Bird Image Classification

Emma, Helen, Jack, Kevin, Love

All group members have contributed evenly to the project

The Project

- Classify 525 Bird Species using multiple image classification algorithms
  - Manually Trained CNN
  - Pre-trained MobileNetV2 CNN
  - Pre-trained ResNet CNN
  - Pre-trained Vision Transformer
  - Manually Trained Vision Transformer
- Compare the results of different models

Ashy Thrush

The Data

African Firefinch

Barn Owl
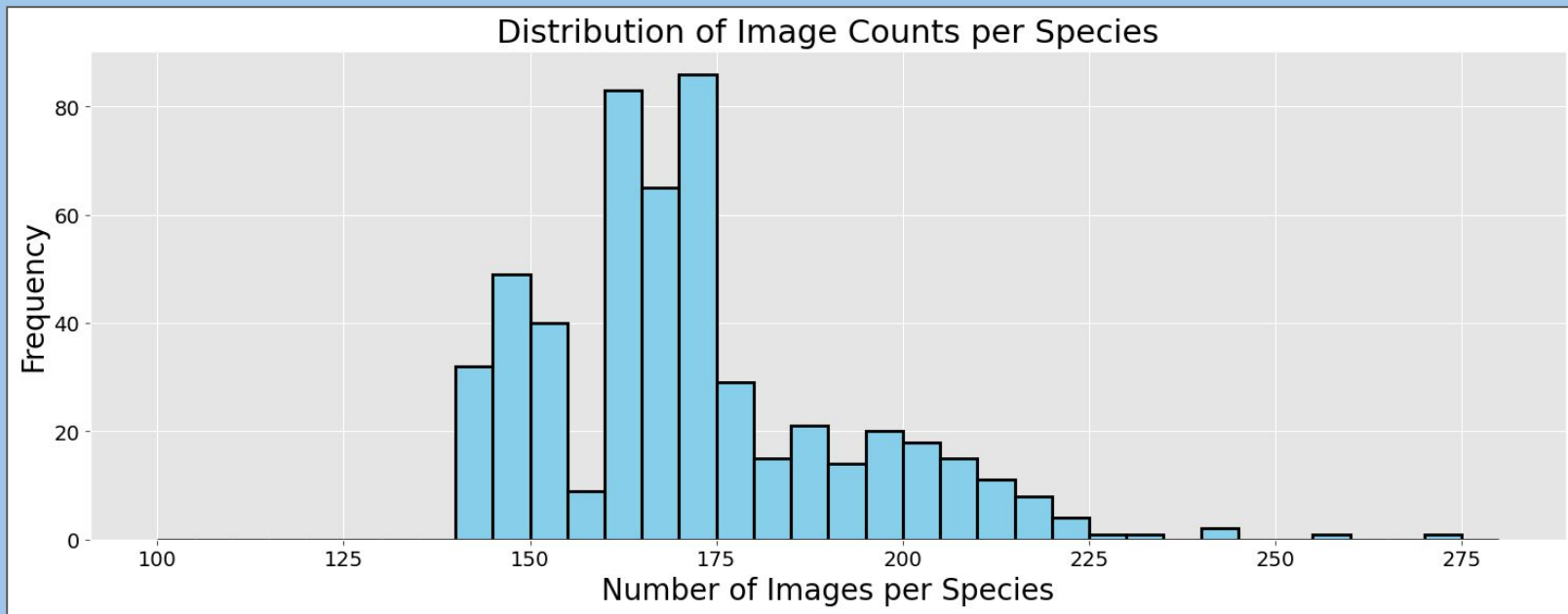
Kiwi

Antbird

Altimira Yellowtooth

American Dipper
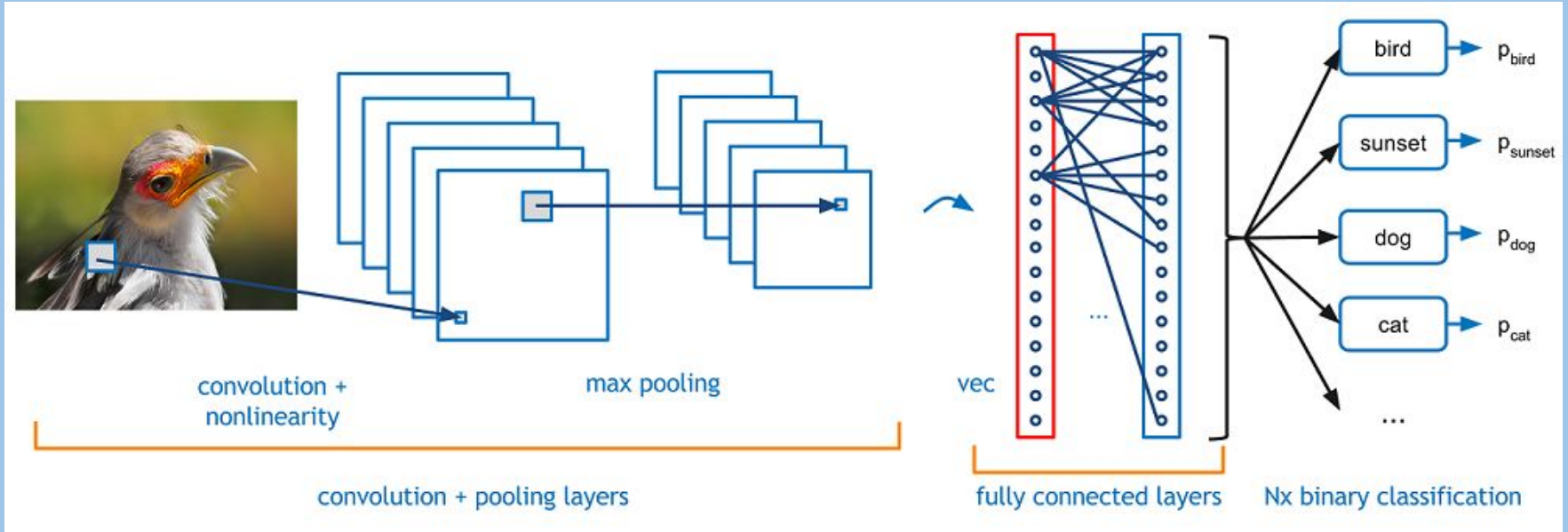
Dusky Robin

The Data

- Kaggle dataset containing 89885 Images of 525 bird species

- Each bird is a 224 x 224 x 3 colour image in jpg format

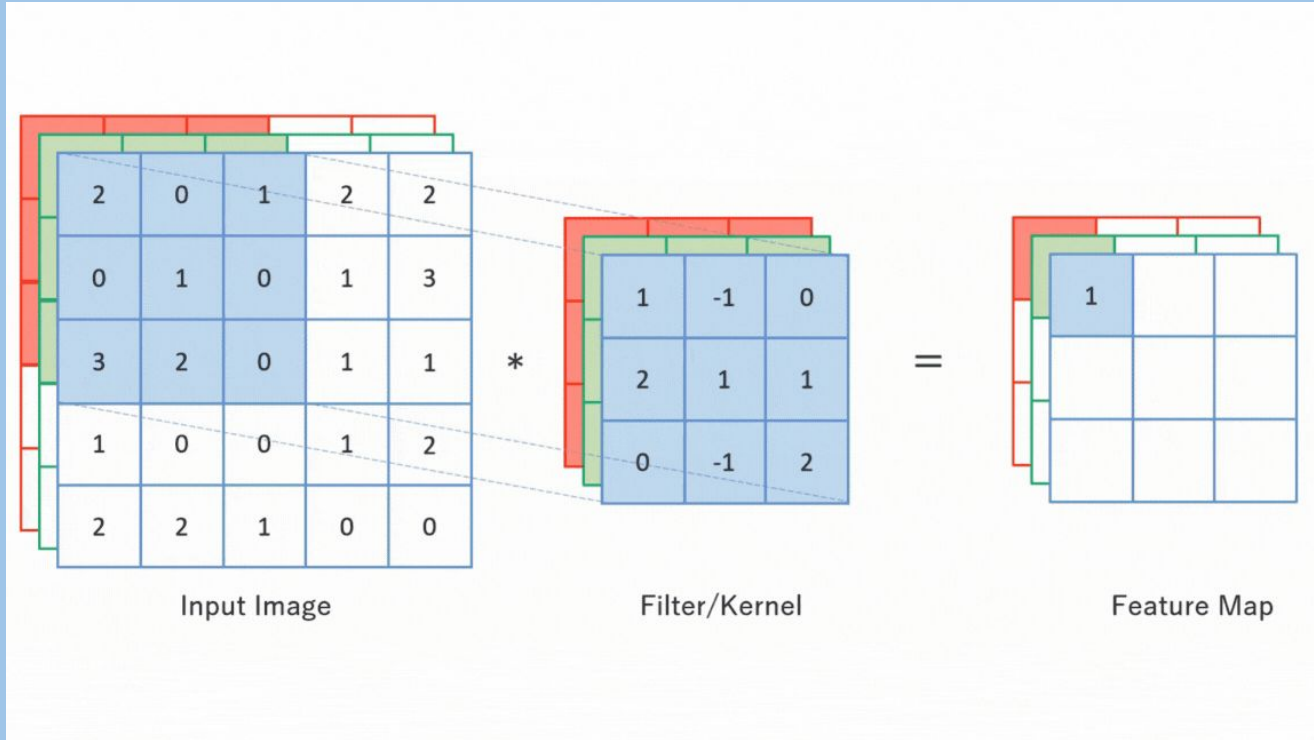- 80/10/10 split for training, validation, and test data

Distribution of Image Counts per Species

Male and female birds

A Brief Recap on CNN

convolution + nonlinearity

max pooling

convolution + pooling layers

vec

fully connected layers

bird — p_{bird}

sunset — p_{sunset}

dog — p_{dog}

cat — p_{cat}

Nx binary classification

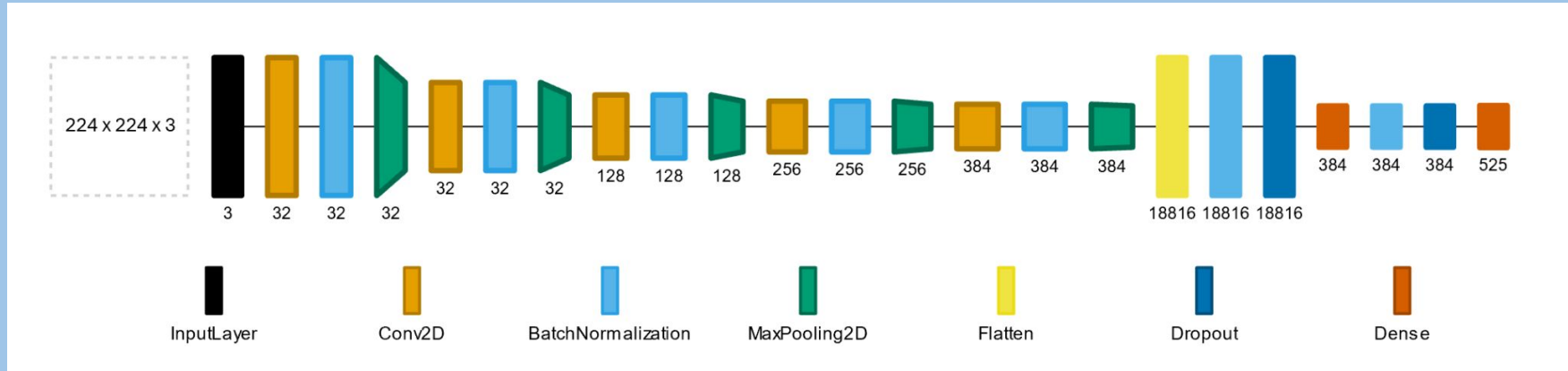# Convolution in Action!

Input Image   Filter/Kernel   Feature Map

Manually Trained CNN

- A baseline for model comparison
- Simple architecture: 5 convolutional layers
- Time taken: ~ 1 hr for 15 epochs
- Test accuracy: 72.9%

Visualization generated using Net2Vis

Manually Trained CNN with Hyperparameter Optimization

- 5 convolutional layers + Bayesian optimization
- BO search failed midway → took "Best Value So Far" instead
- Time taken: ~ 5 hr for BO search, ~ 1.5 hr for 20 epochs
- Test accuracy: 78.3%

Visualization generated using Net2Vis
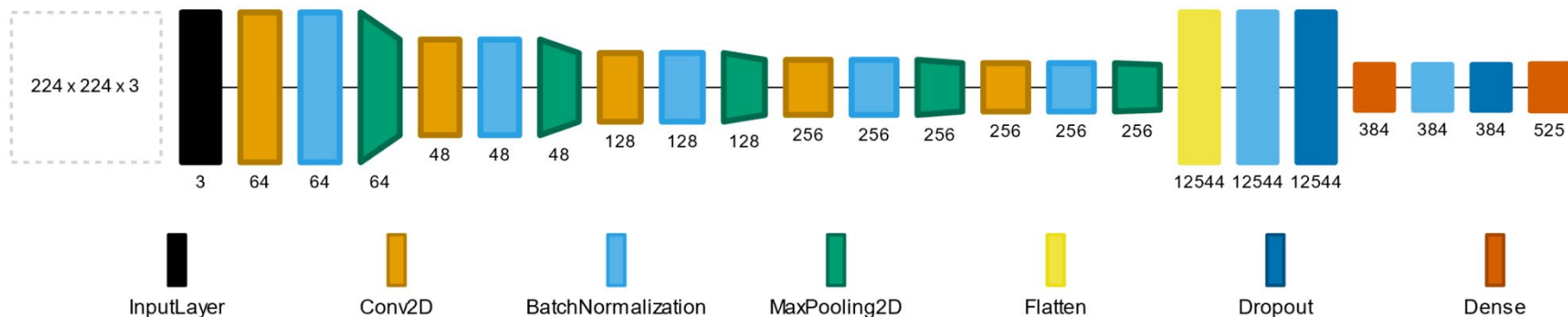
Manually Trained CNN with Hyperparameter Optimization

- Relatively poor performance

- Failed to achieve 100% accuracy in any single class
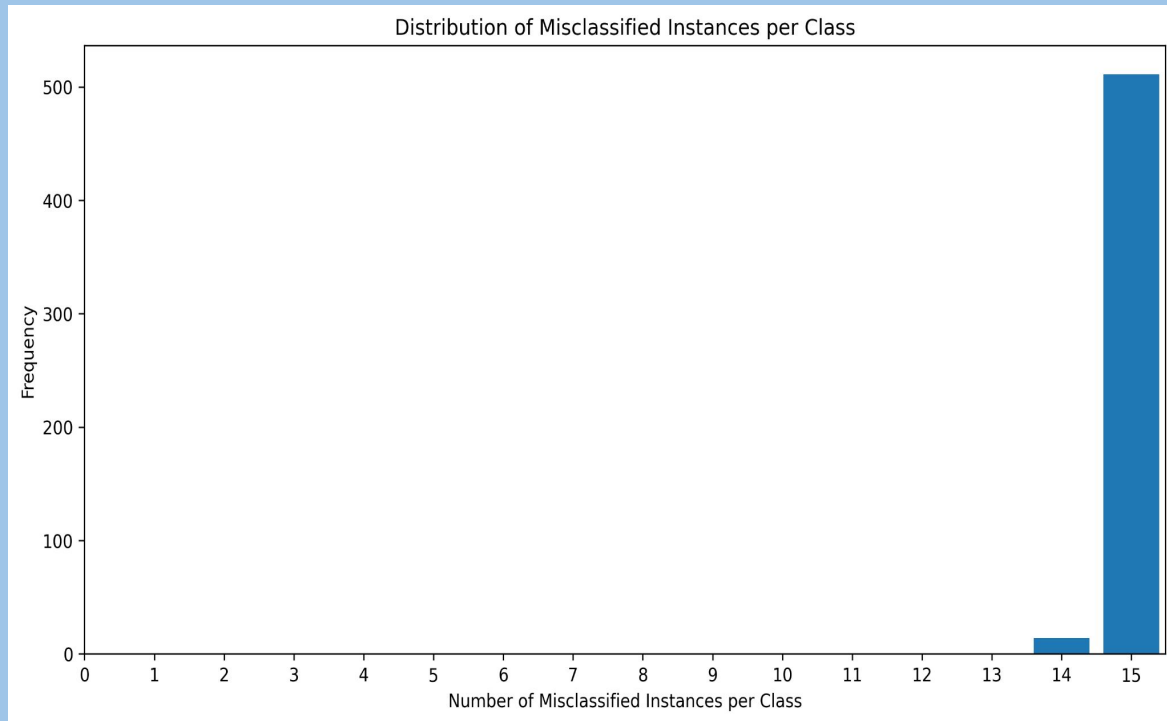
Transfer Learning

- Pre-trained model adapted to specific task
- Good when:
  - Small dataset
  - Limited computational resources
- Choose a base-model
- Change the top layers
- Freeze the layers in the base-model
- Train the model on your own data.
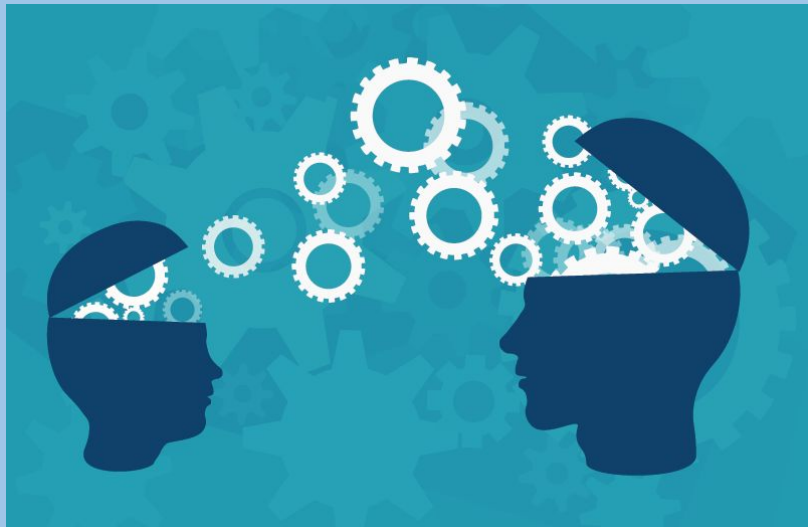
Fine-tuning

- Increases performance

- Some or all of parameters are trainable

- Only some of the layers are frozen

MobileNetV2

- Lightweight CNN

- For limited computational resources

- Trade-off between accuracy and speed

- Fun to bring your classifier out in the nature!
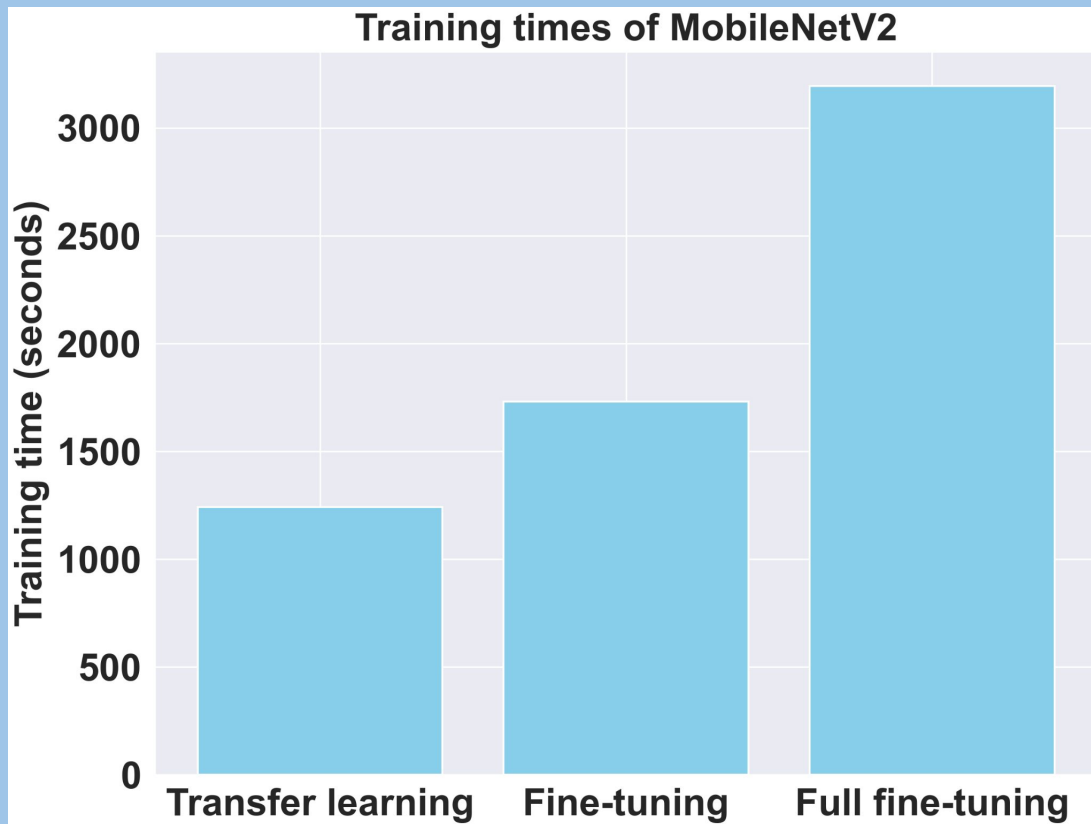
- Depth-wise separable convolutions

Training MobileNetV2

- Transfer learning
- Fine-tuning
- Full fine-tuning
- Accuracies on test:
  - 80 %
  - 91 %
  - 92 %

Training, validation, test accuracies of MobileNetV2
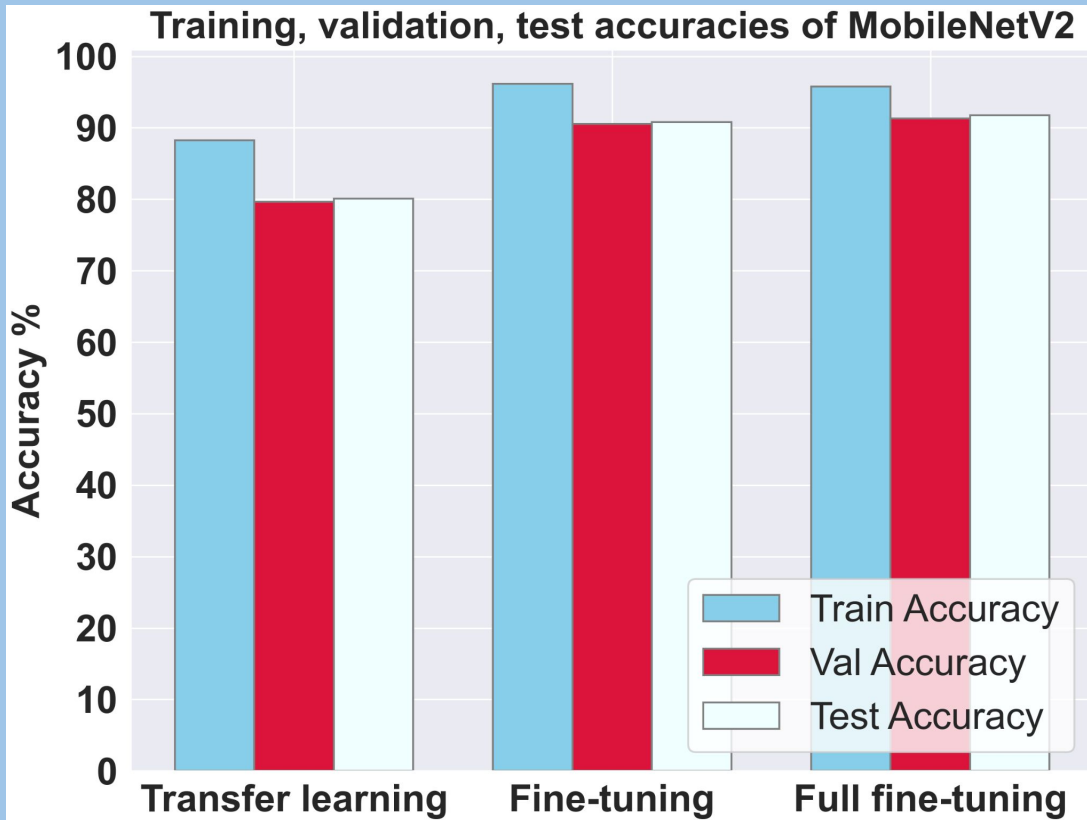
Computation time

- Considering performance and computation time:

- Fine-tuning is the best!

91 % Accuracy - not great, not terrible!

ResNet50 - Pre-trained CNN

- ● **ResNet50 is a 50 layer Residual CNN**
  - ○ Utilises skip connections to solve the "Vanishing Gradient Problem"
- ● **Medium-Weight CNN Model**
  - ○ Deeper CNN than MobileNet
  - ○ Not as computationally intensive as transformers
- ● **Pretrained on Imagenet**
  - ○ Loaded by transfer learning with Keras
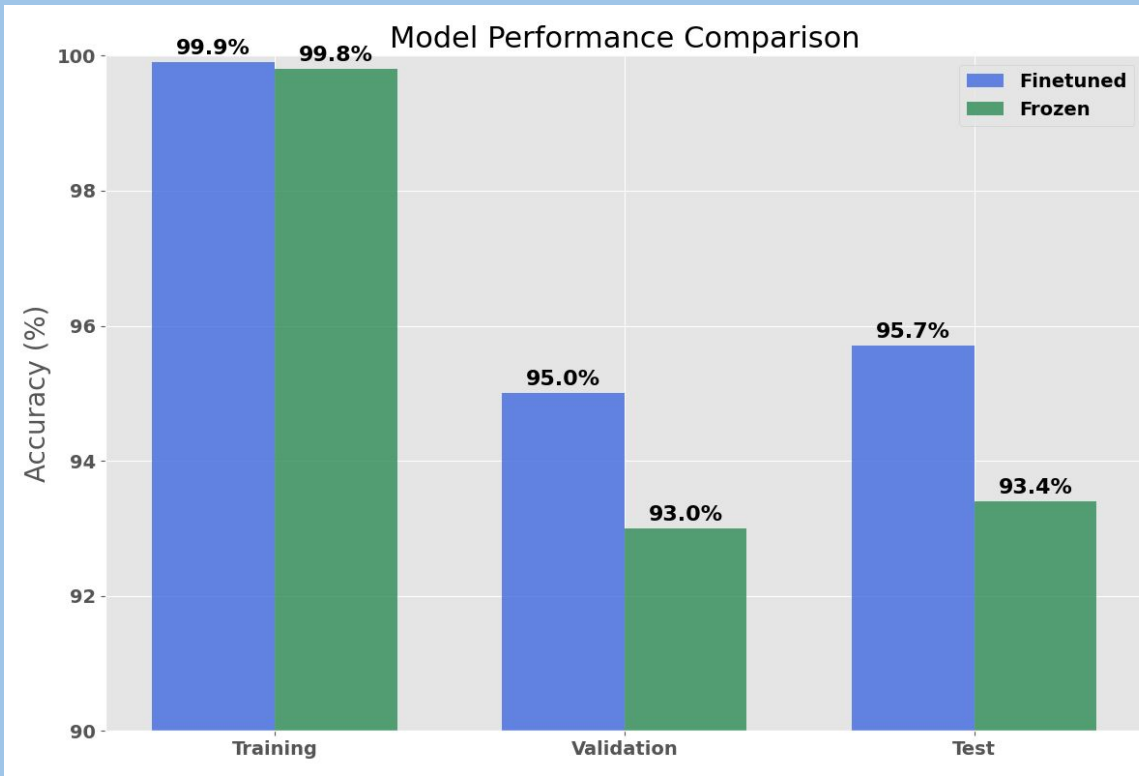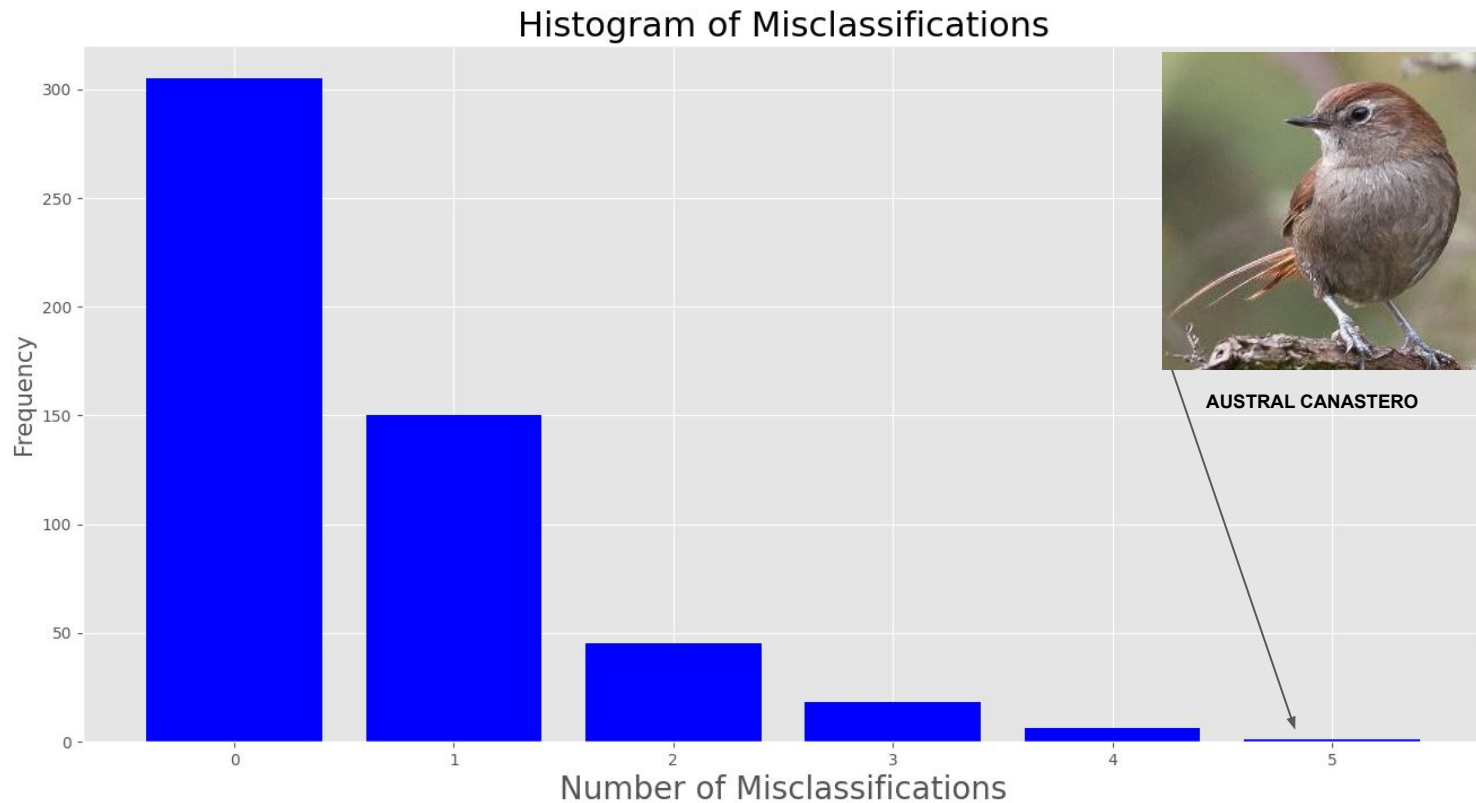 - ○ Both Trainable and Frozen ResNet models were benchmarked
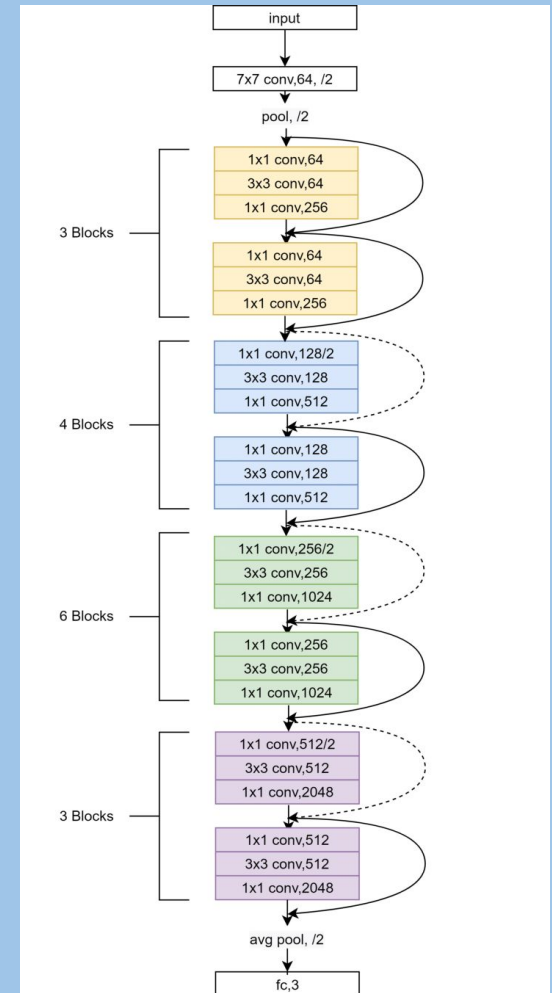
ResNet50 Results

- Full Fine-tuning vs Frozen

- Run Time (T4 GPU):
  - Fine-tuning: ~12 min/epoch
  - Frozen: ~3.5 min/epoch

- Test Accuracy:
  - 95.7% Fine-tuning
  - 93.4% Frozen

ResNet50 - Misclassified Birds

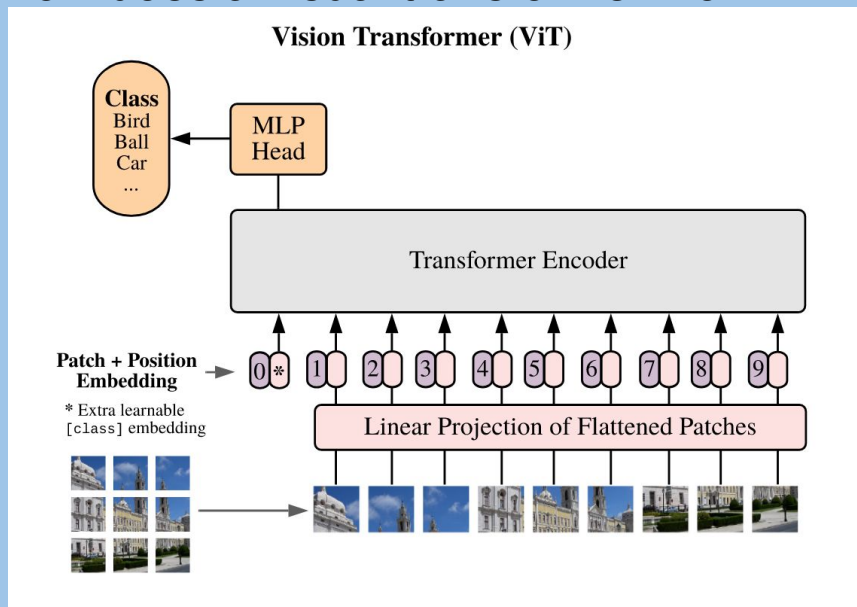# Vision Transformer

How does a visual transformer work?

Vision Transformer (ViT)

Image source: [2010.11929]

Recap: transformer encoders
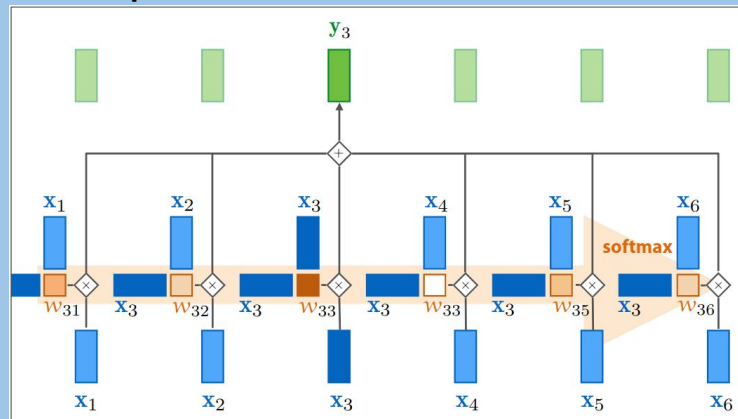
Image source: P. Bloem's lecture notes

Multi-head attention

Image source: Phillip Lippe's tutorials

You do the attention head above multiple times in parallel.
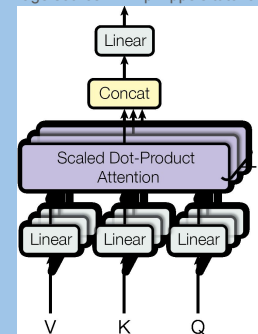
Details of my visual transformer

- ~5.3 minutes per epoch, trained on a T4 GPU

- Best result achieved after ~30 epochs

- Validation accuracy peaked at ~70%

- Accuracy on the test set: ~71%

- 19 different classes got 100% accuracy on the test set

Accuracy distribution per class
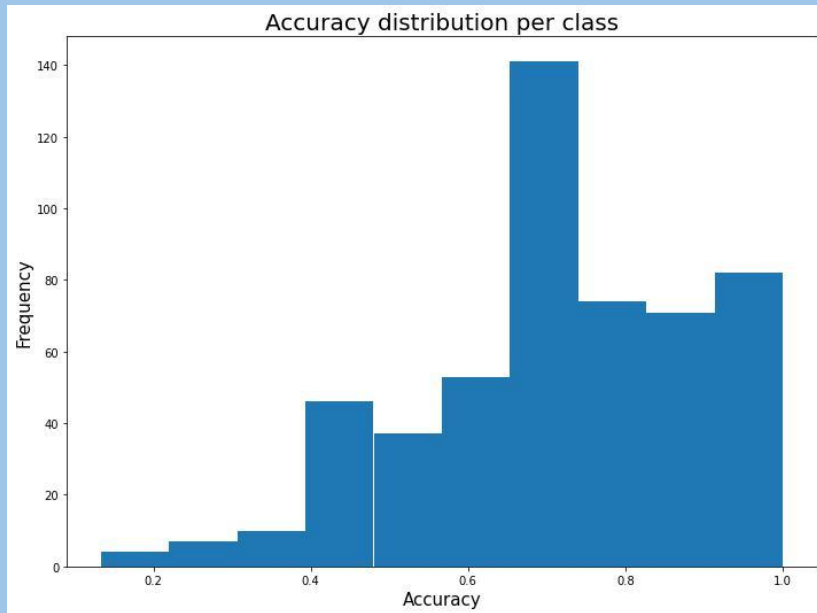
Good and bad bird examples

Bad birds:

Good birds:

Very modest results compared to the rest, is this expected?

From *"An Image is Worth 16x16 Words: Transformers for Image Recognition"* by A. Dosovitskiy, et al. ([2010.11929]):

When trained on mid-sized datasets (...) these models yield modest accuracies of a few percentage points below ResNets of comparable size. This seemingly discouraging outcome may be expected: Transformers lack some of the inductive biases inherent to CNNs (...) and therefore do not generalize well when trained on insufficient data.

Should we then neglect Transformers for image classification?

However, the picture changes if the models are trained on larger datasets (...) Our Visual Transformer attains excellent results when pre-trained at sufficient scale and transferred to tasks with fewer data points.
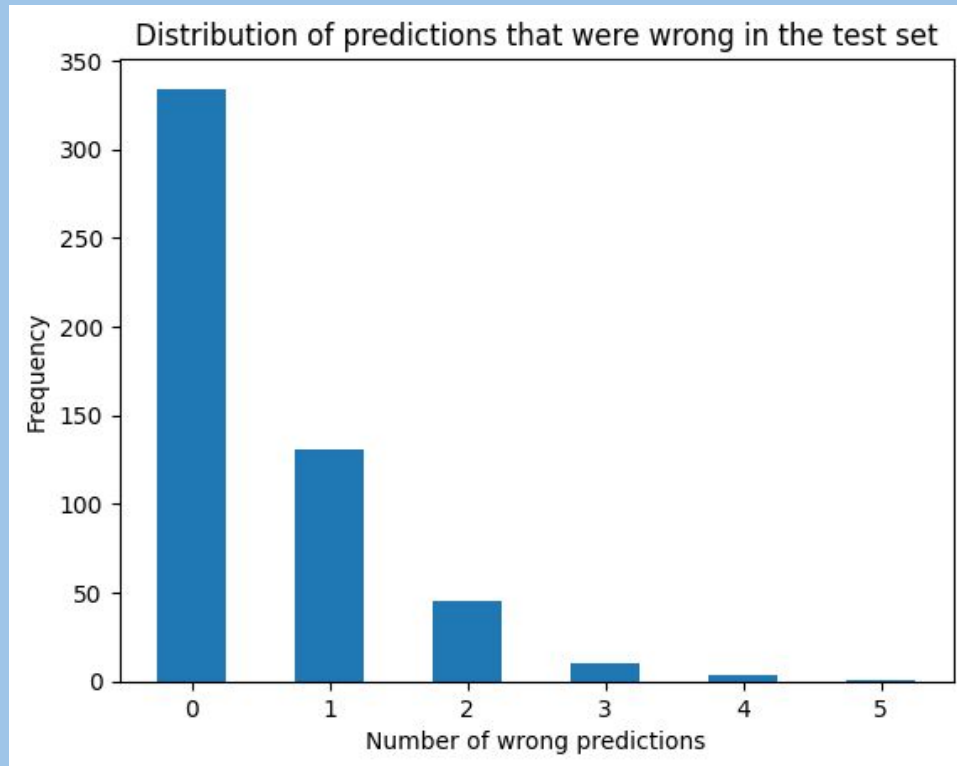
Pre-trained Vision Transformer results

28 transformer-encoder blocks (freezed), with one dense trainable layer.
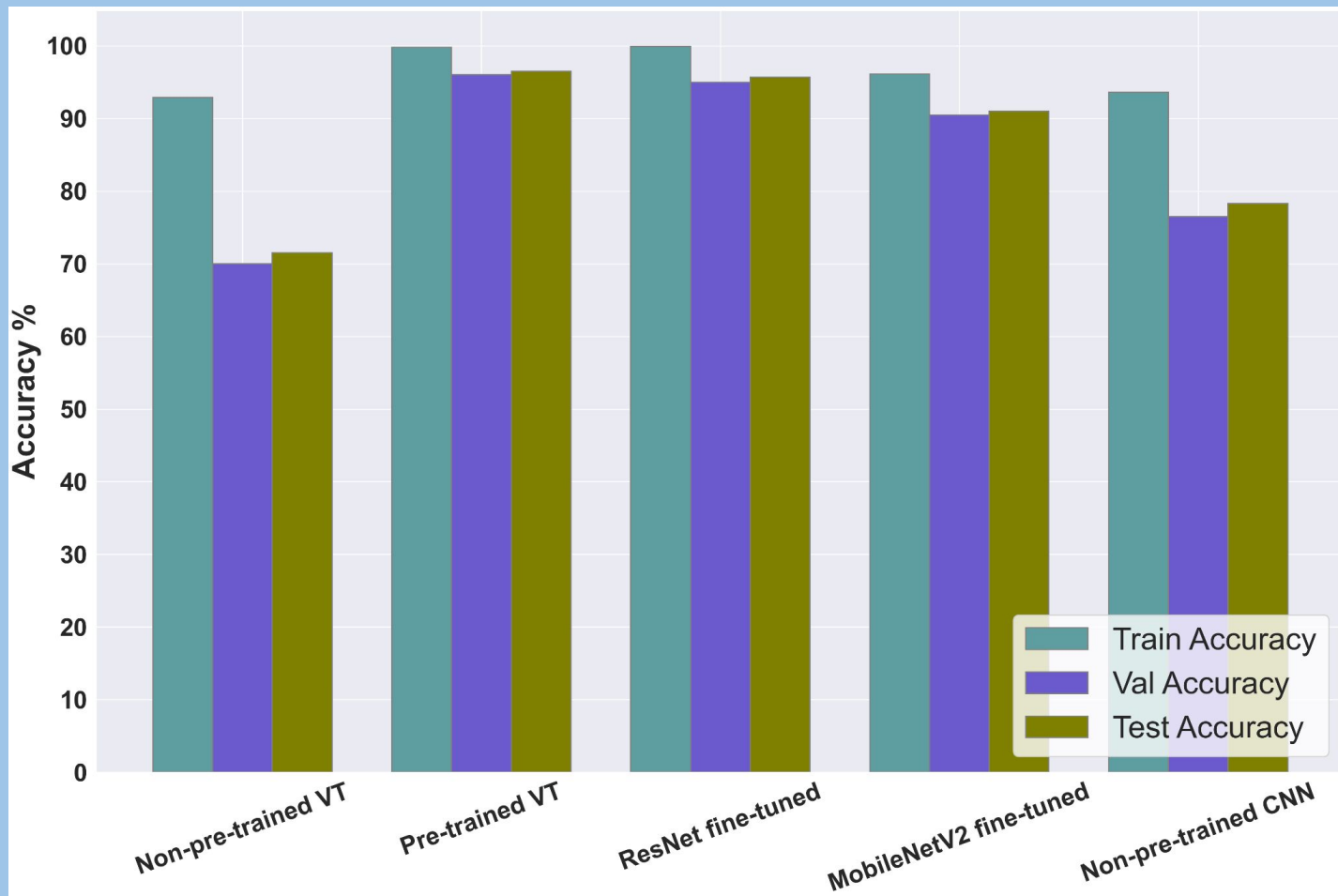
Test accuracy of 96.5 %.

Trained for 25 epochs which took around 4h.

Around 75% of the training time was spent on propagating the images through the network.

Distribution of predictions that were wrong in the test set

Comparison of models

Conclusion

Dataset was clean, easy to implement, and large.

Difficult to train a model from scratch on such a large dataset.

Convolutional neural networks are quick and perform well.

Vision transformers perform well, but are slow.

Appendix

Data and Codes

Kaggle dataset used:
https://www.kaggle.com/datasets/gpiosenka/100-bird-species

All codes used in this project can be obtained from this GitHub repository:
https://github.com/tgr213/ML-Bird-Project

Augmentations

- Due to the relative high training accuracies, we attempted implementing augmentations to increase the number of images.
- We tried adding augmentations to the under-represented bird species so that there was an even amount of images per bird species
  - When tested on the Frozen ResNet50, the Test Accuracy decreased from 93% to 90%
    - This is possibly due to an imbalance in augmentations per class or an insufficient amount of augmentations
  - The Fine-tuned ResNet50 (and other models) could not train in a reasonable amount of time
- Ideally, if we had unlimited GPU access we would try adding 1-3 Augmentations per bird and see how that would impact the Validation accuracy.

Justification for Adopting 80/10/10 Split

- According to the dataset description on Kaggle, the original test and validation sets were "hand-selected to be the 'best' images". Thus, if a portion of the training set is used as validation/test images, it would inevitably decrease the models' performance on the validation and test datasets.

- This approach, however, could enhance the generalizability of our models and potentially improve their performance on unseen images.

Depthwise-separable convolution

Credit: https://animatedai.github.io/
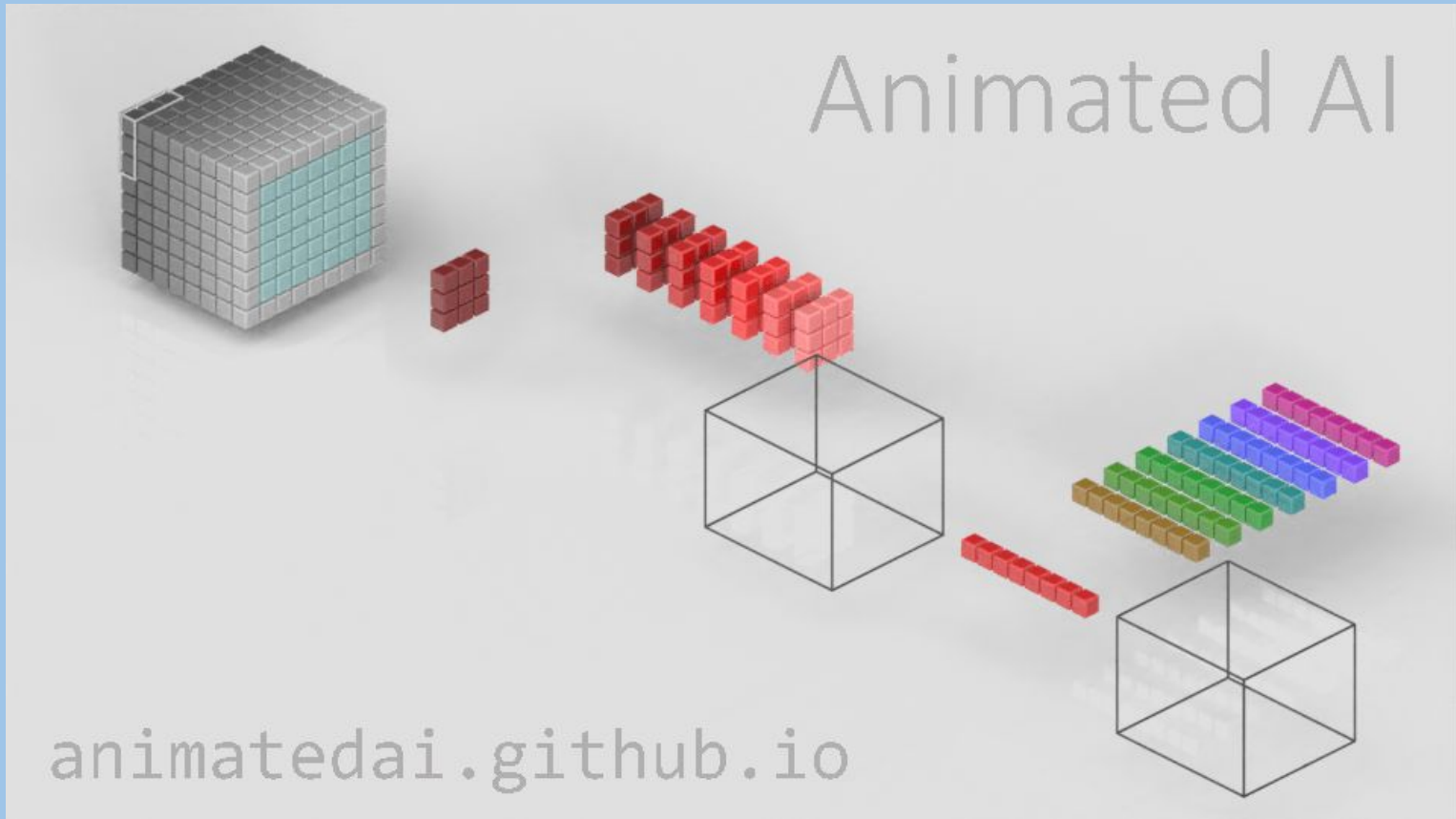
Training Details Manually Trained CNN
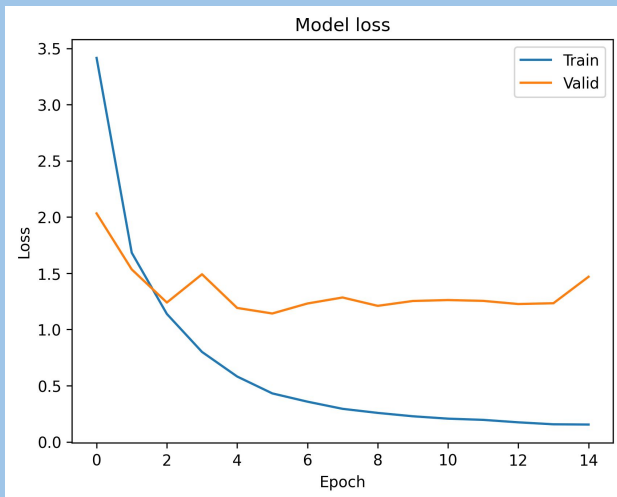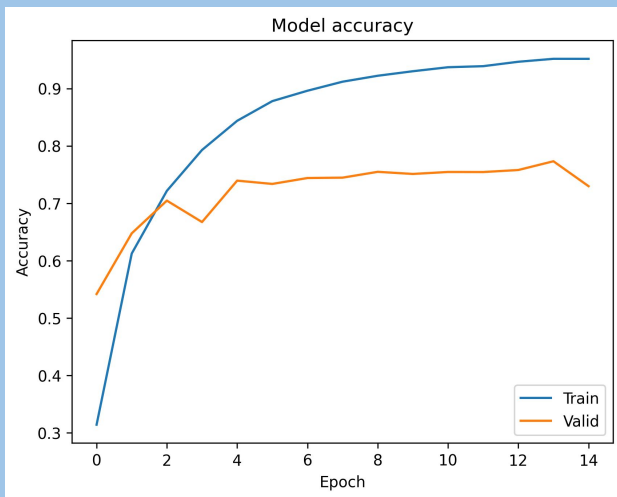
CNN with no hyperparameter optimization

Optimizer: Adam

Loss function: categorical_crossentropy

Metric: Accuracy

Learning rate: 0.00062

Callbacks: early stopping, model checkpoints

Training Details Manually Trained CNN

CNN with bayesian optimization
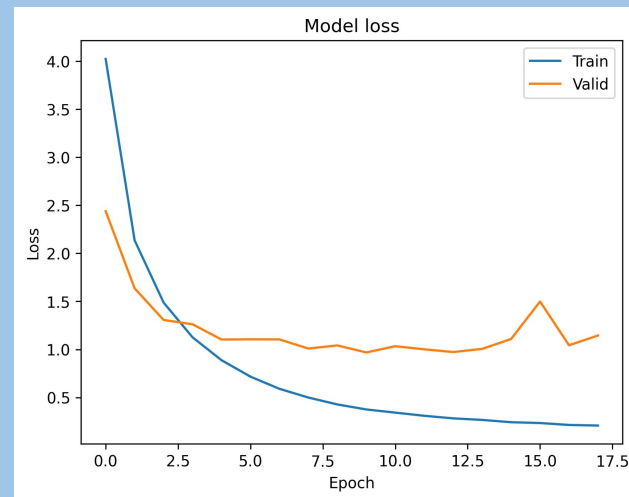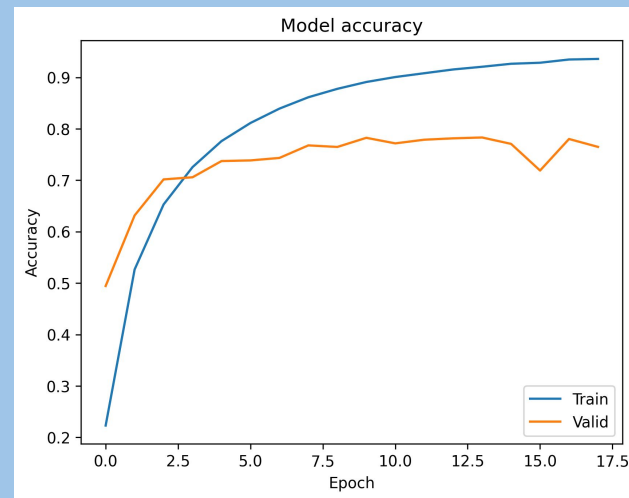
Optimizer: Adam

Loss function: categorical_crossentropy

Metric: Accuracy

Learning rate: 0.00035534

Callbacks: early stopping, model checkpoints

BO search in Keras Tuner: max_trials = 70, failed after trial #27

Manually Trained CNN with Original 94/3/3 Split

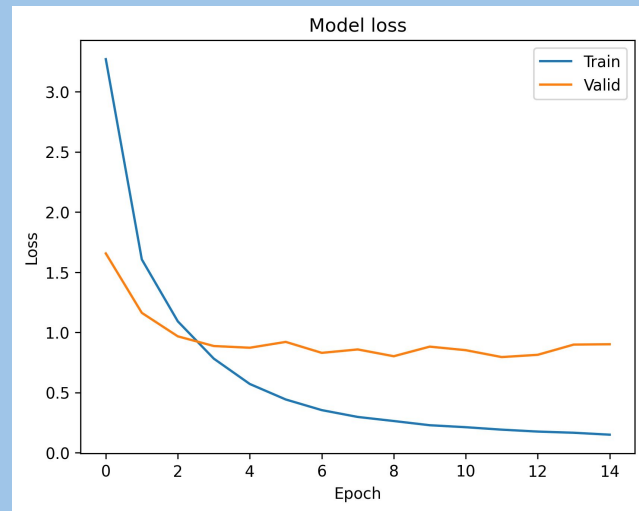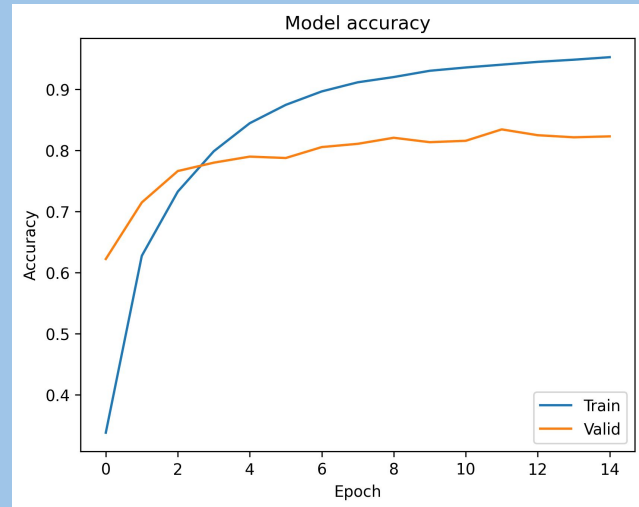CNN with no hyperparameter optimization

Same network architecture & settings as its 80/10/10 split counterpart

Train accuracy: 95.3%

Validation accuracy: 82.3%

Test accuracy: 86.2%

→ Better performance

Manually Trained CNN with Original 94/3/3 Split

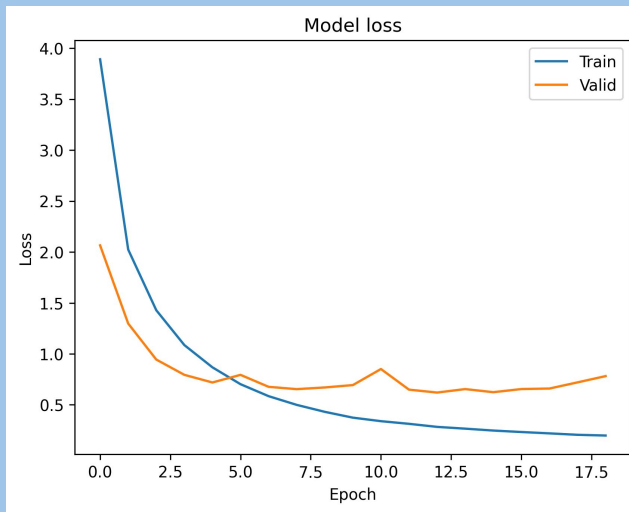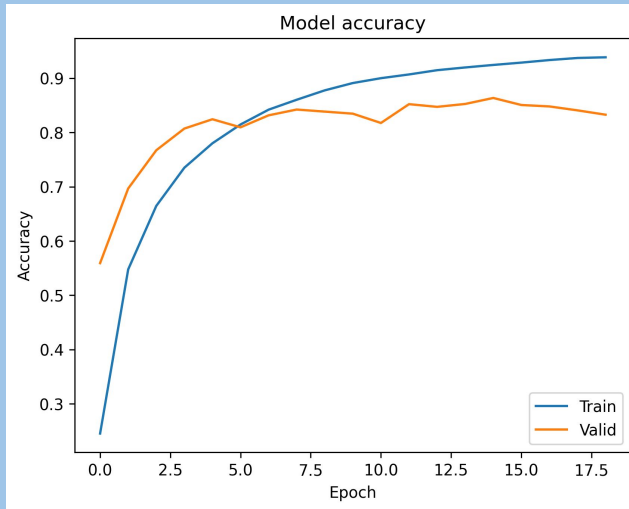CNN with bayesian optimization

Same network architecture & settings as its 80/10/10 split counterpart

Train accuracy: 93.9%

Validation accuracy: 83.3%

Test accuracy: 87.7%

→ Better performance x 2

Training details transfer learning and fine-tuning on MobileNetV2

Used the MobileNetV2 model from TensorFlow

Trainable layers: 77 first layers from base model + 4 additional top layers

Optimizer: Adam

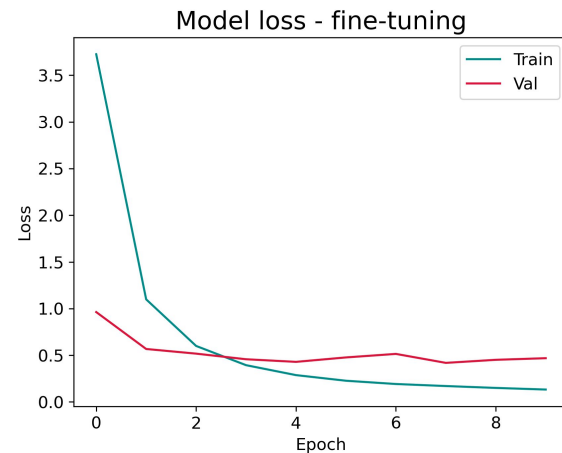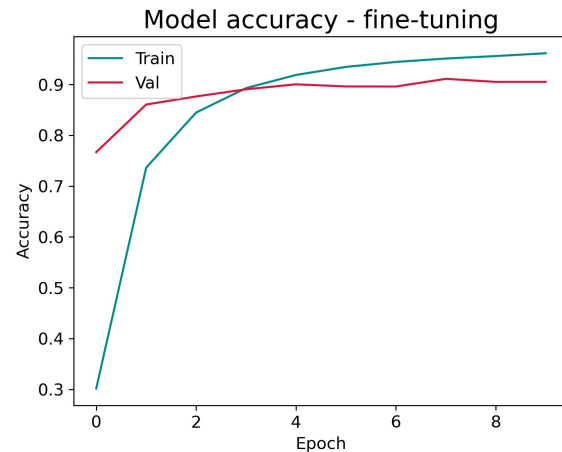Loss function: categorical_crossentropy

Metrics: Accuracy

Learning rate: 0.0001

Epochs: 10

Batch size: 32

Trained on a T4 GPU

ResNet50 Training Structure / Hyperparameters

- Preprocessed images with `tensorflow.keras.applications.resnet50 import preprocess_input`
- Then applied ResNet50, Flattened, Dense 512 layer, Dropout, and Classification output layer
- Utilised early stopping and Learning Rate Scheduler to train over max 30 epochs
- Hyperparameters:
  - Initial LR: 1e-4
  - Final LR: 1e-5
  - Loss: sparse categorical cross-entropy
  - Early stopping monitor: Validation Loss

ResNet50 Epoch Loss/Accuray Plots

Frozen Transfer Learning ResNet50:
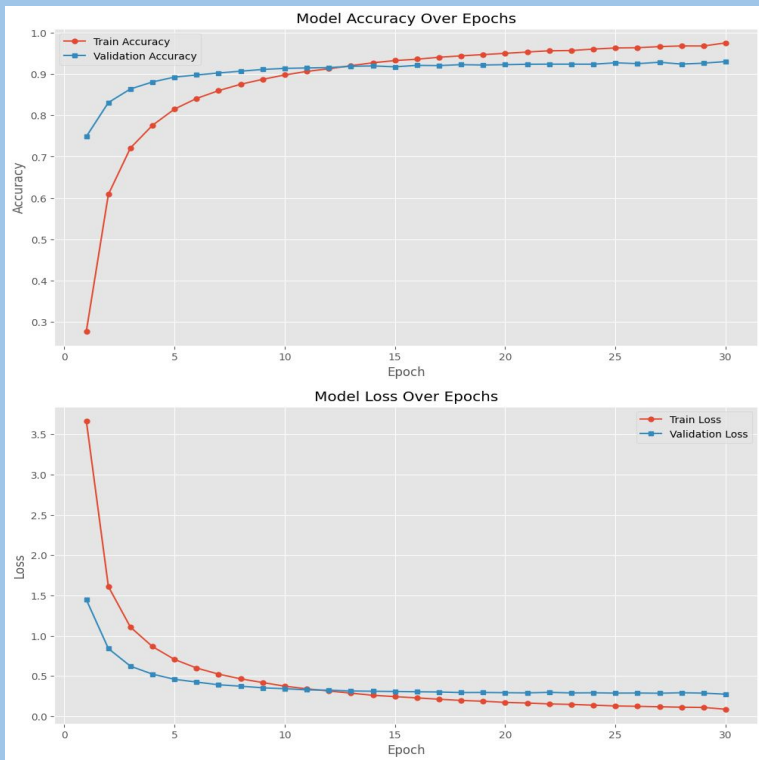
Fine Tuned ResNet50:
(restored weights from epoch 8)
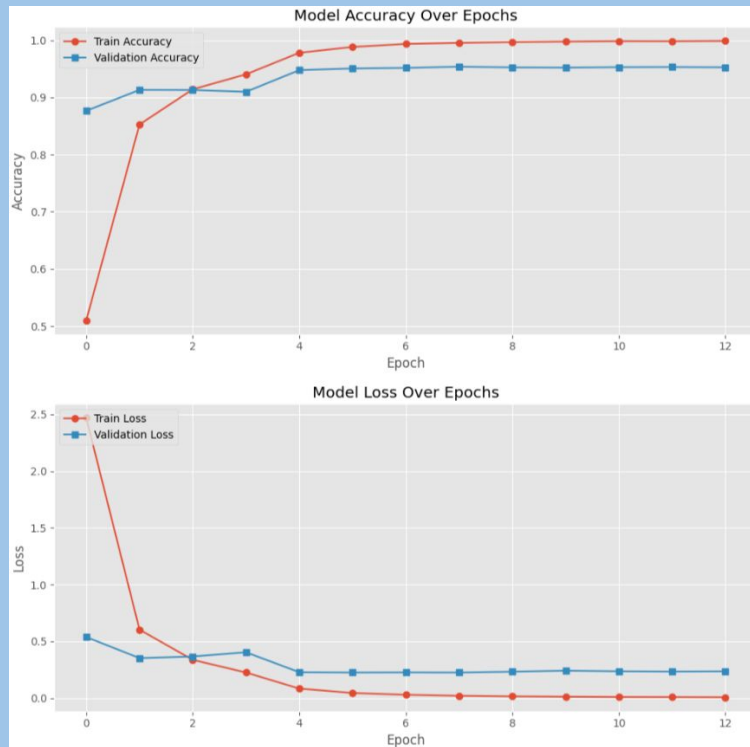
Details on the not pretrained visual transformer

- Pre-LN transformer (normalization was done before the multi-head attention layer and also before the FFN, instead of after)
- Activation function: `ReLU`
- Optimizer: `AdamW`
- Learning rate: `0.001`, weight_decay: `1e-4`
- Relevant hyperparameters:
  - `patch_size: 14x14`
  - `embedding_dimension: 128`
  - `hidden_dim: 512`
  - `num_heads: 8`
  - `num_layers: 6`
  - `dropout: 0.1`

Training details pre-trained vision transformer

Used the ViT_l32 model from keras.

Trainable layers: 1 dense layer with 525 nodes.

Optimizer: Adam.

Loss function: sparse_categorical_crossentropy.

Metrics: Accuracy

Learning rate: Start value = 0.001, reducing by a factor of 10 every seven epoch.

Trained on a L4 GPU, which took 4 hours for 4 epochs.