Project | Deep Learning: Image Classification using CNN and Transfer Learning

Presented by:

Mohamad Traiki
Sergei Volkov
Mitesh Parab

Dataset Overview and Objective

Dataset used: CIFAR Dataset (CIFAR-10 / CIFAR-100)

- Created by **Alex Krizhevsky, Vinod Nair, Geoffrey Hinton**
- **60,000 colored images (32×32)** across **10 classes** (CIFAR-10)
- Classes include: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

- ❏ To develop and evaluate image classification models using **Convolutional Neural Networks (CNNs)** and **Transfer Learning** on the **CIFAR dataset**.
- ❏ To compare custom-built CNN architectures with pre-trained deep learning models, analyzing their performance, accuracy, and efficiency in classifying images.

Used **pre-trained models** from Keras Applications (e.g.,EfficientNet B0 to B7, MobileNetV2 etc..)

Used Optimizers (e.g., Adam, RMSprop, Adadelta etc..)

Data pre-processing

Self-made CNNs

Load data (cifar10) → Change to grayscale → Normalize

→ Convert labels to one-hot vectors

Transfer Models

Load data (cifar10) → Resize from 32 x 32 to 96 x 96

Callbacks

Early Stopping:

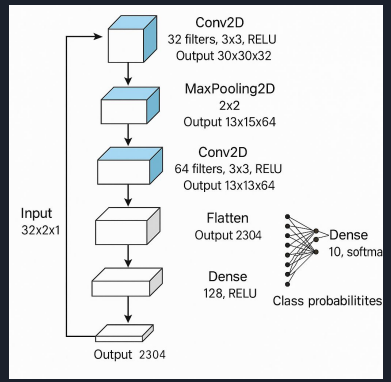
- Monitors validation accuracy
- Patience of 8 epochs. Terminates training when no improvement noticed
- Saves parameters from best performing epoch
- Early Stopping was performed on all our Models.

Learning Rate Reduction: (Transfer Models)

- Monitors validation accuracy
- Patience of 4 epochs
- Reduces learning rate by factor f when no improvement noticed.
- Sets a minimum learning rate that can' t be reduce.
- LRR was performed on only Transfer Models

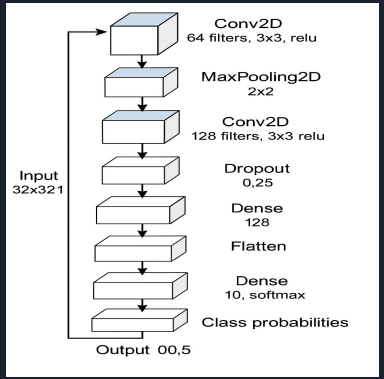
Self-made CNNS

Initial Model (Adam):



- Same number of layers.
- CNN with RMS: double number of filters
- CNN with RMS: dropout 25% of neurons.
- Both CNNS: average performance

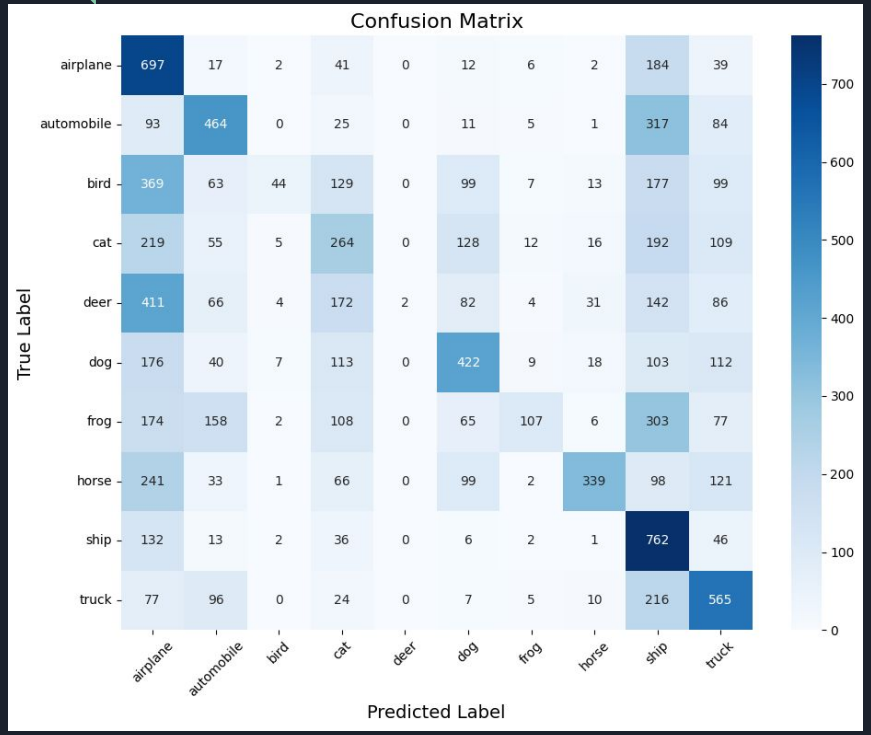
CNN with RMSprop



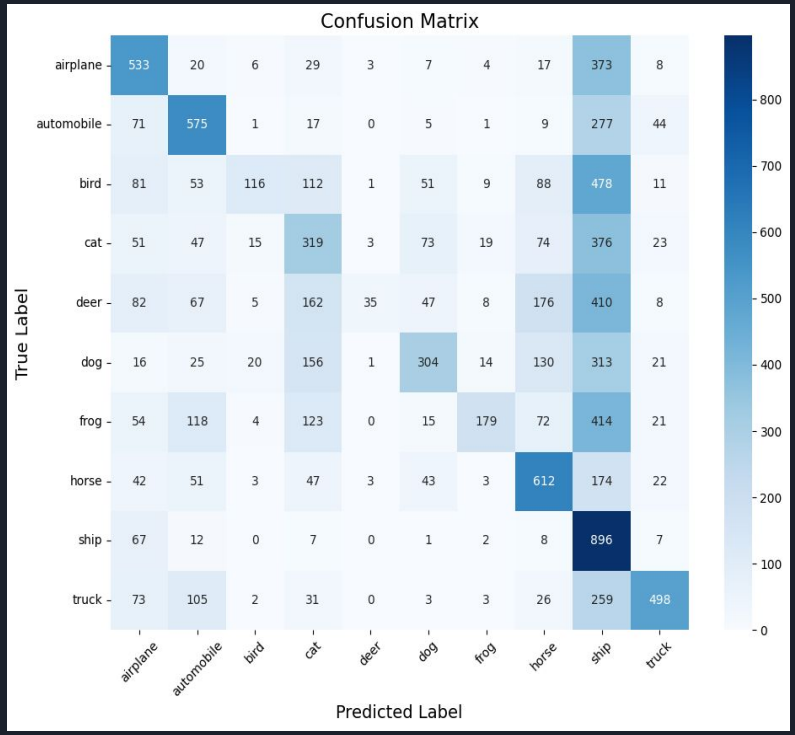
Activation: relu
Optimizer: Adam

Activation: relu
Optimizer: RMSprop

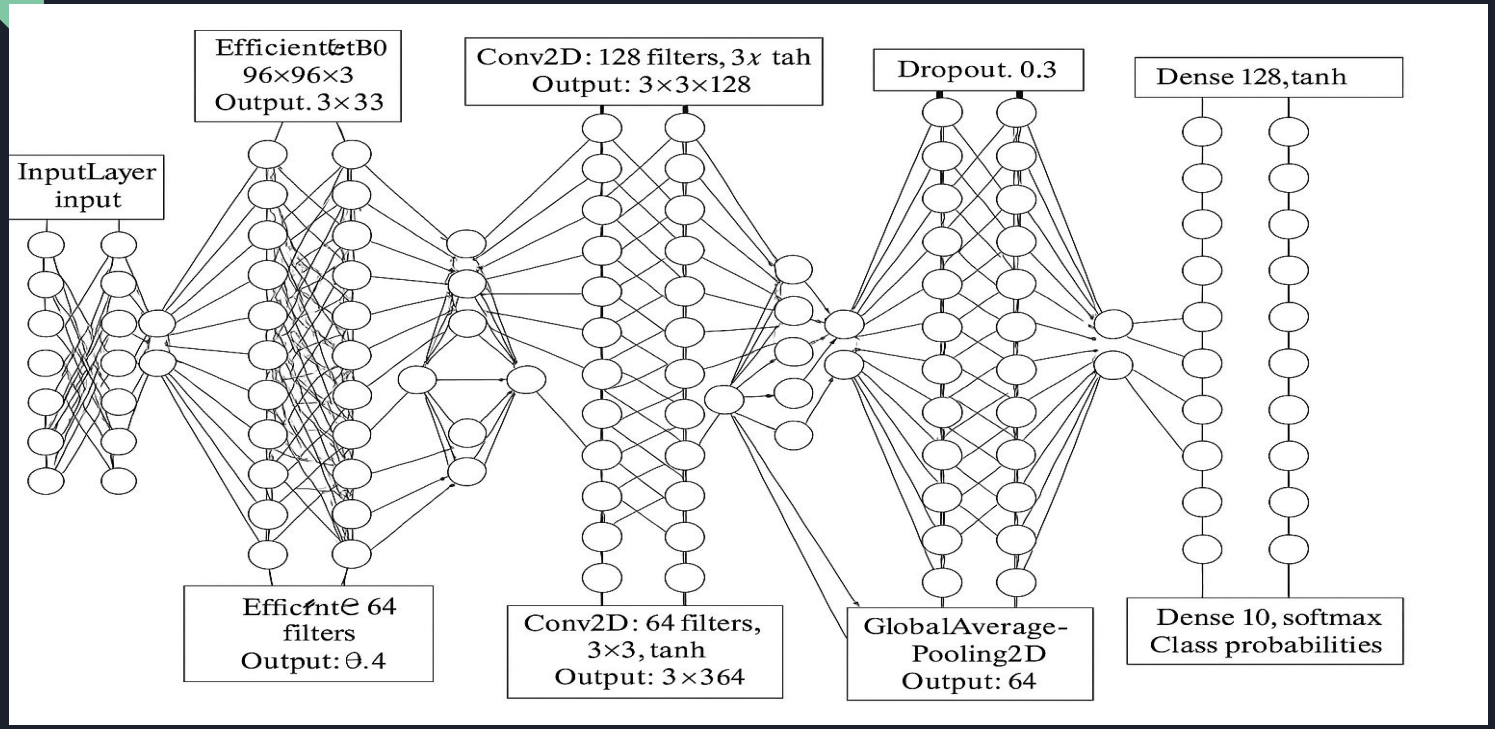
Initial Model (Adam)



CNN (RMSprop)



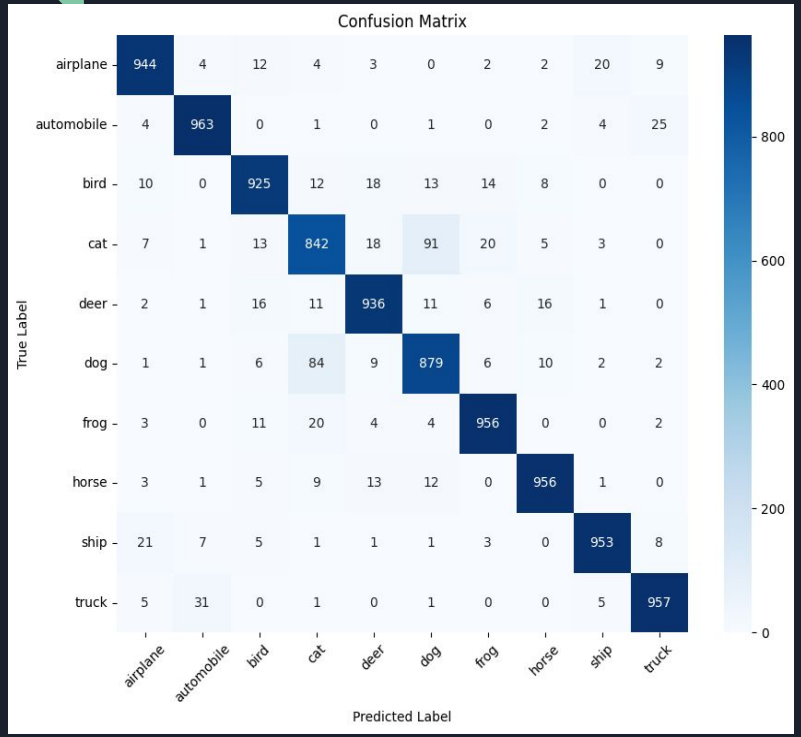
Transfer Models



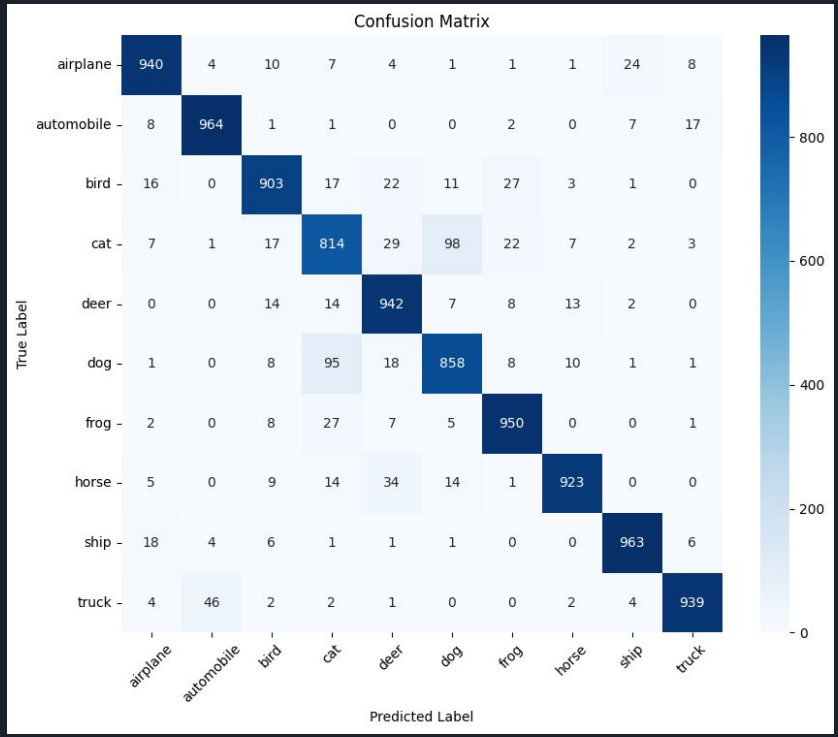
Transfer Models Architecture:

- Base model + Conv2D(128) + dropout(0.3) + Conv2D(64) + Global AVG Pooling + Dense(128) + dropout (0.4) + Dense(10)(Output Layer with 10 Classifiers)
- Both models share same architecture
- Differ in activation function (relu vs tanh)
- Use Early Stopping and Learn Rate Reduction
- Early stopping:
  - Patience: 8 epochs
  - Monitor: val_accuracy
  - Saves best weights
- Learning Rate Reduction:
  - Patience: 4 epochs
  - Monitor: val_accuracy
  - Factor: 0.2
- Introduce Checkpoints
- Initial Learning Rate: 0.001
- Fine tuned by freezing the initial layers of base model and training last 50 layers.

EfficientNETB0(tanh): Test-accuracy 0.920

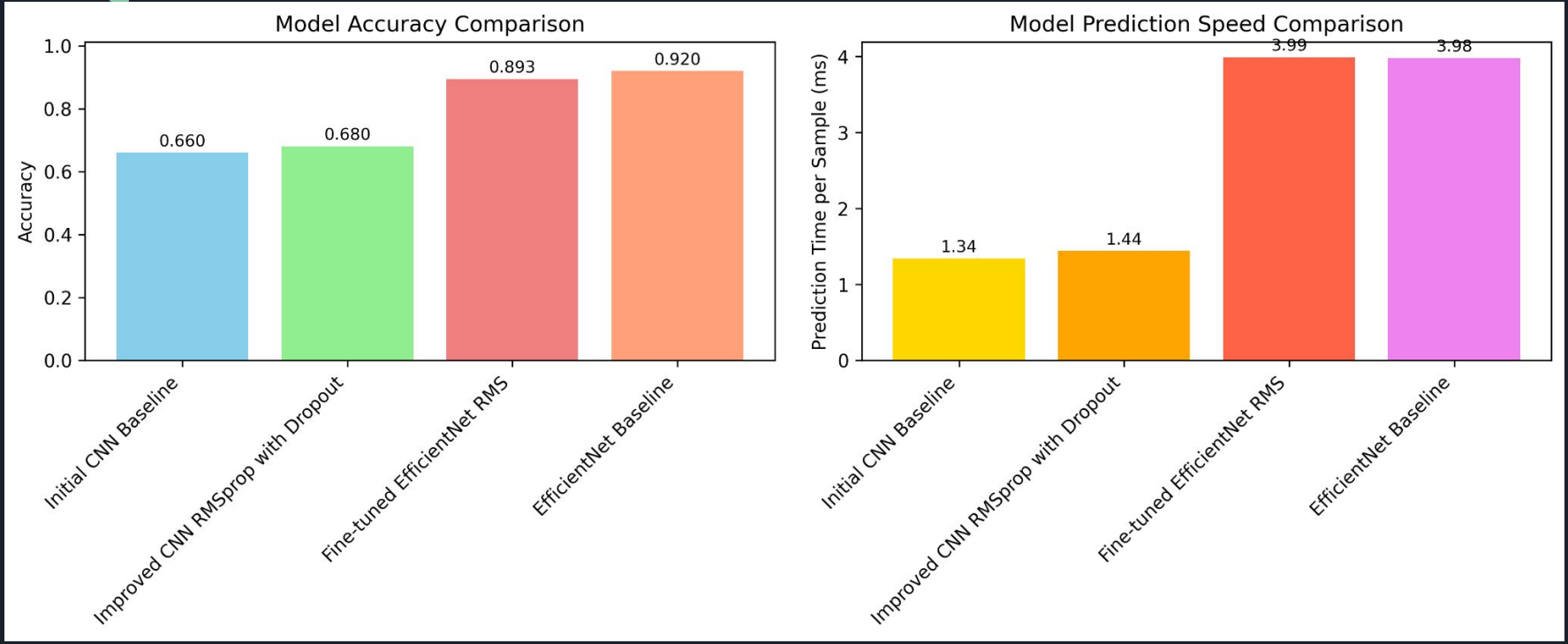


EfficientNETB0(relu): Test-accuracy 0.916



Models serialization issues

- Be careful saving and loading models between environments
  - Colab uses Keras 3.10 which sometimes isn't compatible with latest 3.12
  - This may lead you to corrupted/unreadable model file
- Don't use Lambda layers for preprocessing
  - Some preprocessing layers may not be loaded from .keras files (probably not limited to Lambda)
  - This will result to unreadable model even in the same env
- Reproducibility in Python ML libraries is complicated


Model Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.660
0.680
0.893
0.920
Initial CNN Baseline
Improved CNN RMSprop with Dropout
Fine-tuned EfficientNet RMS
EfficientNet Baseline
Model Prediction Speed Comparison
Prediction Time per Sample (ms)
4
3
2
1
0
1.34
1.44
3.99
3.98
Initial CNN Baseline
Improved CNN RMSprop with Dropout
Fine-tuned EfficientNet RMS
EfficientNet Baseline

Performance comparison

Model Name	Accuracy	Precision	Recall	F1-Score	Loss	Pred Time/ms
Initial CNN Baseline	0.6599	0.6600	0.6599	0.6590	1.0501	1.3399
Improved CNN RMSprop with Dropout	0.6799	0.6865	0.6799	0.6781	1.0629	1.4427
Fine-tuned EfficientNet RMS	0.8935	0.8935	0.8935	0.8935	0.3988	3.9895
EfficientNet Baseline	0.9196	0.9199	0.9196	0.9196	0.6340	3.9787

T - Transfer Learning
H - Hyperparameters
R - ReLu Activation
A - Augmentation
N - Neurons
K - Kernel (Convolutional)

YOU!