IMAGE CLASSIFICATION WITH DEEP LEARNING

From Simple CNNs to Advanced Transfer Learning

Julia, Maryem and Helena

CIFAR-10 IMAGE CLASSIFICATION PROJECT

-

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

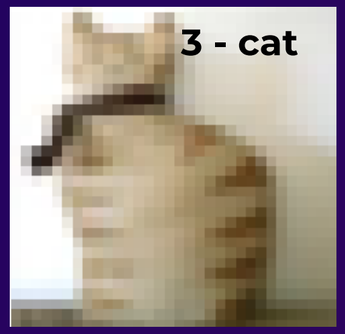
CIFAR-10 dataset

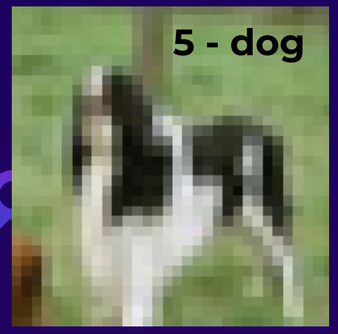
- **Total:** 60,000 images
- **Training dataset**: 50,000 images
- **Test dataset:** 10,000 images
- **Image dimensions:**
  - 32x32 pixels,
  - 3 channels (RGB)
- **10 classes:** 0-9

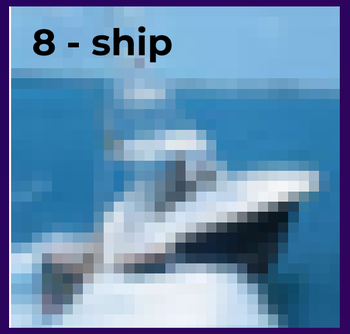
- Even distribution of images per each class:
  - 5,000 images per class

PREPROCESSING:

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







Our Approach

1. **Baseline CNN**
   From basic CNN to a 4 block deep model
2. **VGG Style Architecture (10 layers)**
   Deeper Feature Extraction
3. **ResNet-20 (20 layers)**
   Skip connections
4. **Transfer Learning**
   - ResNet-50 - InceptionV3 -DenseNetV2 -EfficientNetV2

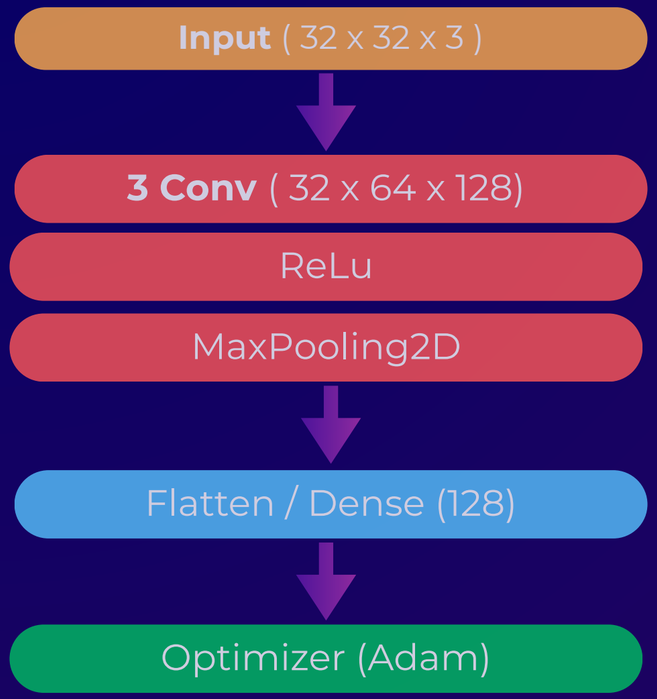
CNN from scratch

FROM BASIC CNN TO DEEPER MODELS

Extra - Preprocessing

- **Converting Numpy arrays to Tensorflow dataset**
- **Shuffle :** Randomly mixes training samples every epoch (only training split)
- **Batch :** Groups individual samples into mini-batches, instead of loading everything at once
- **Prefetch :** Prepares the next batch while the model is training on the current one

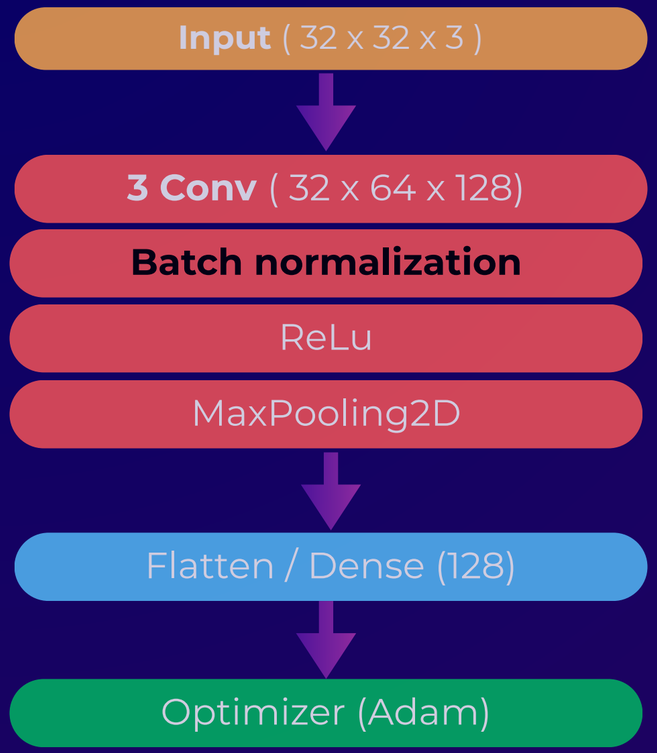
Base Model



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%

- Performance gap between train and test is indicating an overfitting
- Low test and validation , the model can’t generalize effectively

Model with batch normalization

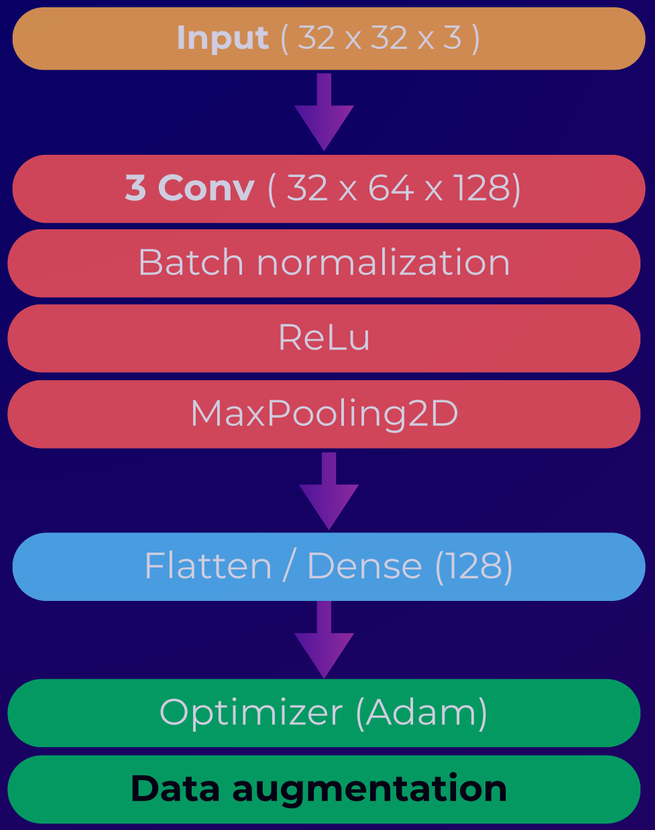


Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%

- Instead of using mean and variance of the hole dataset we use normalized mini batch outputs (mean ≈ 0, variance ≈ 1)
- Add slight noise to ReLu → reduce overfitting.

- Faster and more stable training
- generalization has improved
- overfitting still present

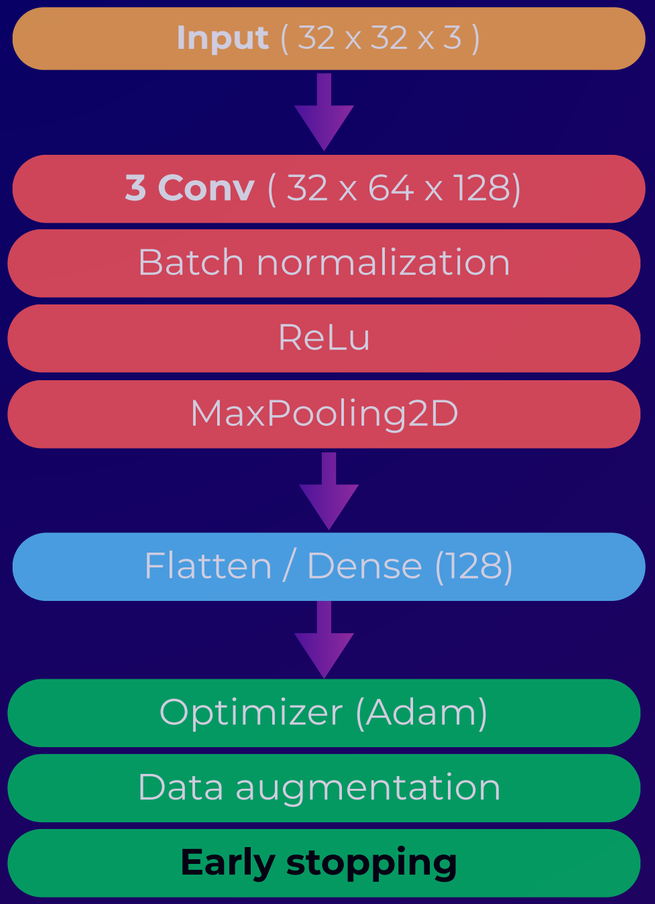
Model with data augmentation



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%

- Data augmentation successfully reduced overfitting
- Test accuracy is still low but we observed that it peaked at epoch 17 and then dropped

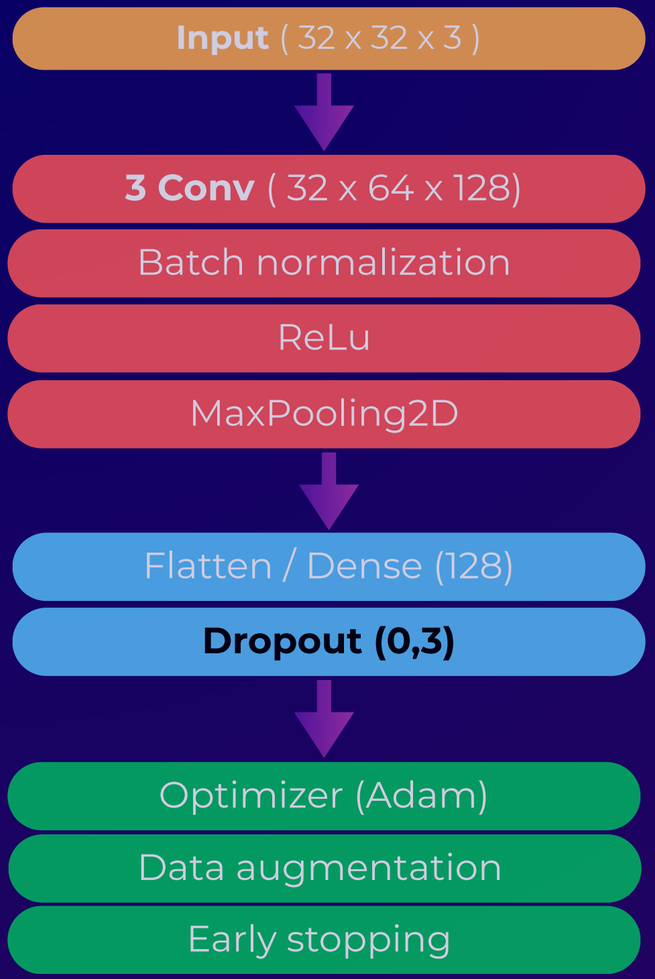
Model with early stopping



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%
+ Early stopping	87%	79%	82%

- The model stopped at epoch 18, preventing further overfitting.
- Early stopping allowed the model to capture useful patterns without overtraining
- Further reducing overfitting and improving generalization

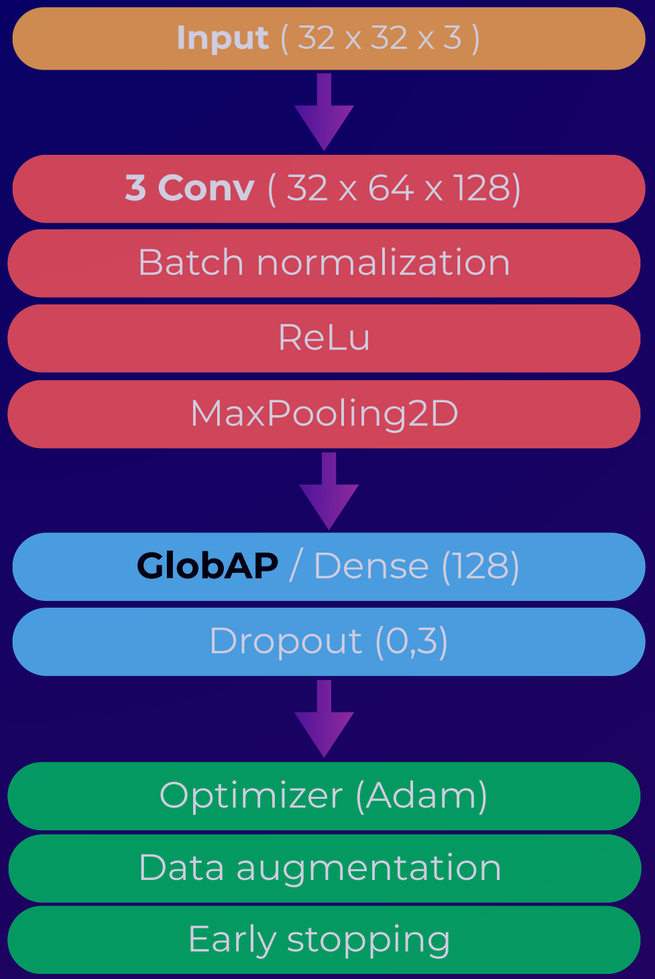
Model with dropout



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%
+ Early stopping	87%	79%	82%
+ Dropout	74%	66%	71%

- The dropout reduced overfitting aggressively, but in this case it caused underfitting
- Lower generalization capacity
- we will keep this change in order to prevent overfitting in the next experiments

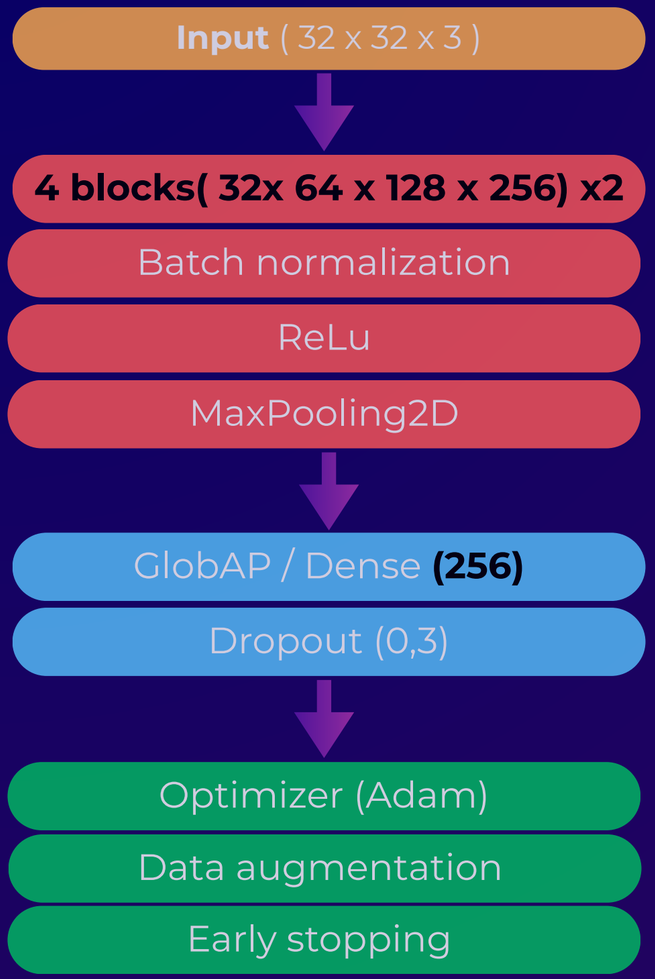
Model with global average pooling 2D



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%
+ Early stopping	87%	79%	82%
+ Dropout	74%	66%	71%
+ GlobalaveragePool	73%	63%	72%

- Flattening increases the number of trainable parameters, which can lead to overfitting.
- Global Average Pooling (GAP) summarizes each feature map into a single value
- Lower generalization capacity
- Underfitting still present
- GAP may be more suitable for larger datasets or deeper models

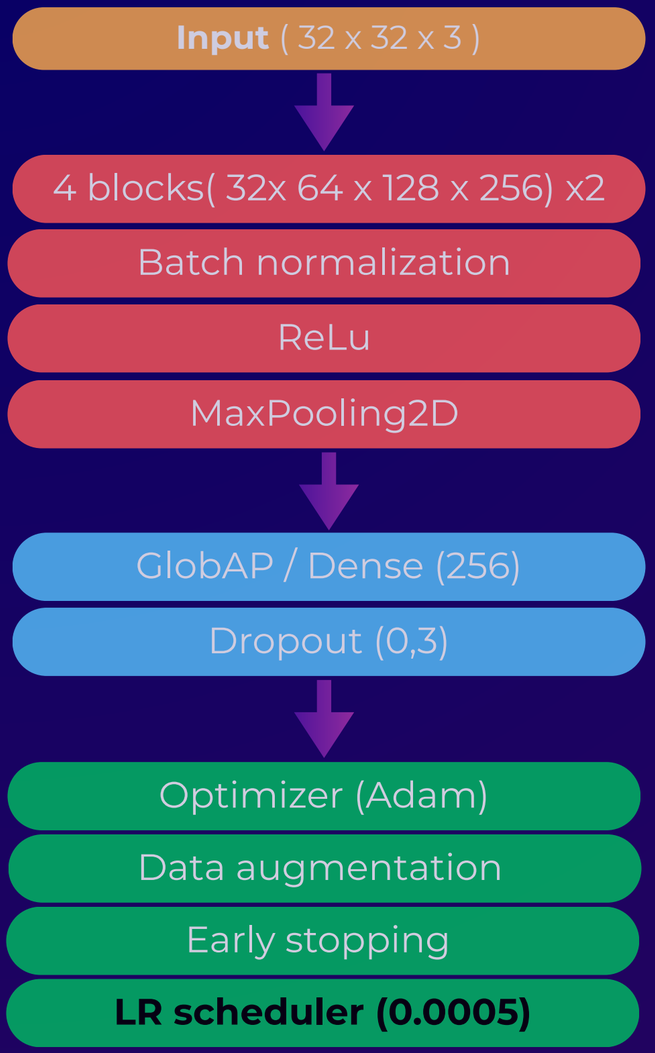
Deeper model : 4 blocks



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%
+ Early stopping	87%	79%	82%
+ Dropout	74%	66%	71%
+ GlobalaveragePool	73%	63%	72%
+ Deeper model	90%	86%	86%

- Best generalization capacity so far .
- Moderate gap between training and test

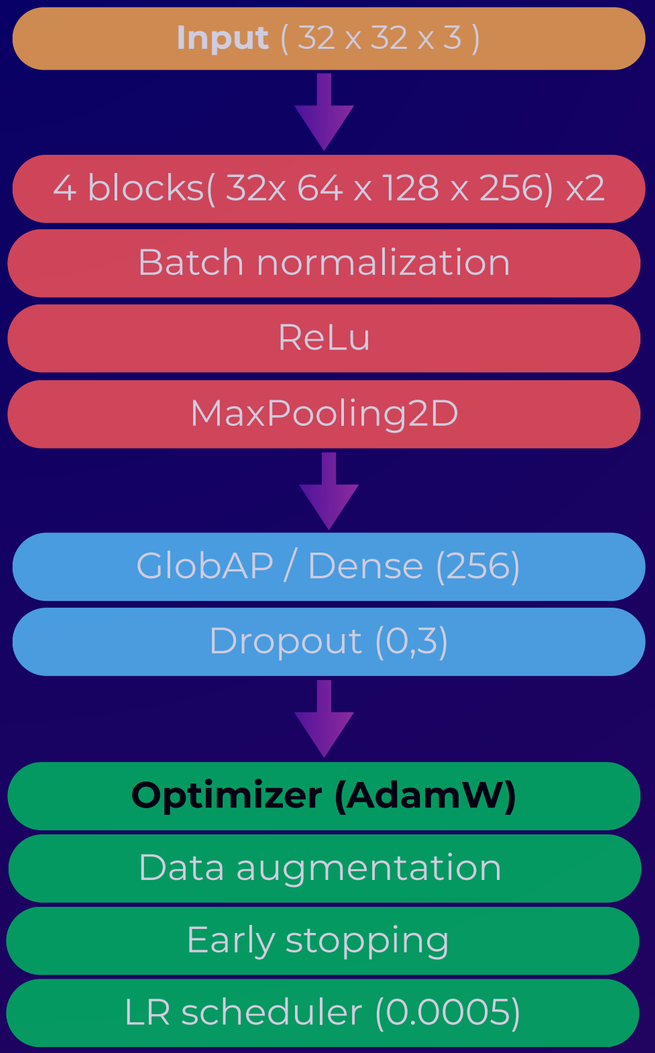
Model with LR scheduler



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%
+ Early stopping	87%	79%	82%
+ Dropout	74%	66%	71%
+ GlobalaveragePool	73%	63%	72%
+ Deeper model	90%	86%	86%
+ LR Scheduler	92%	88%	88%

- A learning rate scheduler (`ReduceLROnPlateau`) was added to dynamically adjust the optimizer's learning rate during training
- While the 4-block model achieved strong performance, training with a fixed learningrate prevented the model from fully converging.
- Better feature refinement without increasing overfitting

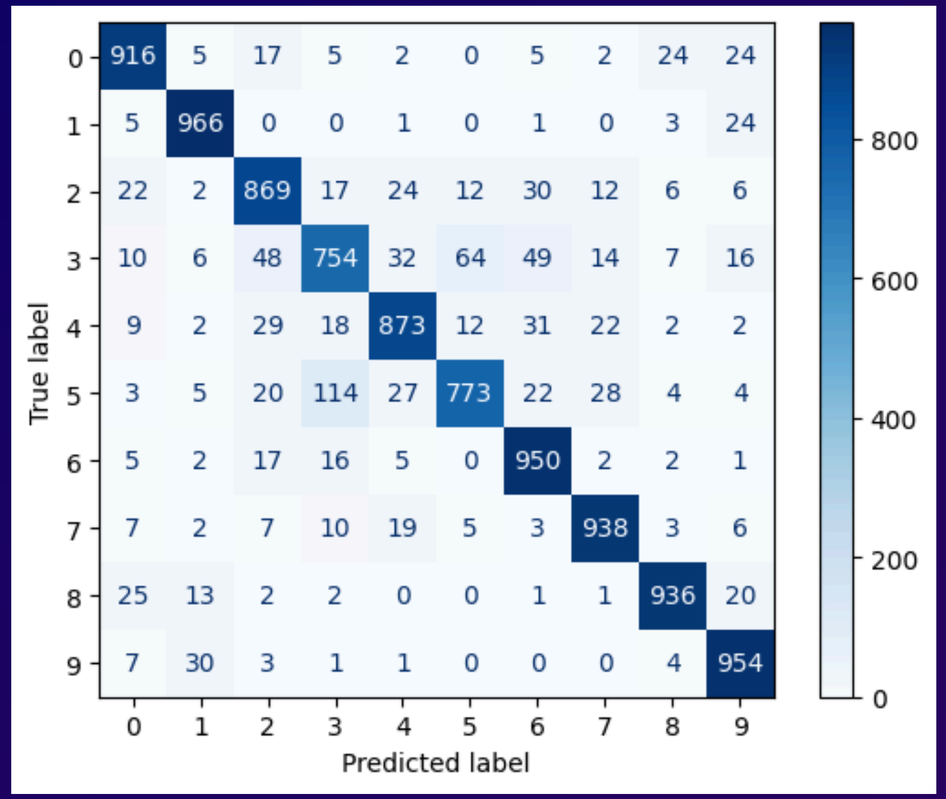
Model with AdamW



Model	Train Acc	Val Acc	Test Acc
Base	93%	73%	73%
+ Batch normalization	98%	76%	75%
+ Data augmentation	83%	78%	78%
+ Early stopping	87%	79%	82%
+ Dropout	74%	66%	71%
+ GlobalaveragePool	73%	63%	72%
+ Deeper model	90%	86%	86%
+ LR Scheduler	92%	88%	88%
+ AdamW	95%	89%	90%

- Adam was making large weights to fit the training data which caused overfitting and worse generalization as AdamW that applies weights decay , a penalty to the loss function to discourage large weights
- We decide to stop the training at this level

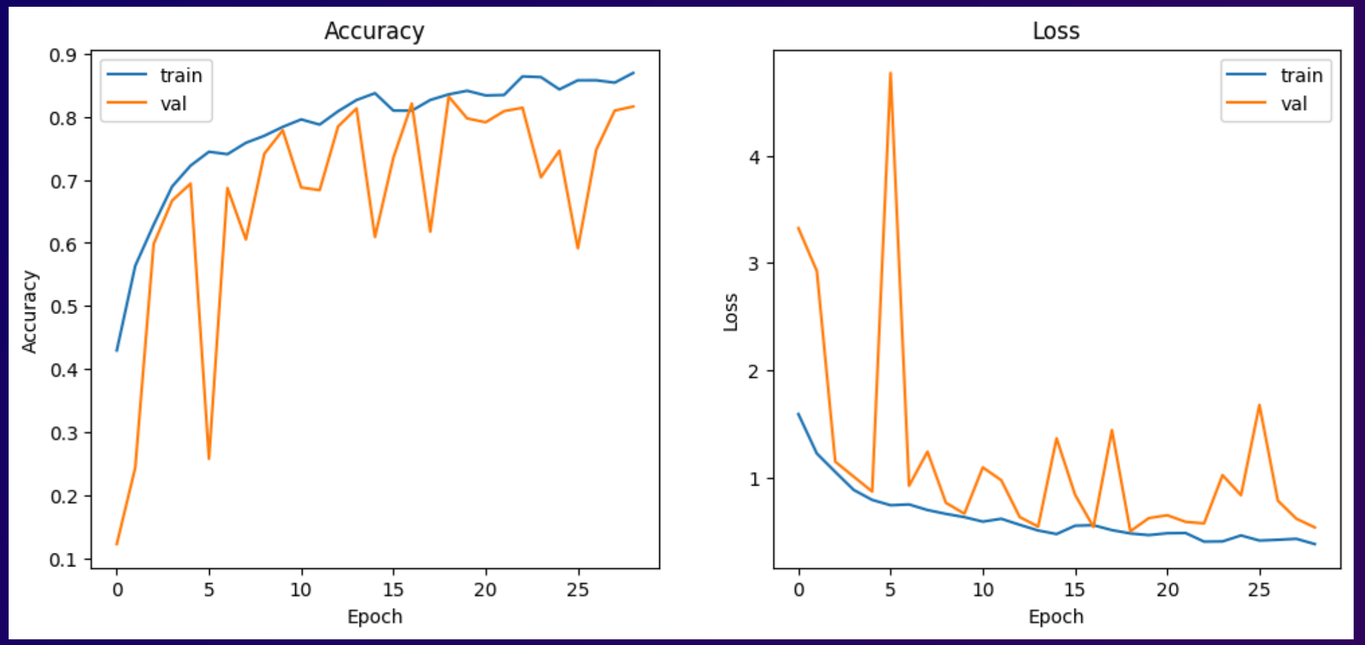
Model with AdamW - Evaluation



- Misclassification occurs between class 3 and 5, which explains lower recall (0.75 and 0.77)
- Most classes show high precision and recall (>0.87)

	precision	recall	f1-score	support
0	0.91	0.92	0.91	1000
1	0.94	0.97	0.95	1000
2	0.86	0.87	0.86	1000
3	0.80	0.75	0.78	1000
4	0.89	0.87	0.88	1000
5	0.89	0.77	0.83	1000
6	0.87	0.95	0.91	1000
7	0.92	0.94	0.93	1000
8	0.94	0.94	0.94	1000
9	0.90	0.95	0.93	1000

Model with AdamW - Evaluation



- **By the end of the training:**
  - a good balance between validation and training

- **During the training**
  - Strong fluctuations in the validation, and stable improvement in training due to
    - Training computed batch by batch / validation computed epoch by epoch
    - Small validation set only 1K image per class , even a small weight change can create fluctuation

CNN

CNN scratch

VGG STYLE ARCHITECTURE (10 LAYERS)

Our VGG-10 CNN: Deeper with Optimizations

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:

- ✓ Callbacks (Early stop, ReduceLROnPlateau)
- ✓ Data augmentation
- ✓ Dropout

Baseline - VGG10
78.6% |Loss: 0.64
20 epochs

Callbacks
80.5% |Loss: 0.64
20 epochs

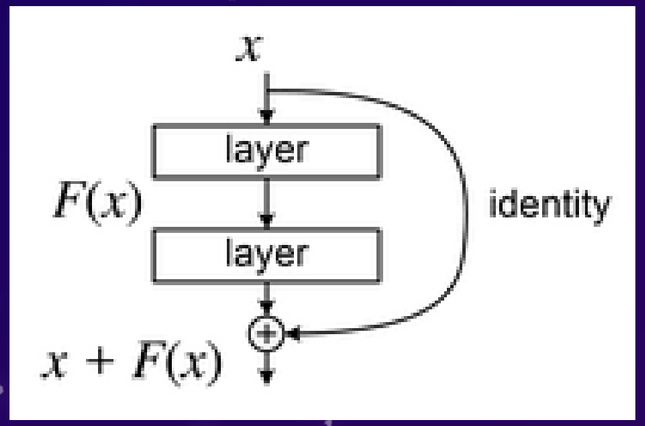
✓ Optimized
82.0% |Loss: 0.54
50 epochs

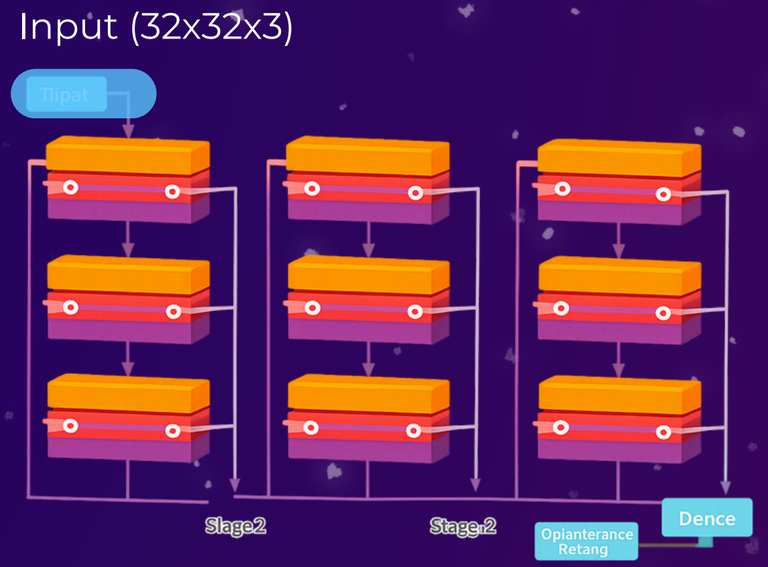
ResNet-20: Solving the Depth Problem

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: Evaluation Dashboard

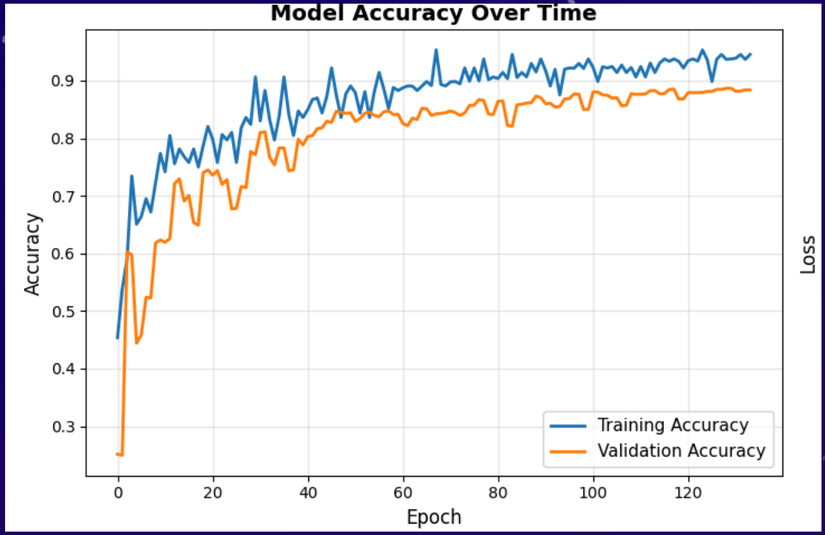
Our tweaks:

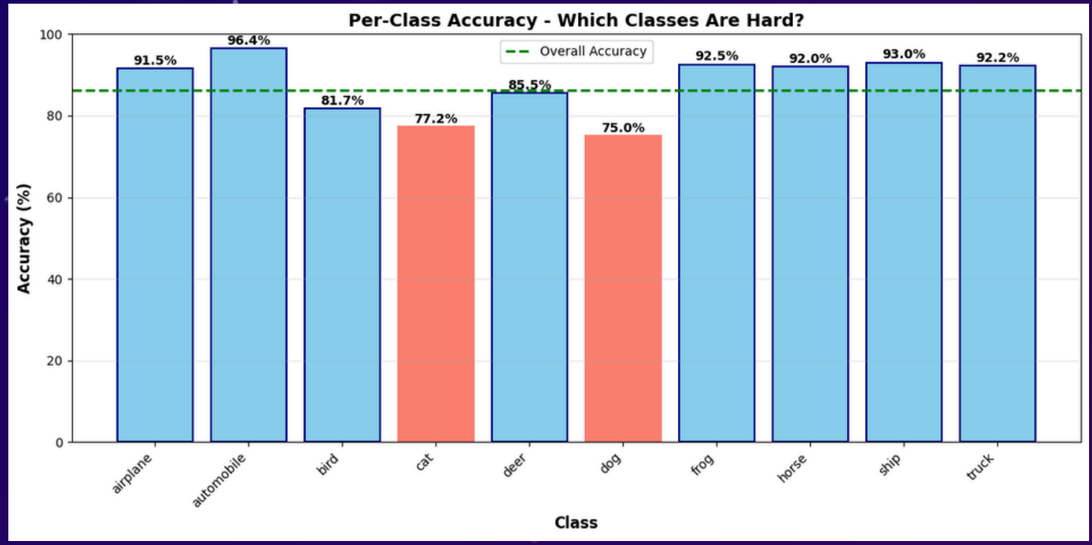
- ✓ Callbacks (Early stop, ReduceLRonPlateau)
- ✓ Data augmentation
- ✓ Dropout

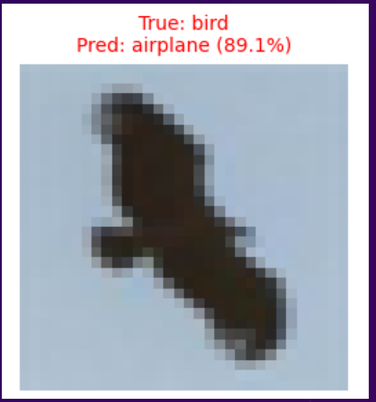
Accuracy: **87.7%**

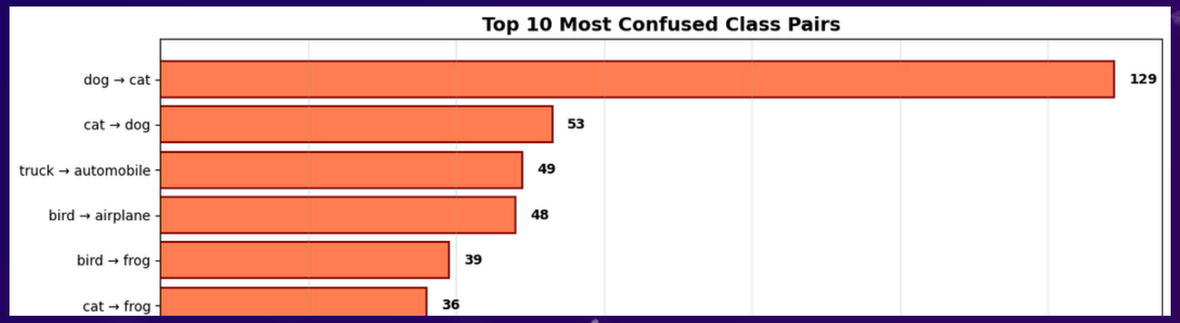
Loss: 0.39

Overfitting gap: 3.1%



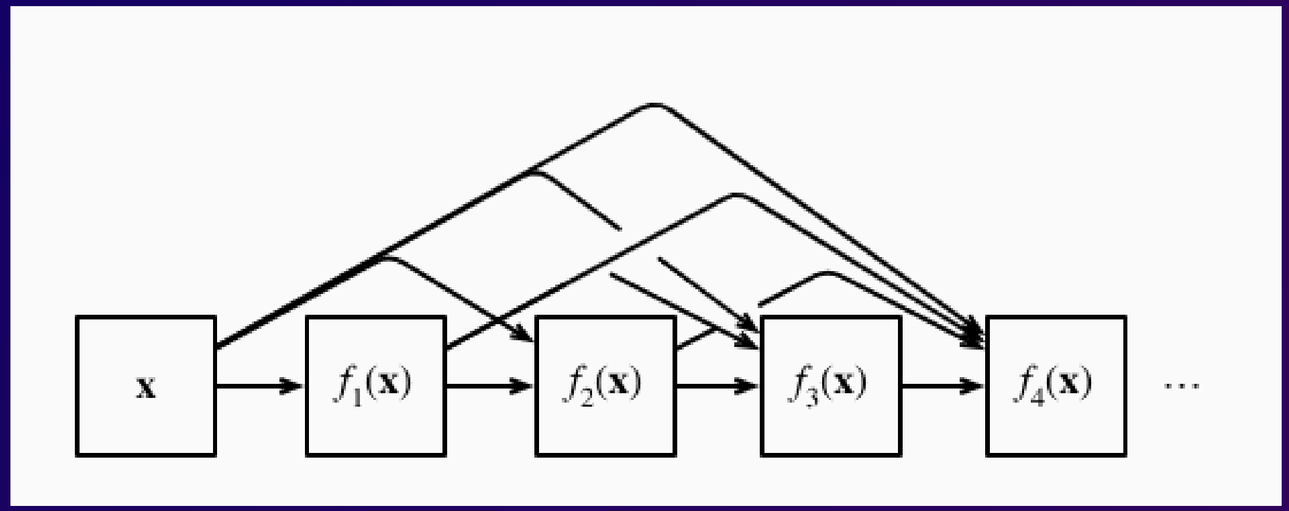






Transfer Learning

DenseNet121 - Architecture



- **Architecture :**
  - Each layer receives the feature maps of all previous layers as input.

- **DenseNet is known for:**
  - Strong feature reuse through dense connections
  - Good performance with very deep networks

DenseNet121 - Freezed

```python
from tensorflow.keras.applications import DenseNet121

base_model = DenseNet121(
    weights="imagenet",
    include_top=False,
    input_shape=(96, 96, 3)
)

base_model.trainable = False
```

```python
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.BatchNormalization(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

- Started with an input shape (96 ,96 , 3)
- Freezed all layers of DenseNet
- Added a custom classifier with
  - GlobalaveragePooling
  - Batch normalization
  - Dropout (0,5)
  - Dense layer (256)
  - ReLu

- **Training Acc**: 90 %
- **Validation Acc:** 88 %
- **Test Acc** : 89%

DenseNet121 - Evaluation

	precision	recall	f1-score	support
airplane	0.89	0.90	0.90	1000
automobile	0.91	0.92	0.92	1000
bird	0.91	0.83	0.87	1000
cat	0.77	0.85	0.80	1000
deer	0.85	0.86	0.86	1000
dog	0.88	0.82	0.85	1000
frog	0.90	0.92	0.91	1000
horse	0.91	0.91	0.91	1000
ship	0.93	0.93	0.93	1000
truck	0.92	0.91	0.91	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
eighted avg	0.89	0.89	0.89	10000

- Cat : Precision is lower → many non-cat instances are misclassified as cats.
- Best classes: ship, automobile, horse, frog very high precision and recall.

DenseNet121 - unfreezed

```python
# Make the base model trainable
base_model.trainable = True

# Freeze all layers except the last N (e.g., last 20)
fine_tune_at = len(base_model.layers) - 20
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

- unfreezed the last 20 layers of DenseNet.
- Fine Tune with a small LR

```python
# Compile with smaller learning rate
from tensorflow.keras import optimizers

model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Fine-tune
history_fine = model.fit(
    train_ds,
    validation_data=test_ds,
    epochs=5
)
```

- **Training Acc**: 92 %
- **Validation Acc:** 89 %
- **Test Acc** : 91%

Restnet50

Input (resized to 224 × 224 × 3

base_inception

GlobalAveragePooling2D()

ReLu

Dense(256, activation='relu')

Model	Accuracy (%)	F1-score
CNN Baseline	69,66%	70%
CNN VGG-like	81%	81%
Restnet	92%	92%

Overall Performance

- Accuracy: 92%
- Dataset is balanced (1000 images per class)
- Strong and consistent performance across all categories

The model correctly classifies 9 out of every 10 images.

Best Performing Classes

- Automobile: F1-score 0.95
- Ship: F1-score 0.95
- Airplane, Frog, Horse, Truck: F1-score ≈ 0.94

✓ Clear shapes and distinctive visual patterns

✓ Features well captured by pre-trained

ResNet50 filters

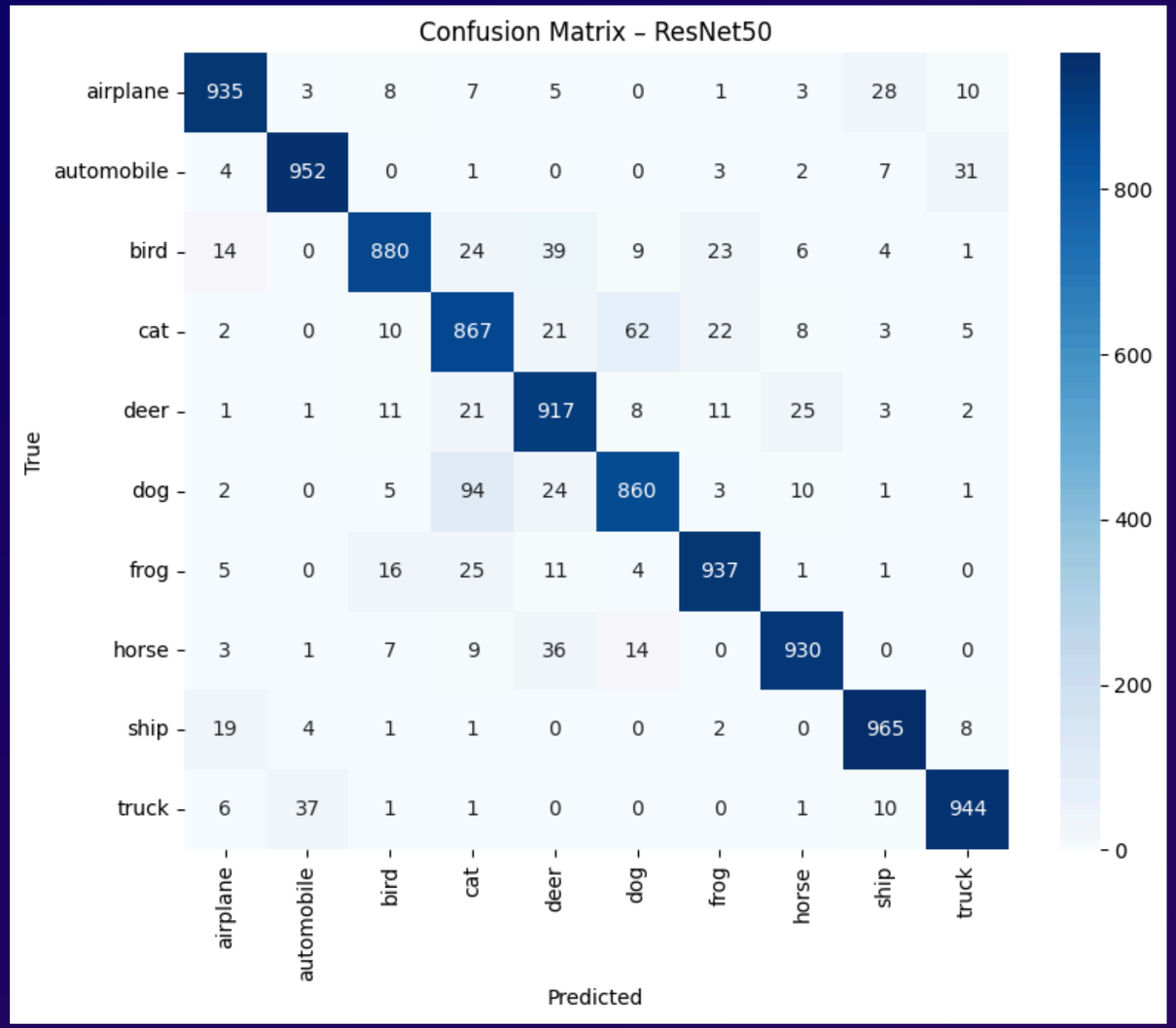
More Challenging Classes

- Cat: F1-score 0.85
- Dog: F1-score 0.88
- Bird: F1-score 0.91

◆ Visual similarity between animals

◆ High variability in poses and backgrounds

ResNet50



Cat vs Dog Confusion

- Cats and dogs share:
  - Similar textures
  - Similar shapes
  - Similar environments

AI

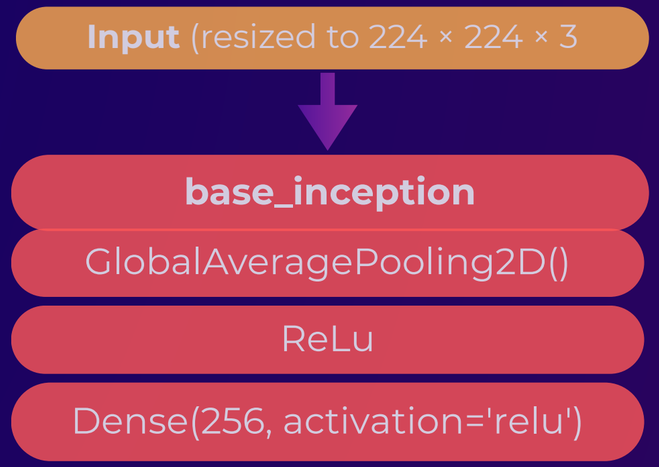
Even with a deep model, these classes remain challenging.

Why ResNet50 Performs So Well

- Transfer Learning from ImageNet
- Residual connections improve gradient flow
- Deep semantic feature extraction
- Built-in batch normalization
- Data augmentation reduces overfitting

The classification report shows that ResNet50 with transfer learning achieves strong and balanced performance across all classes, reaching 92% overall accuracy and high F1-scores, with only minor confusion between visually similar categories such as cats and dogs.

Inceptionv3



Model	Accuracy (%)	F1-score
CNN Baseline	69,66%	70%
CNN VGG-like	81%	81%
Restnet	92%	92%
basic Inception	89%	89
inception fine tuning	90%	90

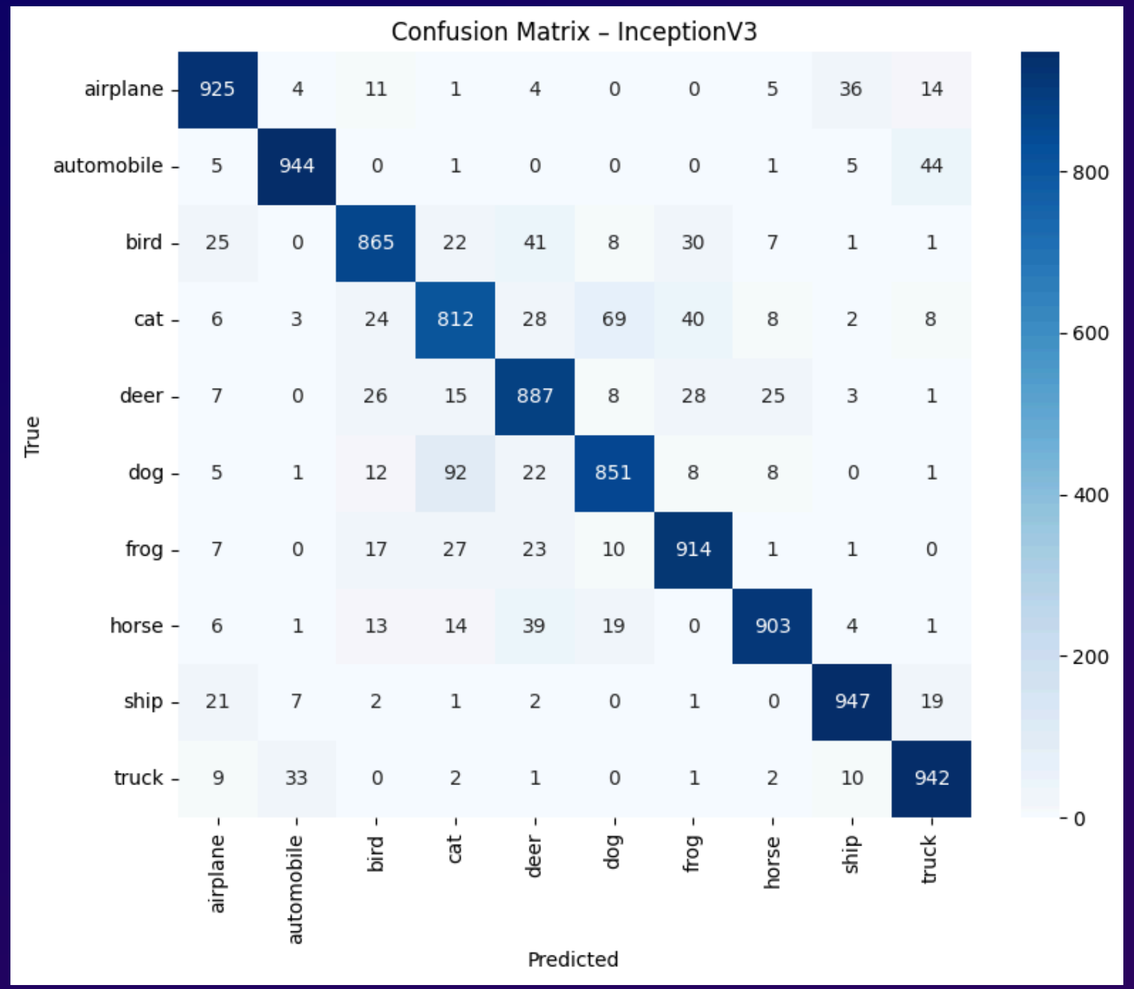
Overall Performance

- Accuracy: 90%
- Dataset is balanced (1000 images per class)
- Strong performance, slightly lower than ResNet50

The model correctly classifies 9 out of every 10 images.

s

Inceptionv3



Best Performing Classes

- Automobile: F1-score 0.95
  - Ship: F1-score 0.94
  - Truck: F1-score 0.93
  - Horse: F1-score 0.92
- ✓ Large, well-defined shapes
- ✓ Multi-scale features effectively captured by Inception modules

More Challenging Classes

- Cat: F1-score 0.82
- Dog: F1-score 0.87
- Bird: F1-score 0.88

◆ High intra-class variability

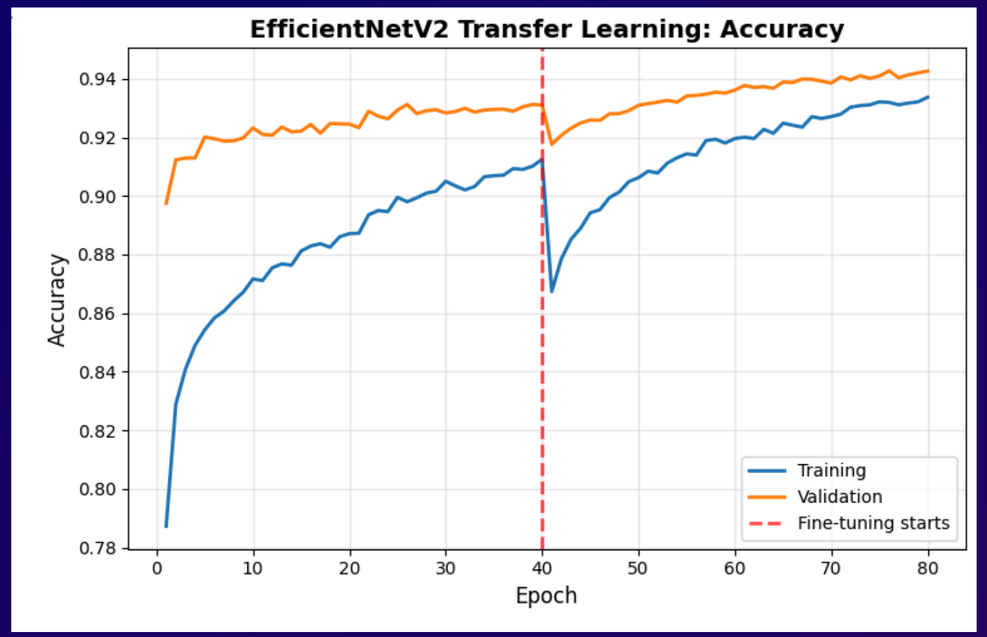
◆ Visual similarity between animals

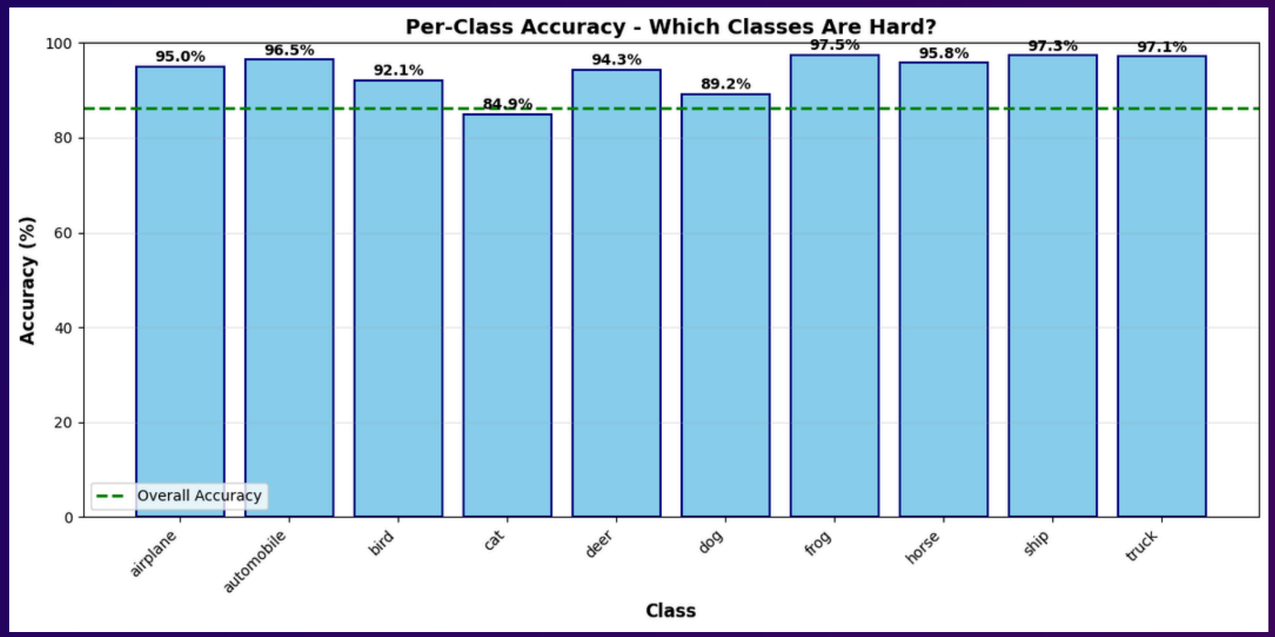
Cat vs Dog Confusion

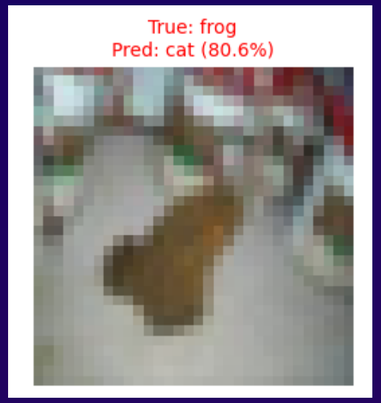
- Cats and dogs share:
- Similar textures
- Similar body structures
- Similar backgrounds
  -

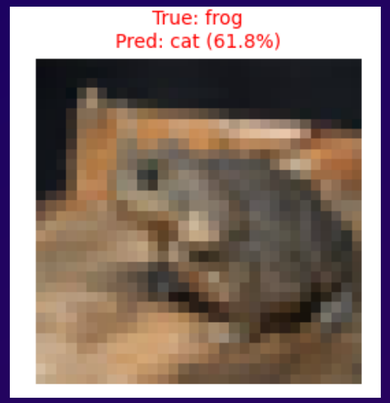
EfficientNetV2

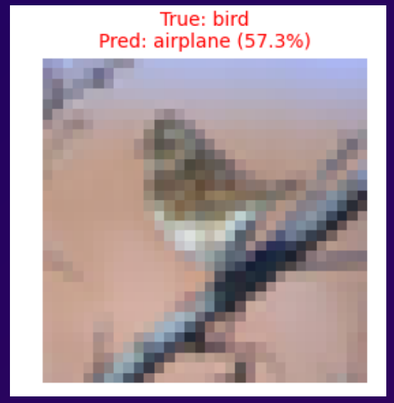
Test Accuracy: **93.97%**
Test Loss: 0.17
Overfitting Gap: 0

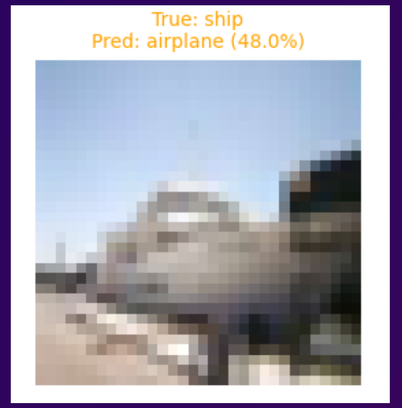












Conclusion

- ResNet-20 achieved a test accuracy of 87.7%.
- ResNet-50 improved performance, reaching 92% test accuracy.
- A simple CNN trained from scratch, without tricks or pretrained weights, achieved performance comparable to pretrained models. 90%
- In all experiments both pretrained and non-pretrained the dog and cat classes showed the weakest performance.
- This indicates that the main limitation is the dataset rather than the model architecture.
- The high visual similarity between dogs and cats plus low resolution likely makes these classes difficult to distinguish.
- These results confirm that modern deep architectures, when combined with transfer learning, outperform traditional convolutional networks trained from scratch in both accuracy and generalization

THANK YOU