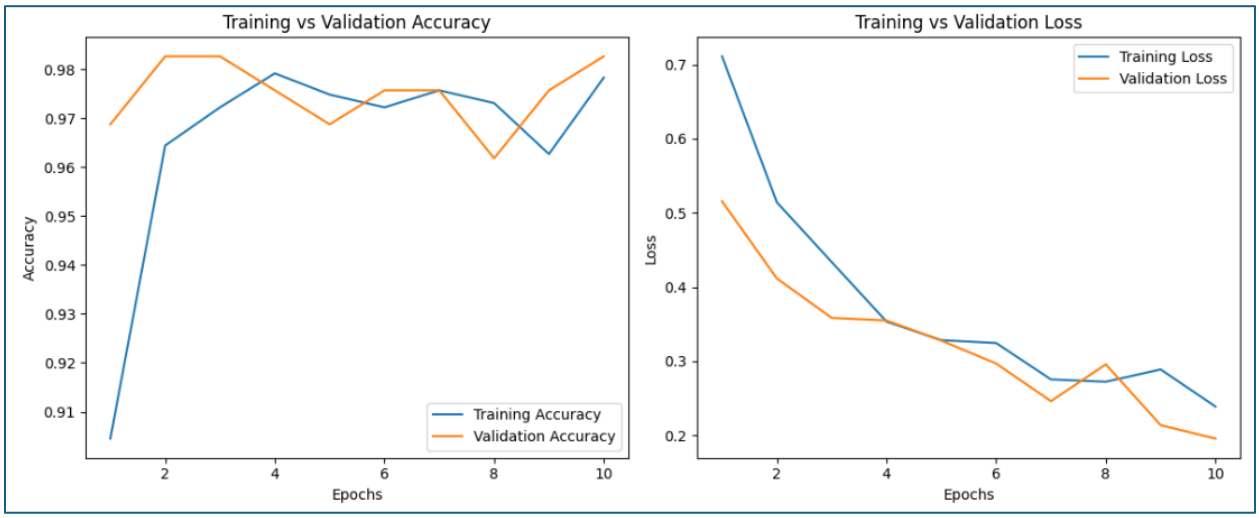
- Phase 1 : Trained only the classification head for 5 epochs (took around 3 hours total)
- Phase 2 : Unfroze top 30 layers and trained model for 5 more epochs (took around 4 hours)
- Full base unfreezing was avoided due to >12 hours runtime per epoch

Final Training Results

Birds vs Squirrels :

- Accuracy over Epochs
- Accuracy observed for Training ~ 98%
- Accuracy observed for Validation ~ 98%

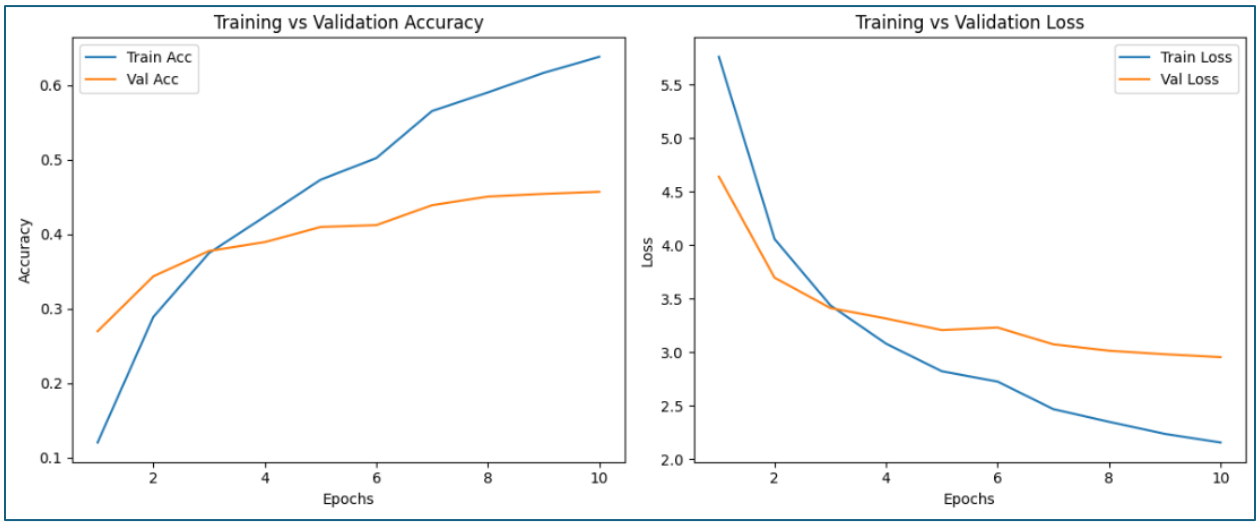
Minor overfitting was observed post epoch 6, but the overall generalization seemed correct



Birder 358-class :

- Accuracy over Epochs
- Top-1 Accuracy : ~ 46%
- Top-5 Accuracy : ~ 76%
- Top-10 Accuracy : ~ 85%
- Top-20 Accuracy : ~ 91%
- Average Top-K Accuracy ~ 74%

Final loss and metrics were stable with reasonable generalization



Experiments Conducted Before Reaching Final Model

Throughout the development of this project, several model variants and techniques were explored to maximize accuracy and generalization performance. Key experiments included:

- EfficientNetB0 : Tried early on, accuracy was promising but took over 6 hours per epoch, which didn't seem feasible
- ResNet50 : Incompatible due to 224×224 input constraint, rejected to maintain shared preprocessing
- Tuned Dropout and Dense Layers : Helped with overfitting control

- Partial Unfreezing (last 30 layers only) : Balanced compute time and fine tuning benefits. With the whole model unfreezed the run per epoch time took more than 14 hours and still didn't complete even 1 epoch
- Added BatchNormalization layers : Improved convergence speed and helped regularize the model

Peer Model Comparison

To benchmark my model performance, I compared results with a peer – Julian de Nijs (jd3846@drexel.edu) who used different architectures :

Birds vs Squirrels :

Metric	My Model	Julian's Model
Accuracy	~98%	~94%
Architecture	Xception	MobileNetV2
Input Size	299×299	224×224

Observation : Xception provided better feature extraction

Birder 358-class :

Metric	My Model	Julian's Model
Top-1 Accuracy	~46%	~29%
Average Top-K Accuracy	~74%	~59%
Architecture	InceptionV3	ResNet50V2

Observation : While my model took longer to train, regularization and partial unfreezing led to substantially better accuracy. Julian's model was faster but underfit the complex distribution.

Conclusion

The model did show some sense of overfitting but based on the computational time the model submitted had obtained the best results so far with minimal overfitting compared to all other models I ran. The biggest challenge was managing runtime while ensuring compliance with input size, file formats, and evaluation requirements.

What worked better in the final model :

- Partial unfreezing balanced computing time and accuracy
- Overfitting controlled to some extent with slight higher dropout, data augmentation and batch normalization