IMAGE CLASSIFICATION WITH DEEP LEARNING

From Simple CNNs to Advanced Transfer Learning

Julia Parnis

CIFAR-10 IMAGE CLASSIFICATION PROJECT

-

Attribution: While conducted in a team setting, **all experiments and evaluations presented here were performed by me.**

Project Overview

- ✓ **Goal:** Build an accurate image classification model for CIFAR-10 dataset
- ✓ **Dataset**: 60,000 images across 10 classes (32×32 RGB)
- ✓ **Challenge**: Improve accuracy through architecture design and optimization
- ✓ **Approach**: Systematic progression from simple to complex models

CIFAR-10 dataset

- **Total:** 60,000 images
- **Training dataset**: 50,000 images
- **Test dataset:** 10,000 images
- **Image dimensions:**
  - 32x32 pixels,
  - 3 channels (RGB)
- **10 classes:** 0-9

- Even distribution of images per each class:
  - 5,000 images per class

PREPROCESSING:

- Data was normalized:
  - *X_train = X_train/255*
  - *X_test = X_test/255*

Our Approach

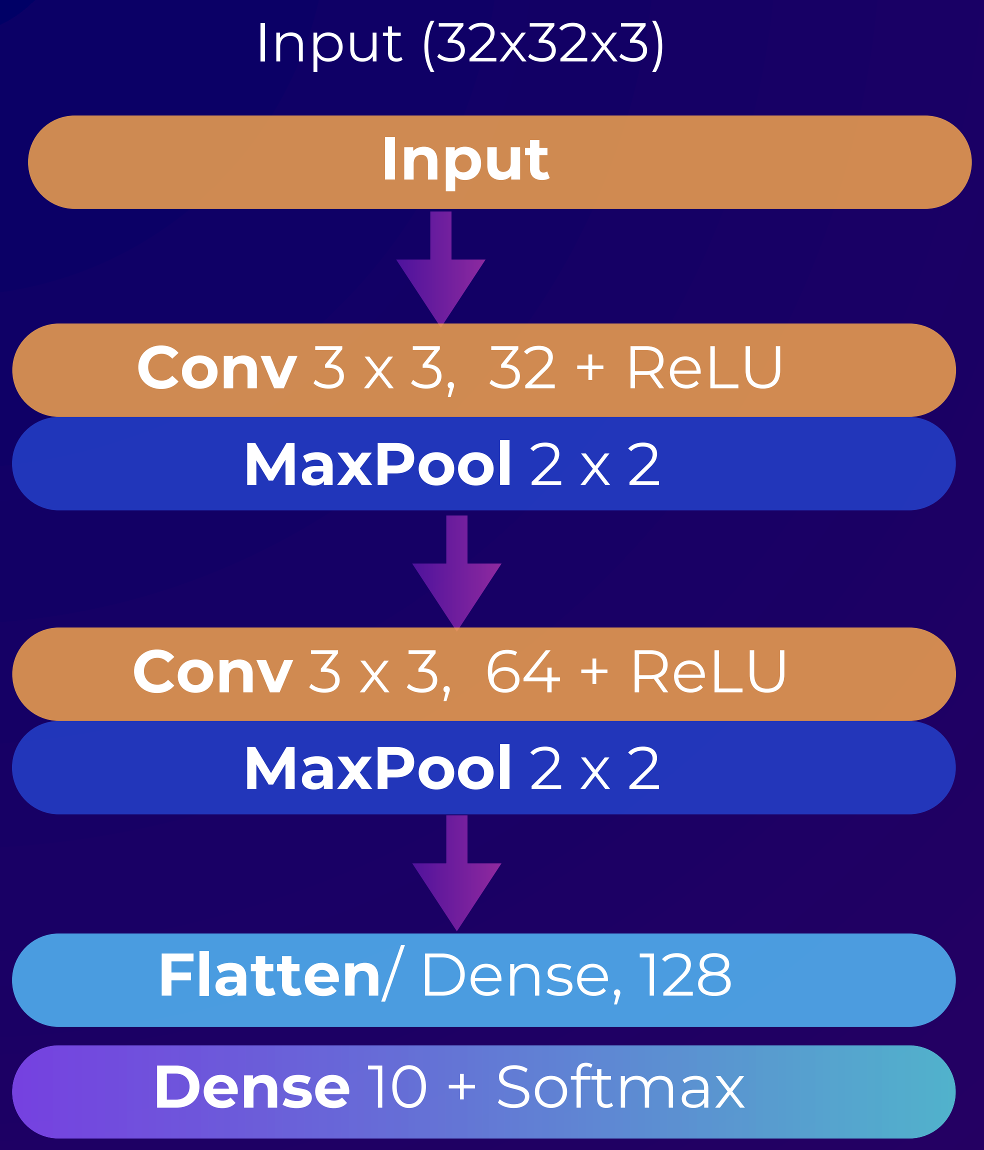
1. **Baseline CNN**
   Basic CNN model
2. **VGG Style Architecture (10 layers)**
   Deeper Feature Extraction
3. **ResNet-20 (20 layers)**
   Skip connections
4. **Transfer Learning**
   - EfficientNetV2

1

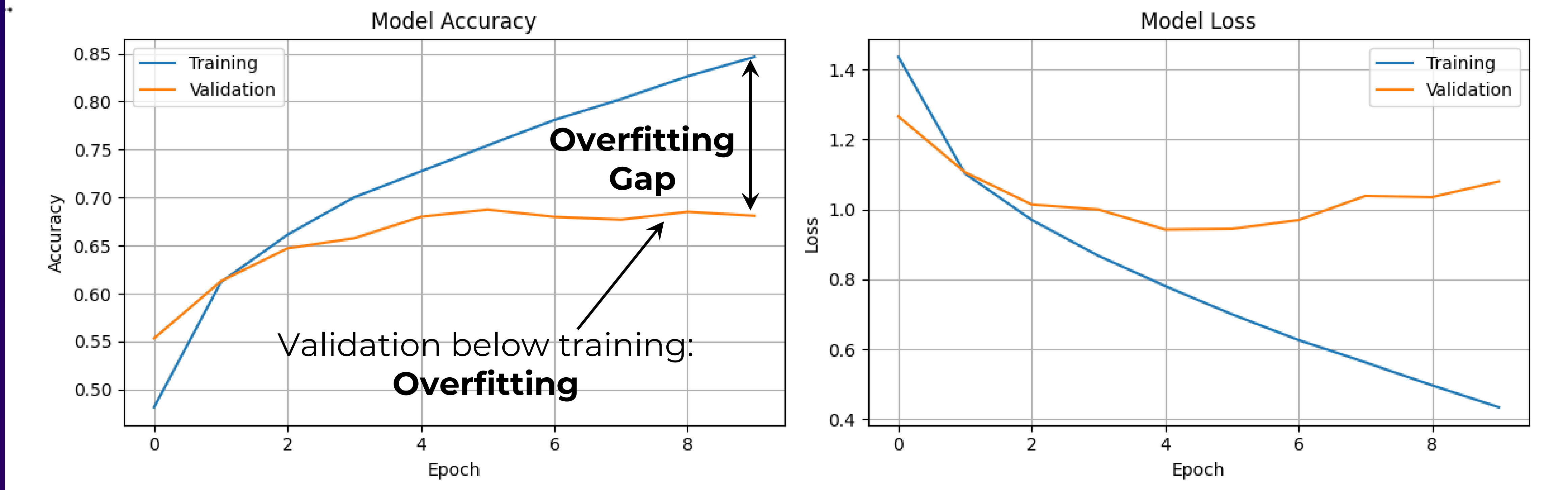
Baseline CNN

Basic CNN model

Baseline CNN Model



Model	Total Parameters	Key Modifications	Test Acc	Test Loss	Overfitting Gap	Epochs
5-layer CNN	316K	Dropout 0.2	**67.5%**	1.1	+17.8%	10



- The training and the validation datasets develop at different rates ➔ **Severe Overfitting**
- Large test loss and low accuracy ➔ **Further model improvement needed**

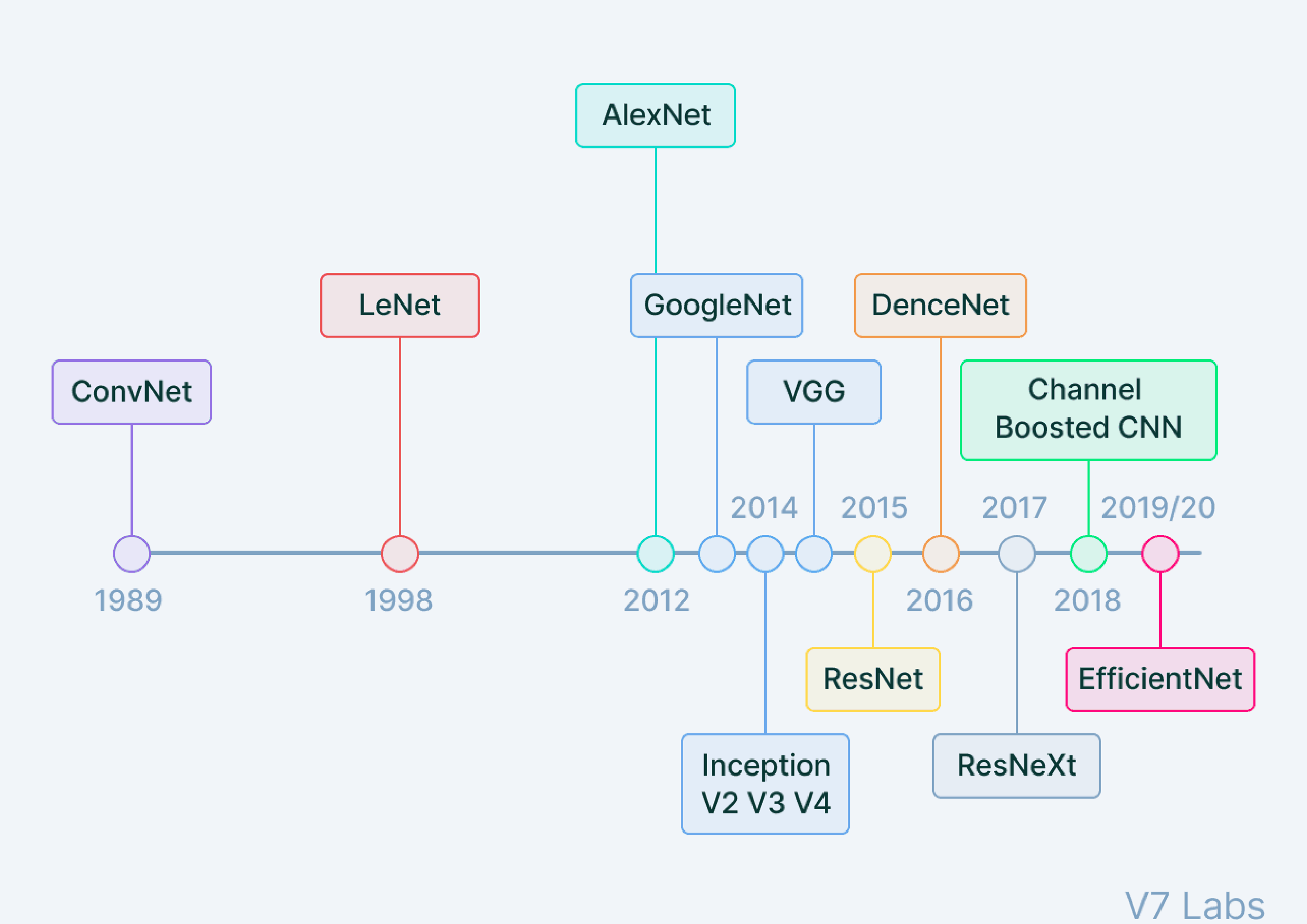
2

VGG Style Architecture (10 layers)

Deeper Feature Extraction

Why VGG Model is so Popular?

Timeline of CNN models' Development



https://www.v7labs.com/blog/convolutional-neural-networks-guide

VGG - (Visual Geometry Group, Oxford, 2014)

Key idea: *Go deeper using a simple, repeatable design*

- **Small filters:** uses 3×3 convolution filters
- **Block design:** Conv → Conv → MaxPool, repeat
- **Simple & consistent:** same building blocks throughout → easy to implement, and modify.
- **Practical impact:** became a popular ***baseline and feature extractor*** for many vision tasks

Our tweaks:

- *v* Callbacks (Early stop, ReduceLRonPlateau)
- *v* Data augmentation
- *v* Dropout

VGG-10 CNN: Deeper with Optimizations

Input (32x32x3)

Input

Conv 3 x 3, 32 + ReLU (same)
Conv 3 x 3, 32 + ReLU (same)
MaxPool 2 x 2

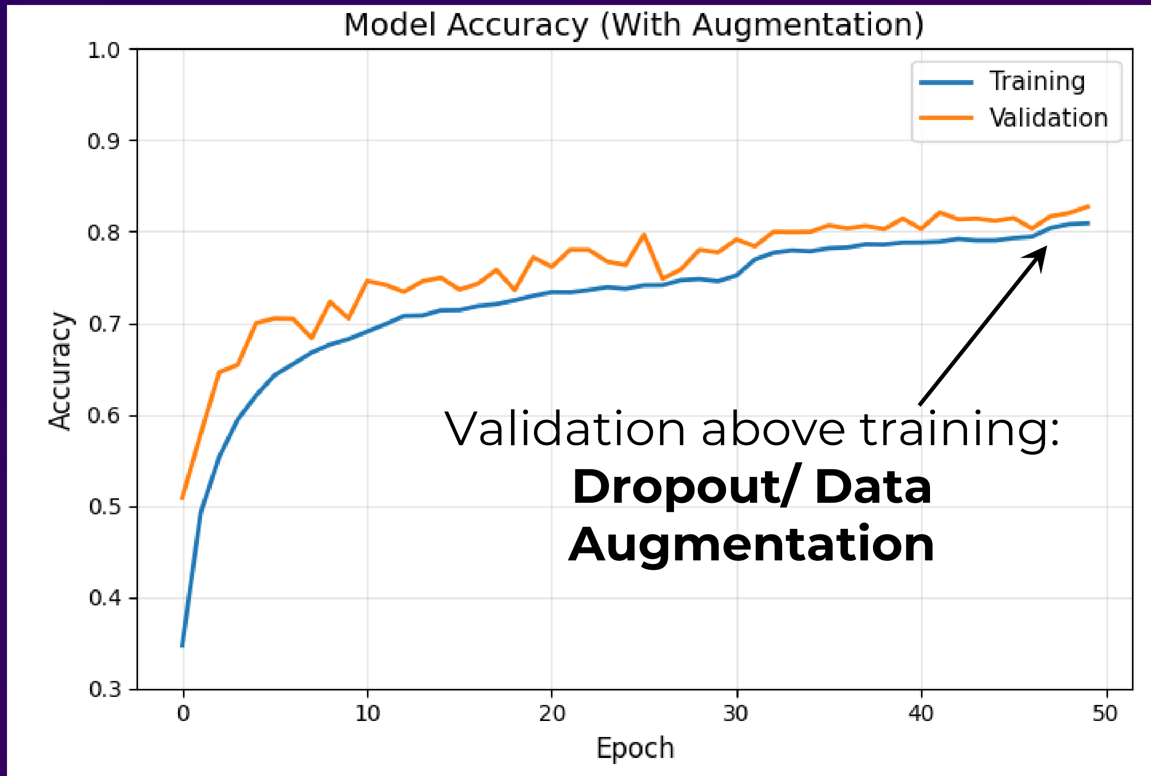
Conv 3 x 3, 64 + ReLU (same)
Conv 3 x 3, 64 + ReLU (same)
MaxPool 2 x 2

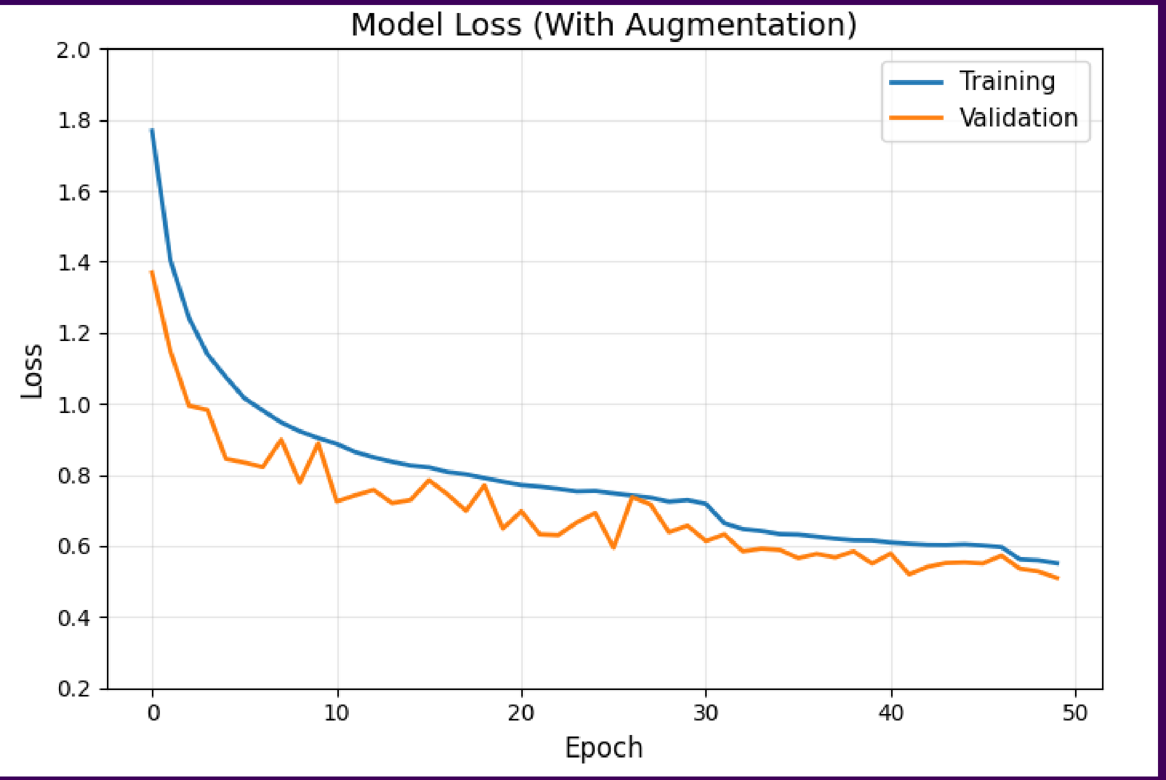
Flatten/ Dense, 128

Dense 10 + Softmax

Model	Total Parameters	Key Modifications	Test Acc	Test Loss	Overfitting Gap	Epochs
5-layer CNN	316K	Dropout 0.2	67.5%	1.1	+17.8%	10
VGG10-base	403K	Dropout 0.2	78.6%	0.6	+2.8%	20
VGG10+callbacks	403K	+ early stop, ReduceLR	80.5%	0.6	+7.4%	25/50
VGG10-optimized	403K	All+Data Augmentation	**82.1%**	0.5	-1.8%	50/50

- The training and the validation datasets develop together ➔ **Model is learning efficiently**
- Deeper architecture + change in learning rate + data augmentation gradually improved model efficiency





3

ResNet-20 (20 layers)

Skip connections

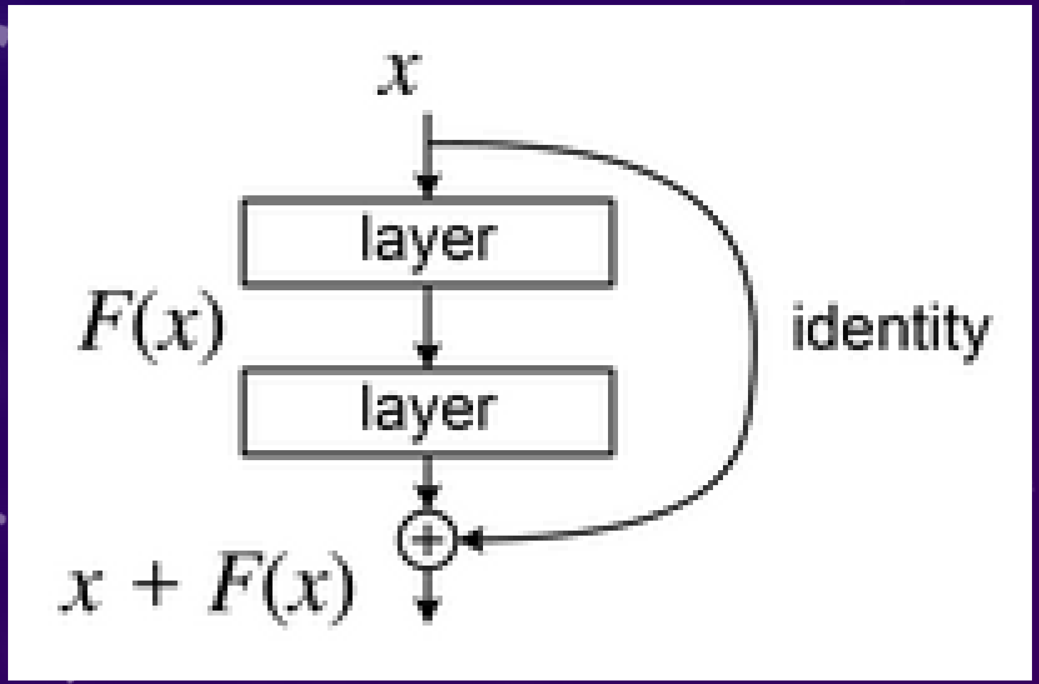
ResNet-20: Solving the Depth Problem

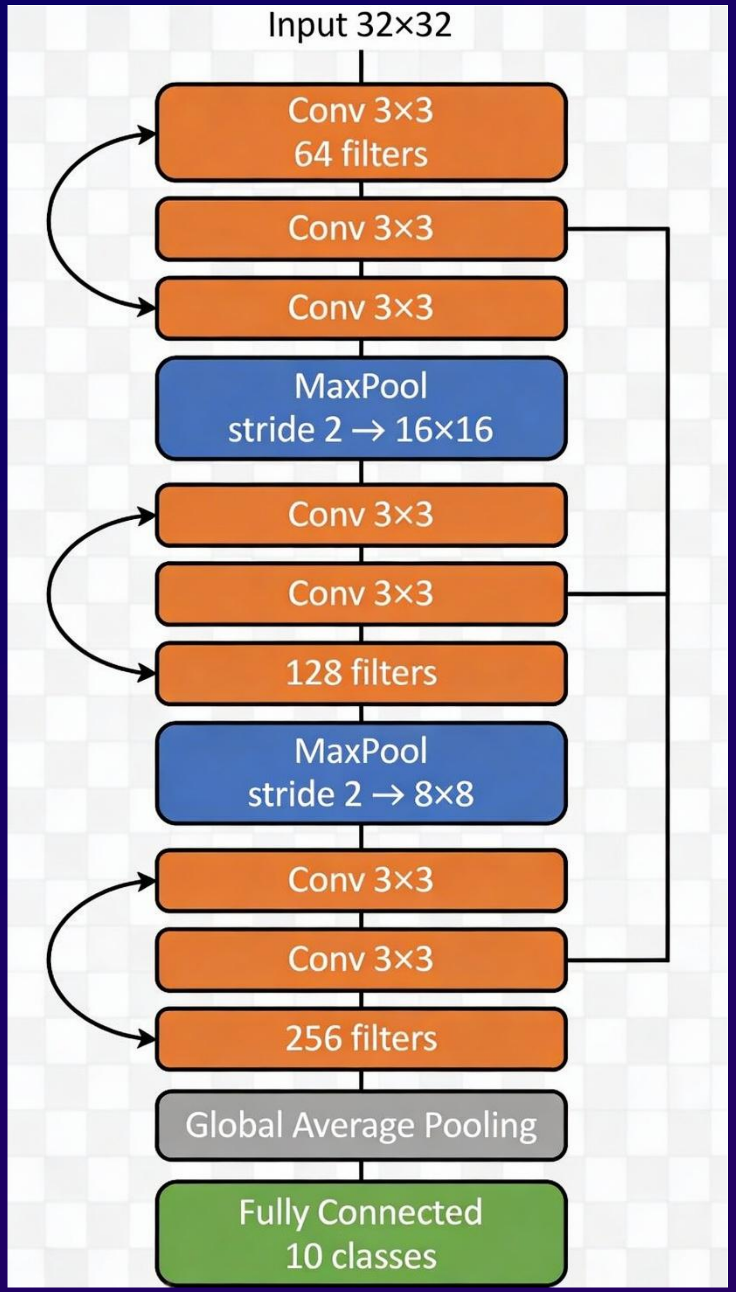
ResNet (2015) - “Residual Learning”

Key idea: skip connections learn a residual F(x) so the block outputs x + F(x)

- **Paper:** Deep Residual Learning for Image Recognition (He et al., 2015)
- **Problem:** very deep nets started to train worse (“degradation” / vanishing gradients)

```python
# Save input for skip connection
shortcut = x

# First conv layer
x = Conv2D(filters, (3,3), strides=stride, padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

# Second conv layer
x = Conv2D(filters, (3,3), strides=1, padding='same')(x)
x = BatchNormalization()(x)

# Adjust shortcut if dimensions changed
if stride != 1 or shortcut.shape[-1] != filters:
    shortcut = Conv2D(filters, (1,1), strides=stride, padding='same')(shortcut)
    shortcut = BatchNormalization()(shortcut)

# Add skip connection
x = Add()([x, shortcut])
x = Activation('relu')(x)

return x
```

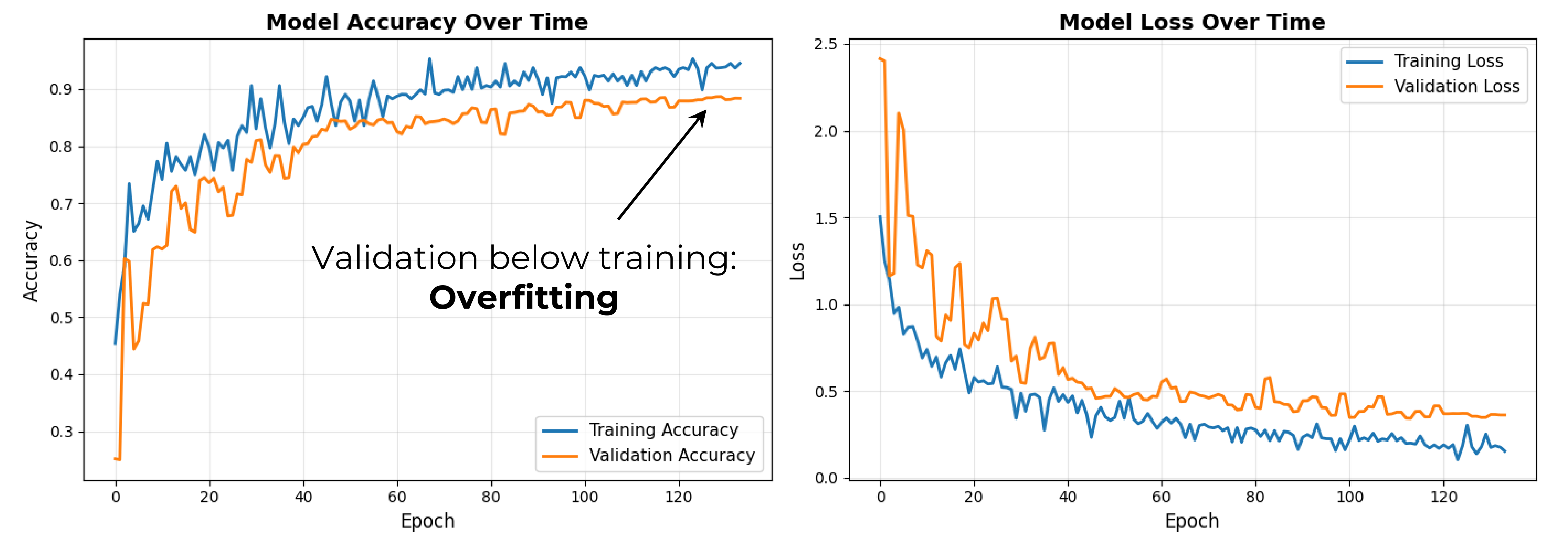




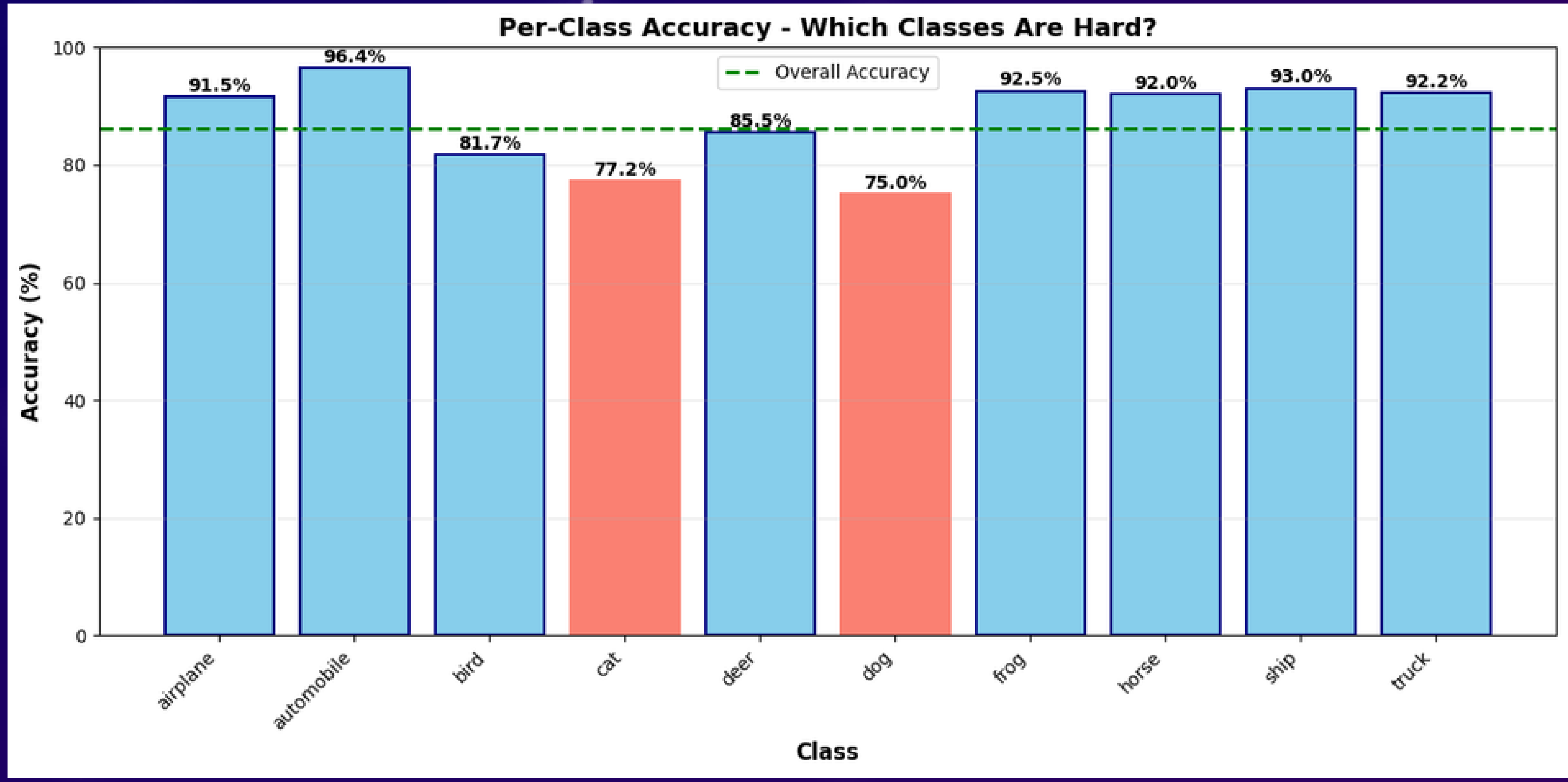
ResNet-20 CNN Model

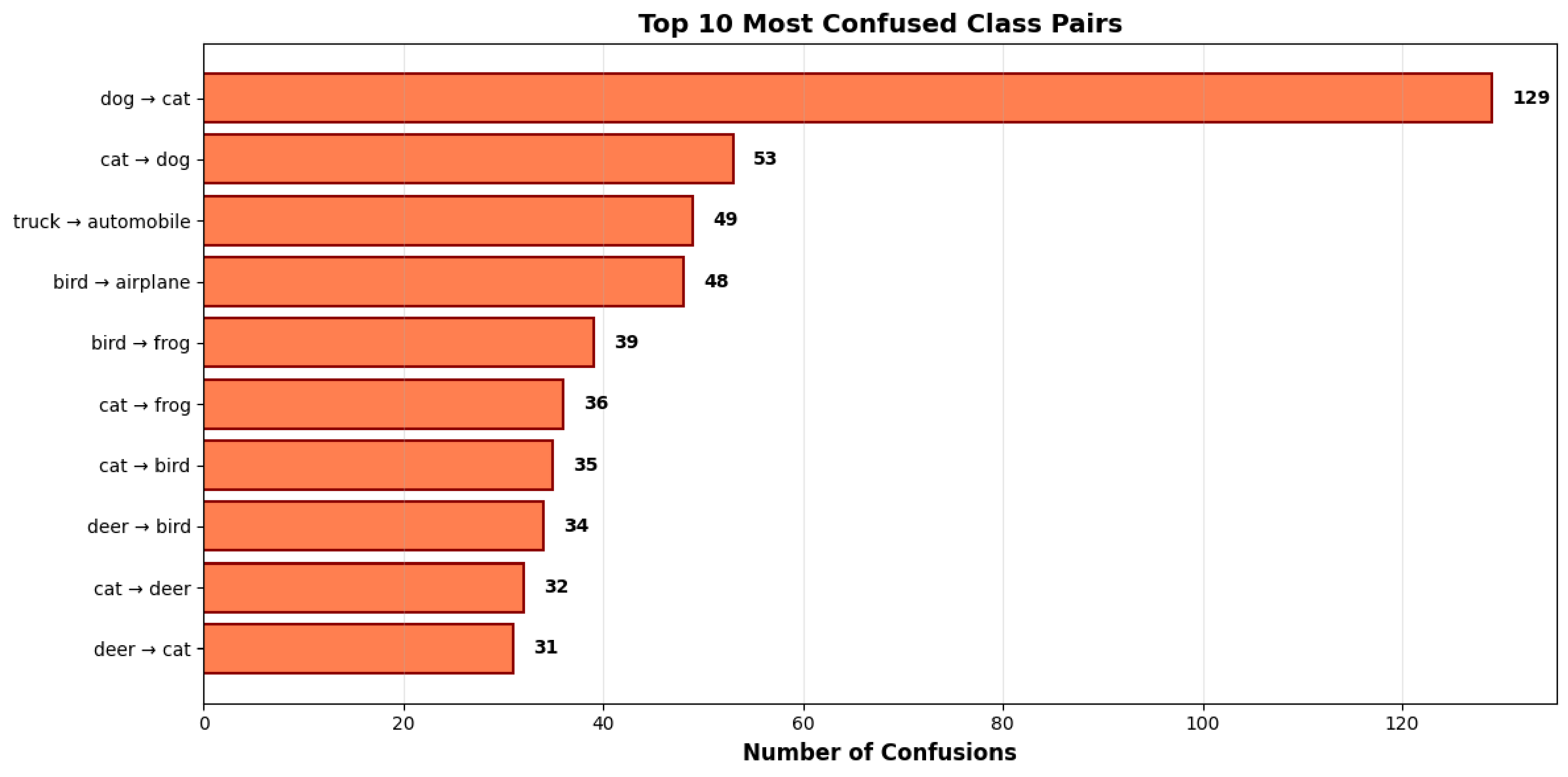
Model	Total Parameters	Key Modifications	Test Acc	Test Loss	Overfitting Gap	Epochs
5-layer CNN	316K	Dropout 0.2	67.5%	1.1	+17.8%	10
VGG10-base	403K	Dropout 0.2	78.6%	0.6	+2.8%	20
VGG10+callbacks	403K	+ early stop, ReduceLR	80.5%	0.6	+7.4%	25/50
VGG10-optimized	403K	All+Data Augmentation	82.1%	0.5	-1.8%	50/50
ResNet-20	275K	All previous, no Dropout	**87.7%**	0.4	+3.1%	114/200

- The training and the validation datasets develop together ➔ **Model is learning efficiently**
- Skip connections approach: ResNet-20 further improved model efficiency



ResNet-20: Evaluation Dashboard





Misclassified Examples: Where the model went wrong?

True: bird
Pred: airplane (89.1%)

True: frog
Pred: bird (48.5%)

True: dog
Pred: horse (68.8%)

True: dog
Pred: deer (56.7%)

True: bird
Pred: dog (31.6%)

True: bird
Pred: ship (56.8%)

4 Transfer Learning

EfficientNetV2

EfficientNet: Smarter scaling for better accuracy–compute trade-off

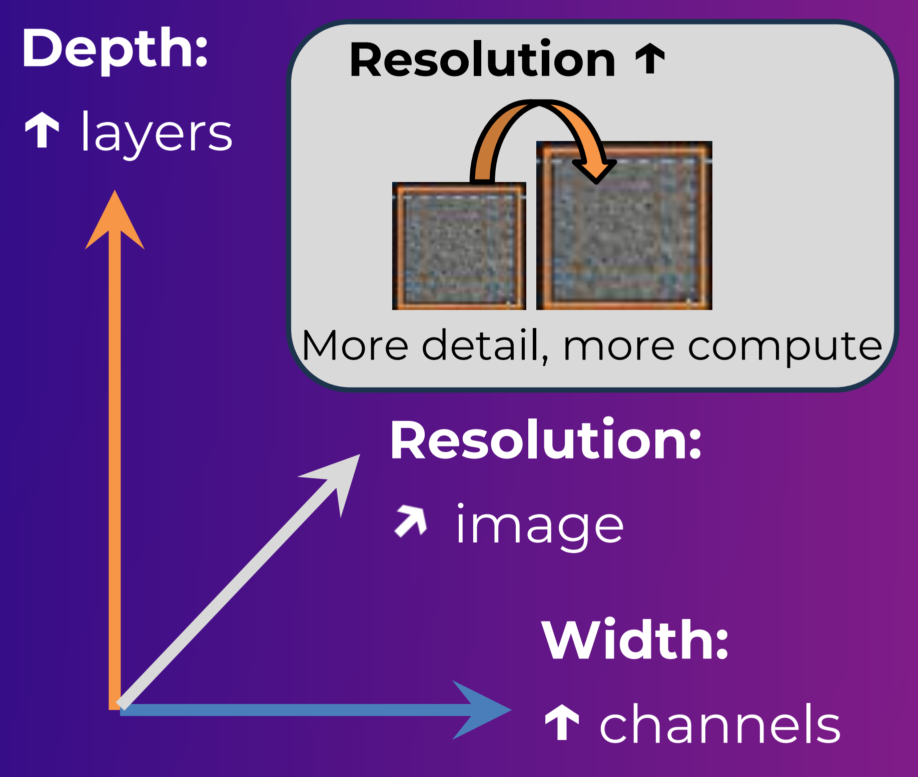
EfficientNet (2019) – Core idea

Tan & Le (2019). **"EfficientNet: Rethinking Model Scaling for CNNs."** ICML 2019

- **Goal:** high accuracy with less compute (fewer parameters / fewer FLOPs)
- **Key idea:** compound scaling (scale depth + width + resolution together)
- **B0–B7:** B0 is the baseline; B1–B7 are scaled versions using the same rule

Compound Scaling:

Depth + Width + Resolution



Stronger accuracy – efficiency balance

EfficientNetV2 (2021): faster training

- **Fused blocks:** faster early layers
- **Progressive learning:** small → bigger images
- **Training speed**: faster training overall

Transfer Learning Pipeline

Input: 32 x 32 x 3

Resize: 224 x 224 x 3

Phase 1: Feature extraction
Freeze backbone – train only head

Backbone:
EfficientNetV2B0 (ImageNet)

Pooling: Global Average Pool

Phase 2 Fine-tuning
Unfreeze last N layers – train

Head:
BN→Dropout→Dense(256)→BN →Dropout

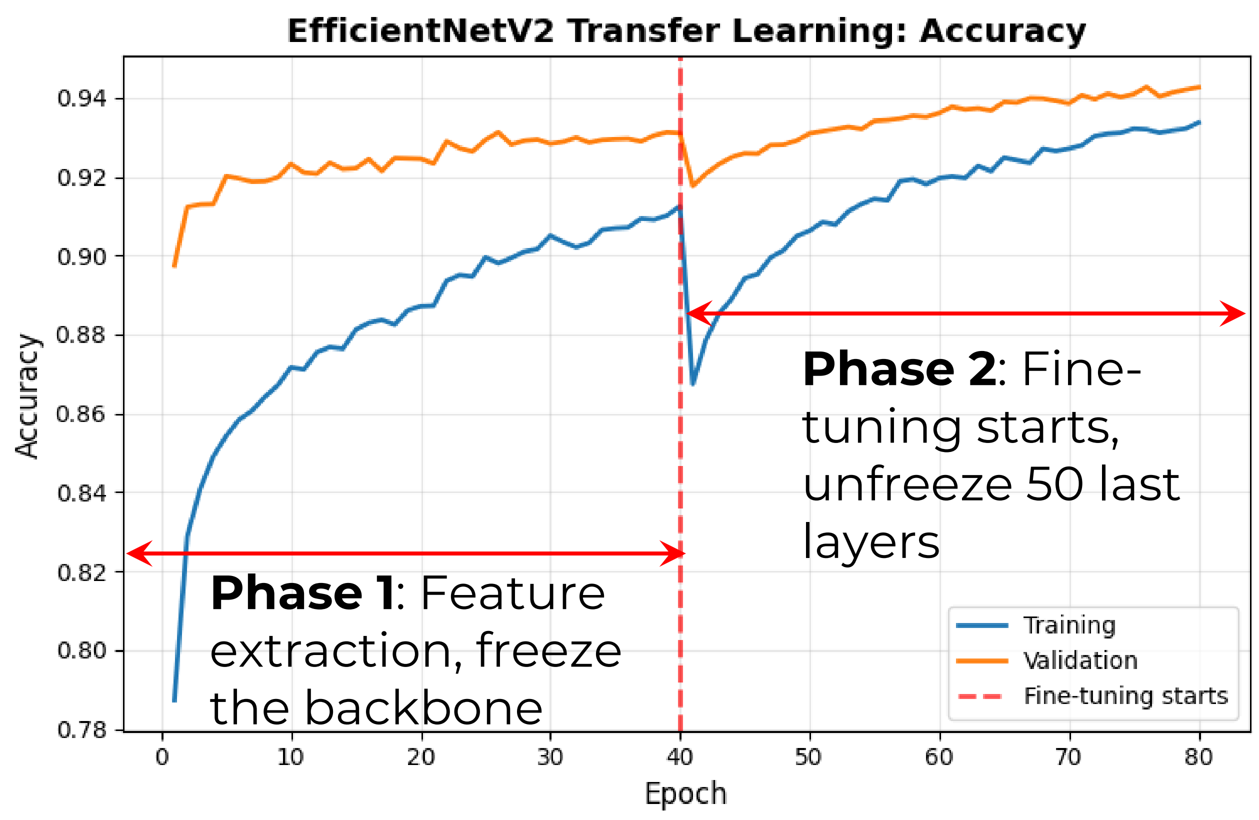
Output: Dense (10) (softmax)

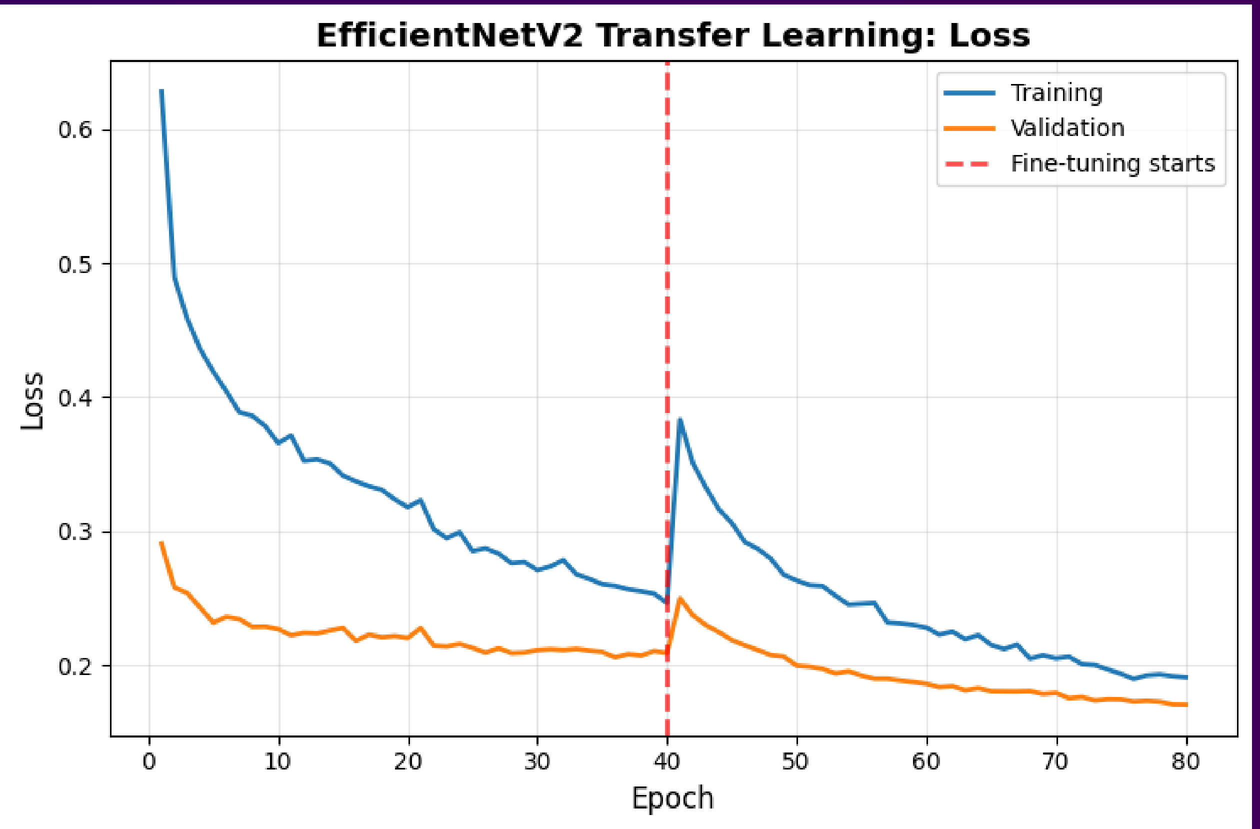
Total params 6.26M |
Trainable 334K (Phase 1)

Transfer learning gave a strong jump in accuracy without training from scratch.

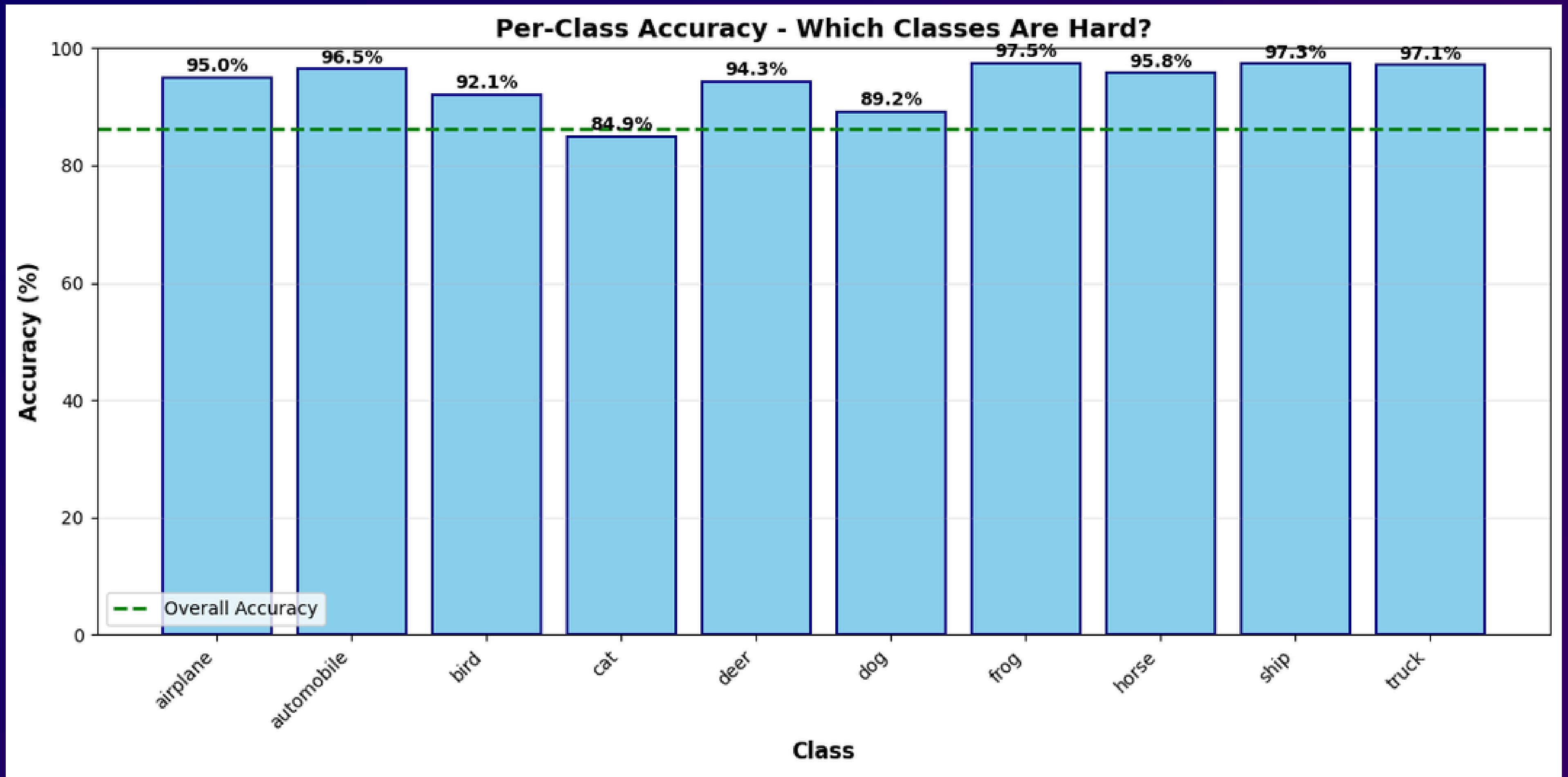
EfficientNetV2B0 CNN Model

Model	Total Parameters	Key Modifications	Test Acc	Test Loss	Overfitting Gap	Epochs
5-layer CNN	316K	Dropout 0.2	67.5%	1.1	+17.8%	10
VGG10-optimized	403K	+ Early Stop, Reduced LR, Data Augmentation	82.1%	0.5	-1.8%	50/50
ResNet-20	275K	All previous, no Dropout	87.7	0.4	+3.1%	114/200
EfficientNetV2B0	6.26M	All+, Input: 96x96, 30 layers unfrozen	90.4%	0.3	-4.5%	30+30
EfficientNetV2B0	6.26M	All+, Input: 224x224, 50 layers unfrozen (25%)	94.0	0.2	0.0%	40+40



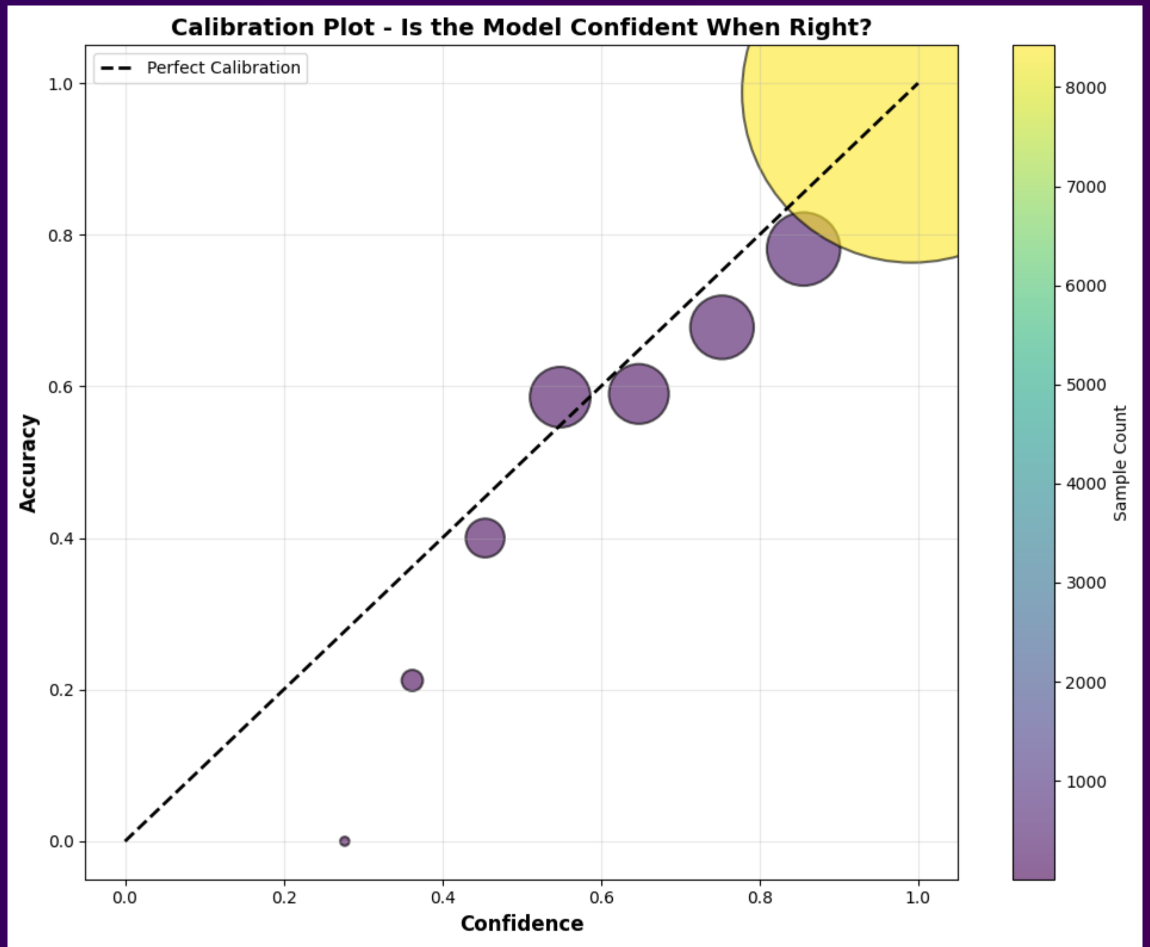


EfficientNetV2: Evaluation Dashboard

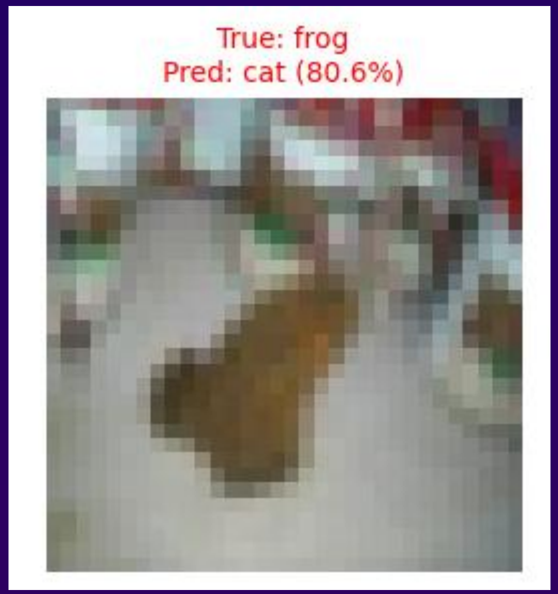


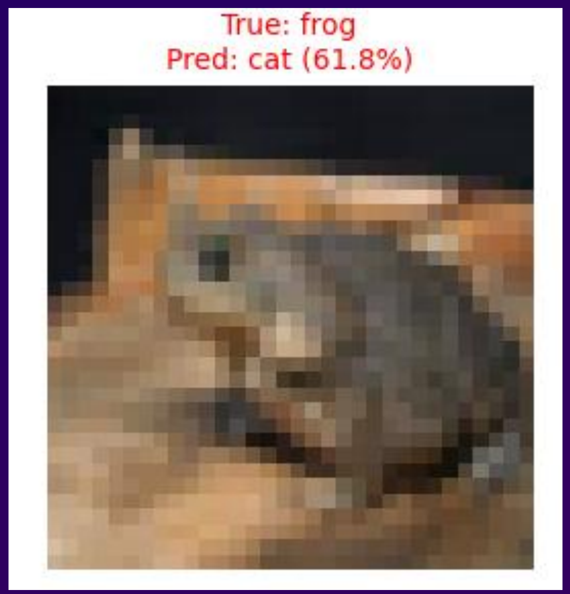
Top 10 Most Confused Class Pairs:

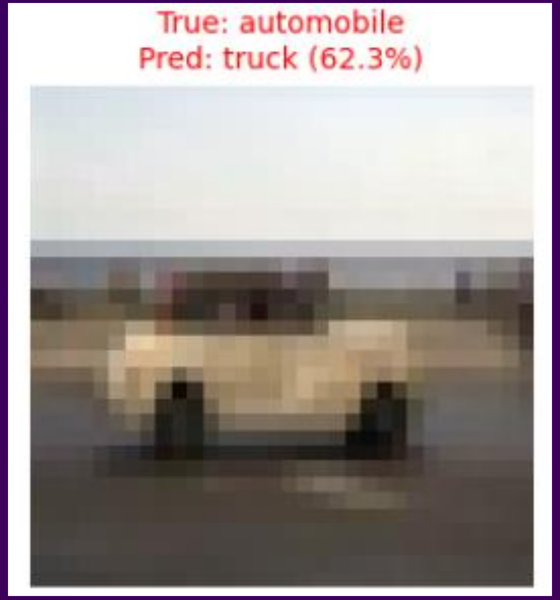
1. Dog→Cat 73
2. Cat→Dog 70
3. Automobile→Truck 27
4. Cat→Frog 25
5. Cat→Deer 21
6. Airplane→Ship 20
7. Bird→Deer 19
8. Truck→Automobile 19
9. Deer→Horse 18
10. Horse→Deer 18



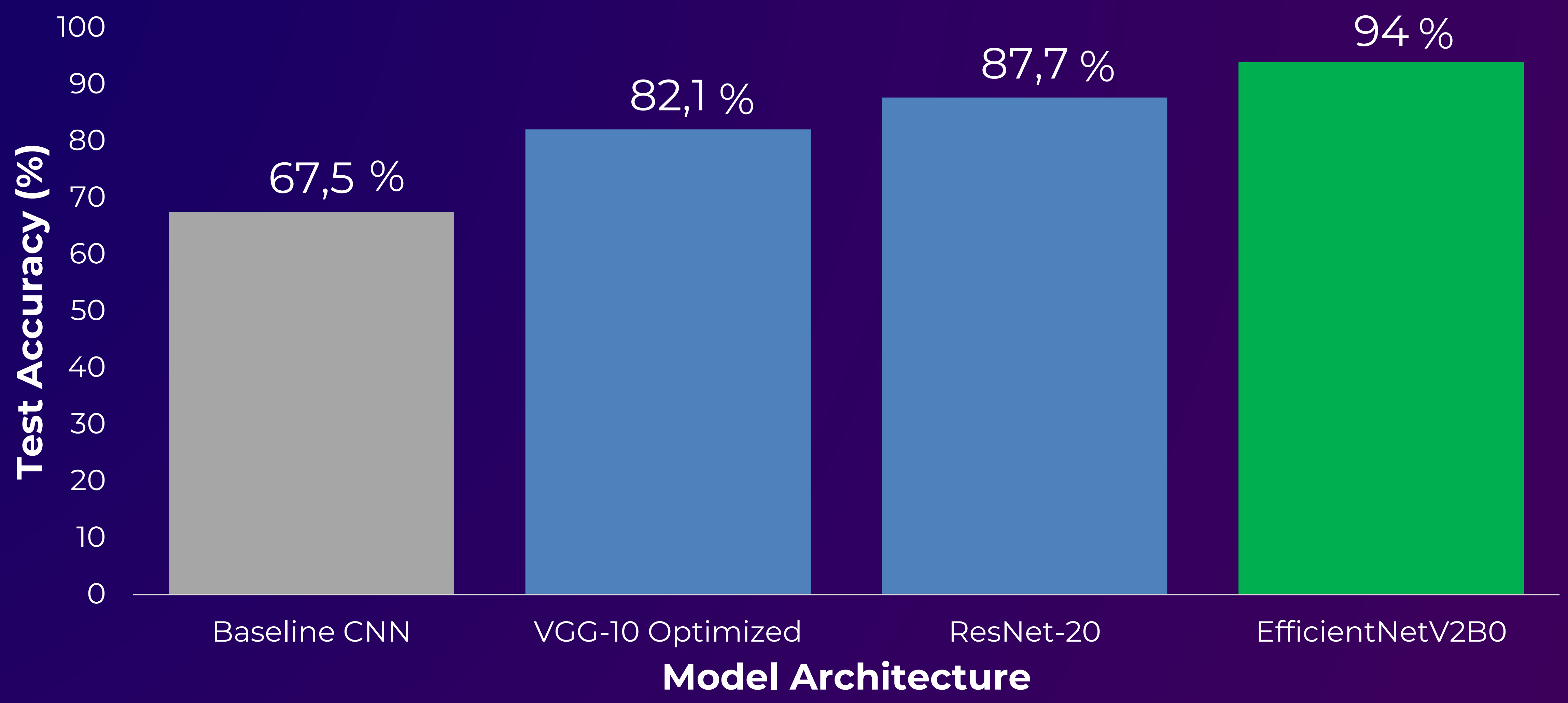
Misclassified Examples: Where the model went wrong?







Conclusions



Best Model

Best model: EfficientNetV2B0 (transfer learning)

Test accuracy: 94.0% | **Lowest test loss**

Two-phase training: **freeze → fine-tune (last 50 layers)**

What improved performance?

VGG: deeper conv blocks → big jump vs baseline

ResNet-20: skip connections → more stable training

EfficientNetV2B0: transfer learning → largest gain with fewer trainable weights

Error patterns & next steps

Confusion mostly between **similar classes** (dog↔cat, automobile↔truck)

Next steps :

- stronger augmentation (random crop/ flip etc.,)
- tune fine-tuning depth (unfreeze fewer/more layers)