**MASTER OF SCIENCE OF INFORMATION SYSTEMS**
INTELLIGENT SYSTEMS

# DEEP LEARNING

## Image Recognition with Transfer Learning (CIFAR-10 Dataset)


| | | |
|---|---|---|
| STUDENTS (GROUP2) | AINEDEMBE DENIS | 2024-M132-23999 |
| | MUSINGUZI BENSON | 2024-M132-23947 |
| LECTURER | Dr. Sibitenda Harriet | |

# Project Objectives

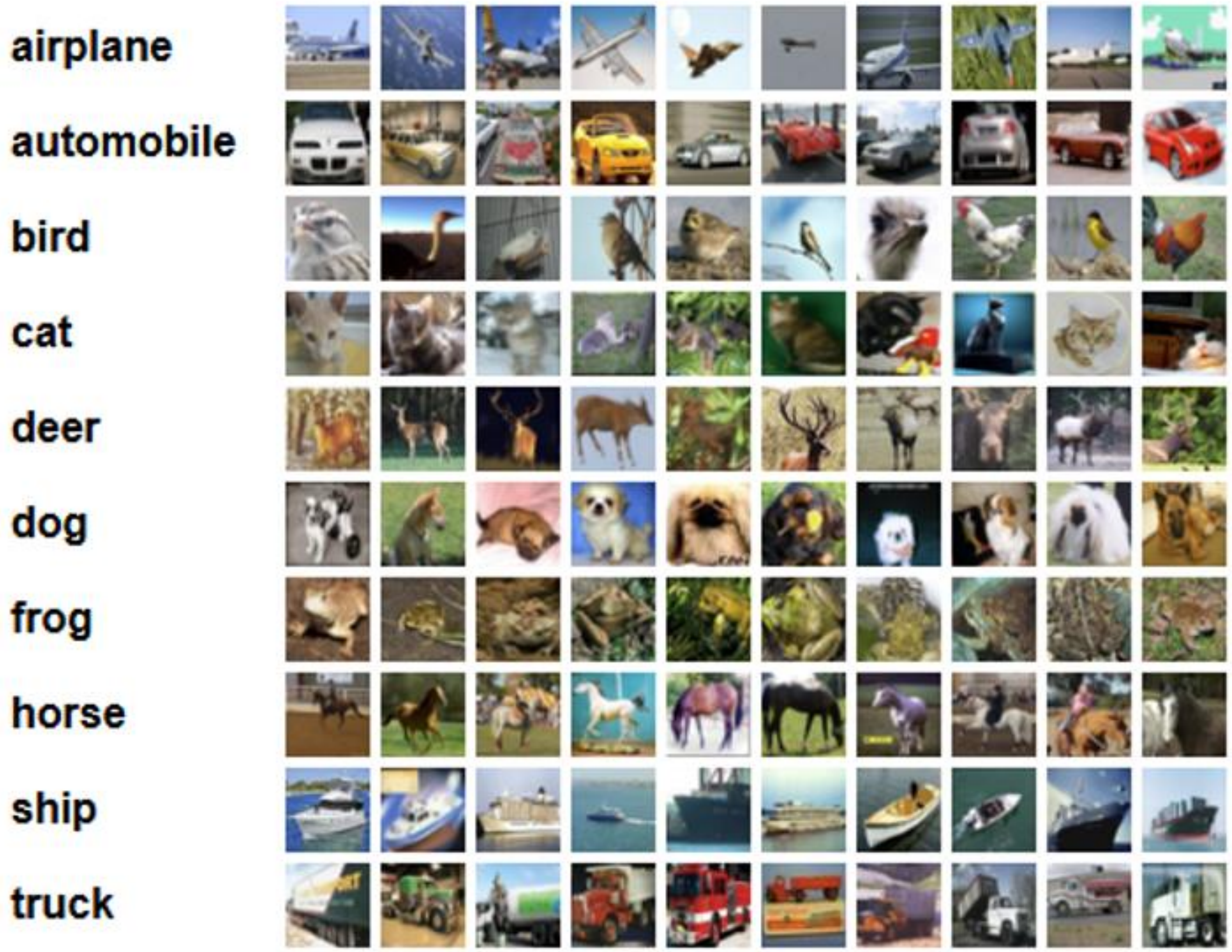
- Apply transfer learning to the CIFAR-10 image classification task
- Compare performance of pre-trained models and a custom CNN
- Implement preprocessing, upsampling, and augmentation techniques
- Analyze training outputs and model behavior

# About the CIFAR-10 Dataset

- CIFAR-10 contains 60,000 color images sized 32×32 pixels.
- 10 mutually exclusive classes, with 6,000 images per class.
- Split: 50,000 training images and 10,000 test images.
- Dataset structure: 5 training batches (10,000 each) and 1 test batch.
- Test batch has exactly 1,000 images per class.
- Training batches contain 5,000 images per class in total (random distribution).
- Each batch is a pickled Python dictionary containing 'data' and 'labels'

# About the CIFAR-10 Dataset

Here are the classes in the dataset, as well as 10 random images from each:



# Real-Life Applicability

1. **Automated Image Classification**

- Used in photo apps, digital libraries, and content filtering
- Helps systems recognize and sort images automatically

**2. Object Detection Pre-Training**

- CIFAR-10 teaches models to detect basic objects
- Useful for self-driving cars, surveillance, and drone vision

**3. Transfer Learning for Real Datasets**

- Same method used in medical imaging (X-ray/MRI)
- Also applied in face recognition and other advanced AI systems

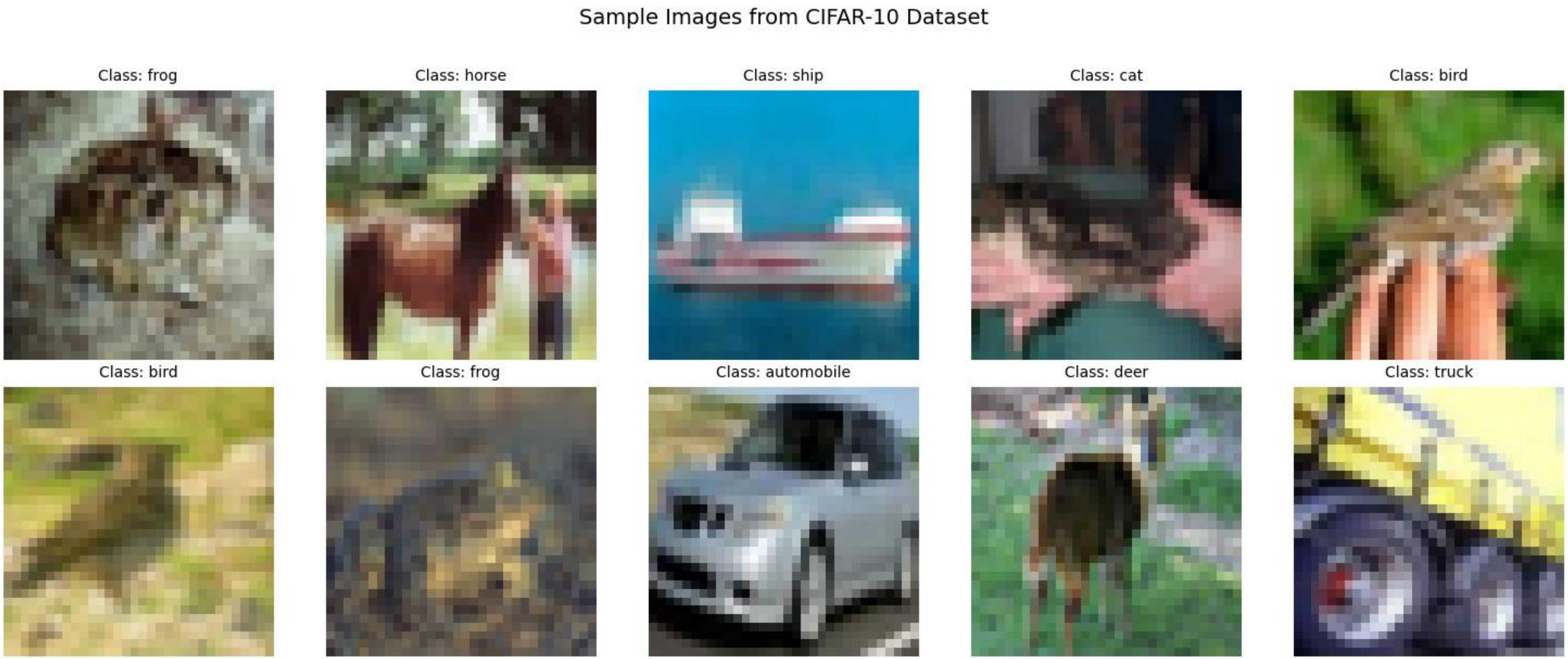
# Load CIFAR-10 datase

```
Train: (50000, 32, 32, 3) (50000,)
Test : (10000, 32, 32, 3) (10000,)

After split:
Training:   (40000, 32, 32, 3), (40000,) (40000 samples)
Validation: (10000, 32, 32, 3), (10000,) (10000 samples)
Test:       (10000, 32, 32, 3), (10000,) (10000 samples)

Visualizing 10 sample images from the dataset...
```

# Visualize sample images from each class



# Pre-trained model Used
# **MobileNetV2**

# Data augmentation - Applying rotation, flipping, and shifts

**BATCH_SIZE = 128:**

- Number of samples processed together in one training step
- With 40,000 training images and batch size 128, you get ~313 batches per epoch (40,000 ÷ 128)

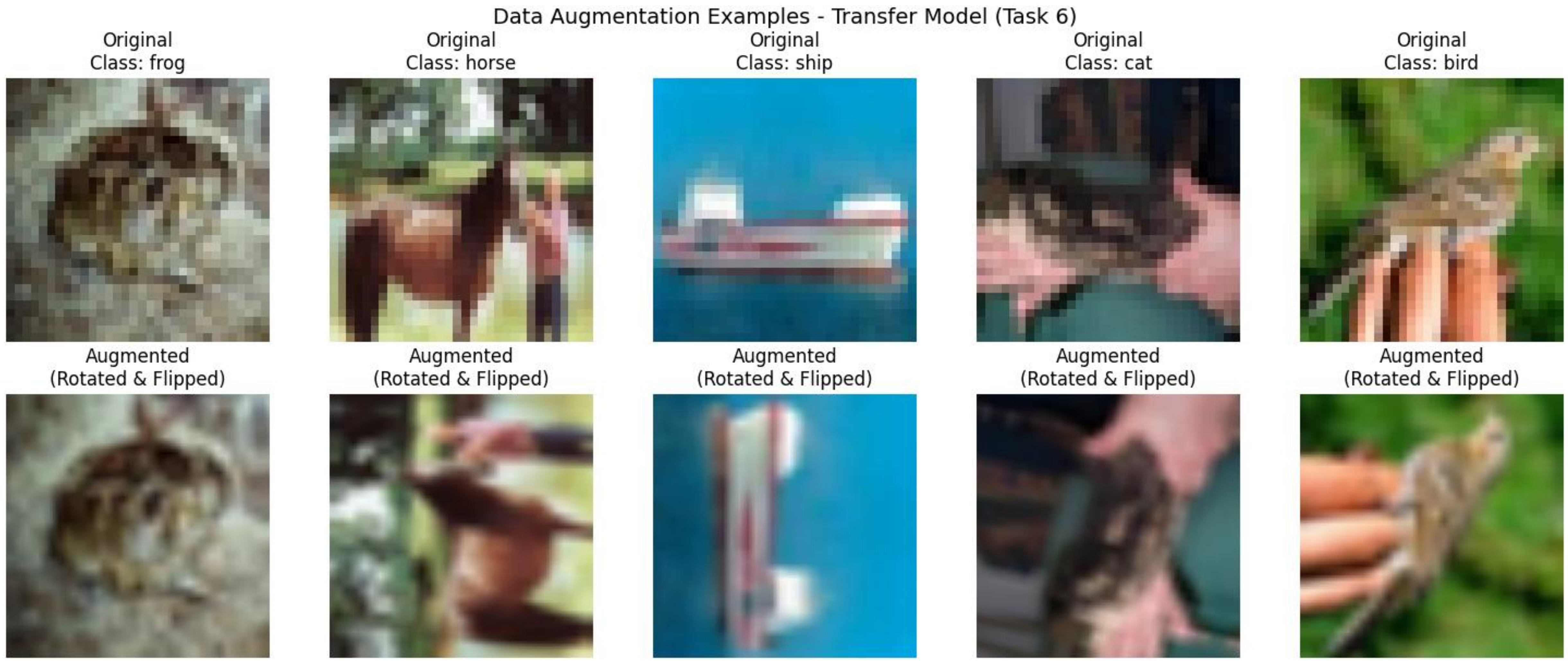
**Target image dimensions for MobileNetV2 input =96**

- CIFAR-10 images are 32x32, but MobileNetV2 expects larger images
- 96x96 is a good size: larger than 32x32 but smaller than standard 224x224

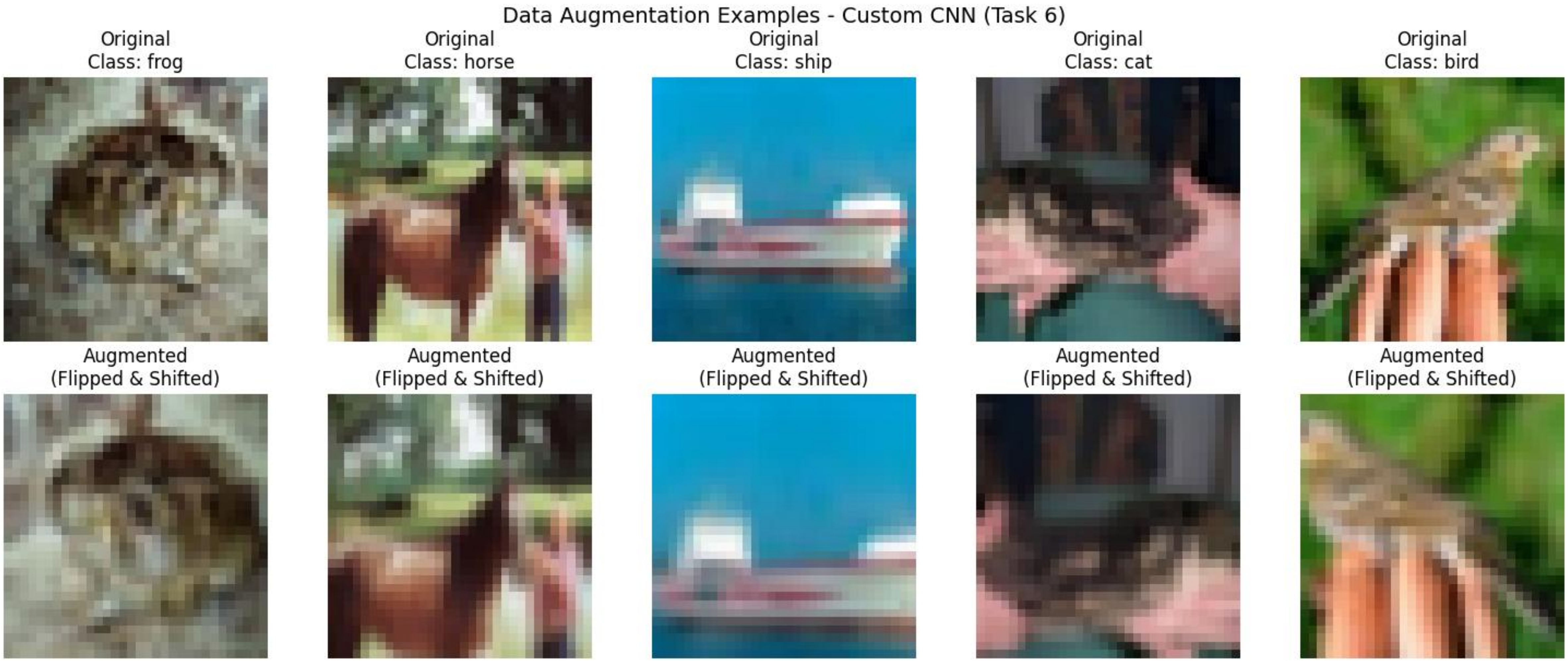
**Data augmentation - Apply rotation, flipping, and shifts**

- Random rotation (up to 15 degrees ≈ 0.262 radians)
- Random horizontal flipping /vertical shifts

# Apply data augmentation (rotation, flipping, shifts)



# Apply data augmentation (rotation, flipping, shifts)



# Using pre-trained MobileNetV2 for transfer learning

Model: "mobilenetv2_transfer"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_layer_6 (InputLayer) | (None, 96, 96, 3) | 0 |
| mobilenetv2_1.00_96 (Functional) | (None, 3, 3, 1280) | 2,257,984 |
| global_average_pooling2d_1 (GlobalAveragePooling2D) | (None, 1280) | 0 |
| dropout_1 (Dropout) | (None, 1280) | 0 |
| dense_1 (Dense) | (None, 10) | 12,810 |

Total params: 2,270,794 (8.66 MB)

Trainable params: 12,810 (50.04 KB)

Non-trainable params: 2,257,984 (8.61 MB)

# Freeze lower layers, retrain top layers for 10 epochs

```
Starting Training: 10 epochs, 40,000 training samples per epoch (313 batches)
______________________________________________________________________

Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 155s 472ms/step - accuracy: 0.1036 - loss: 2.4352 - val_accuracy: 0.1065 - val_loss: 2.3167
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 147s 470ms/step - accuracy: 0.1023 - loss: 2.3439 - val_accuracy: 0.1000 - val_loss: 2.3037
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 149s 475ms/step - accuracy: 0.1054 - loss: 2.3255 - val_accuracy: 0.1000 - val_loss: 2.3107
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 150s 480ms/step - accuracy: 0.1092 - loss: 2.3205 - val_accuracy: 0.1330 - val_loss: 2.3015
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 148s 473ms/step - accuracy: 0.1122 - loss: 2.3164 - val_accuracy: 0.1365 - val_loss: 2.3073
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 147s 469ms/step - accuracy: 0.1148 - loss: 2.3121 - val_accuracy: 0.1182 - val_loss: 2.2933
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 150s 478ms/step - accuracy: 0.1153 - loss: 2.3071 - val_accuracy: 0.1318 - val_loss: 2.2874
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 148s 473ms/step - accuracy: 0.1169 - loss: 2.3036 - val_accuracy: 0.1572 - val_loss: 2.2979
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 149s 475ms/step - accuracy: 0.1184 - loss: 2.3032 - val_accuracy: 0.1442 - val_loss: 2.2973
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 147s 470ms/step - accuracy: 0.1193 - loss: 2.3058 - val_accuracy: 0.1015 - val_loss: 2.3244
```

# Freeze lower layers, retrain top layers for 10 epochs

```
DATASET TRAINING SUMMARY
________________________________________________________________________
Training Dataset: 40,000 images (after 80/20 split from 50,000)
Batch Size: 128
Batches per Epoch: 313 (40,000 ÷ 128 = 313 batches, including partial batch)
------------------------------------------------------------------------
Epoch 1: 40,000 training samples (313 batches)
Epoch 2: 40,000 training samples (313 batches)
Epoch 3: 40,000 training samples (313 batches)
Epoch 4: 40,000 training samples (313 batches)
Epoch 5: 40,000 training samples (313 batches)
Epoch 6: 40,000 training samples (313 batches)
Epoch 7: 40,000 training samples (313 batches)
Epoch 8: 40,000 training samples (313 batches)
Epoch 9: 40,000 training samples (313 batches)
Epoch 10: 40,000 training samples (313 batches)
------------------------------------------------------------------------
Total training samples across all 10 epochs: 400,000
Total batches processed: 3,130 (313 batches × 10 epochs)
Average samples per epoch: 40,000
```

```
____________________________________________
PERFORMANCE METRICS
____________________________________________
Training Accuracy:
  - Epoch 1: 0.1036 (10.36%)
  - Epoch 10: 0.1193 (11.93%)
  - Improvement: 0.0157 (1.57% increase)

Training Loss:
  - Epoch 1: 2.4352
  - Epoch 10: 2.3058
  - Decrease: 0.1294 (5.31% reduction)

Validation Accuracy:
  - Range: 0.1000 - 0.1572 (10.00% - 15.72%)
  - Average: 0.1229 (12.29%)

Validation Loss:
  - Range: 2.2874 - 2.3244
  - Average: 2.3040
```

# Fine-tuning different layers and learning rates

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 256s 772ms/step - accuracy: 0.2480 - loss: 2.3077 - val_accuracy: 0.1000 - val_loss: 2.9783
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 234s 747ms/step - accuracy: 0.3178 - loss: 1.9005 - val_accuracy: 0.1000 - val_loss: 2.6724
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 238s 760ms/step - accuracy: 0.3462 - loss: 1.8200 - val_accuracy: 0.1000 - val_loss: 2.6204
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 251s 801ms/step - accuracy: 0.3688 - loss: 1.7686 - val_accuracy: 0.1000 - val_loss: 3.0509
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 235s 752ms/step - accuracy: 0.3806 - loss: 1.7354 - val_accuracy: 0.1000 - val_loss: 2.6806
```

Fine-tune from this layer onwards
fine_tune_at = 100

EPOCHS_FINE_TUNE = 5

# Experimenting with fine-tuning different layers & learning rates

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 259s 788ms/step - accuracy: 0.2498 - loss: 2.2933 - val_accuracy: 0.1000 - val_loss: 2.4917
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 241s 768ms/step - accuracy: 0.3203 - loss: 1.8911 - val_accuracy: 0.1000 - val_loss: 3.0779
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 240s 765ms/step - accuracy: 0.3478 - loss: 1.8236 - val_accuracy: 0.1000 - val_loss: 2.5314
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 239s 763ms/step - accuracy: 0.3671 - loss: 1.7743 - val_accuracy: 0.1000 - val_loss: 2.6182
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 239s 762ms/step - accuracy: 0.3808 - loss: 1.7345 - val_accuracy: 0.1000 - val_loss: 2.7991
```

Fine-tune from this layer onwards
Fine Tune at = 100

EPOCHS FINE TUNE = 5

# Building custom CNN from scratch for performance comparison

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 32, 32, 32) | 896 |
| batch_normalization (BatchNormalization) | (None, 32, 32, 32) | 128 |
| max_pooling2d (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 16, 16, 64) | 18,496 |
| batch_normalization_1 (BatchNormalization) | (None, 16, 16, 64) | 256 |
| max_pooling2d_1 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 8, 8, 128) | 73,856 |
| batch_normalization_2 (BatchNormalization) | (None, 8, 8, 128) | 512 |
| max_pooling2d_2 (MaxPooling2D) | (None, 4, 4, 128) | 0 |
| flatten (Flatten) | (None, 2048) | 0 |
| dense_2 (Dense) | (None, 256) | 524,544 |
| batch_normalization_3 (BatchNormalization) | (None, 256) | 1,024 |
| dropout_2 (Dropout) | (None, 256) | 0 |
| dense_3 (Dense) | (None, 10) | 2,570 |

Total params: 622,282 (2.37 MB)

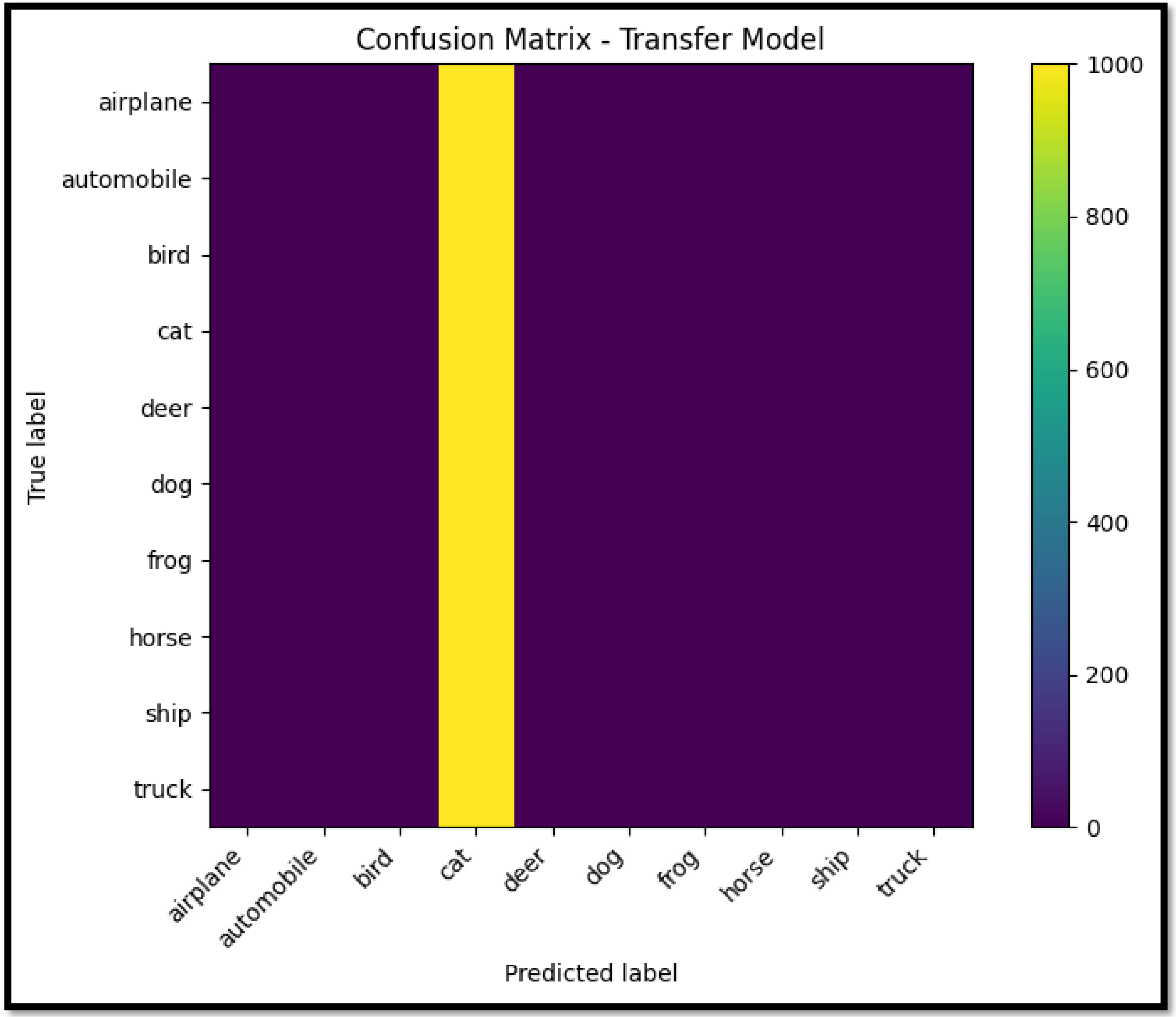
Trainable params: 621,322 (2.37 MB)

Non-trainable params: 960 (3.75 KB)

# Building custom CNN from scratch for performance comparison

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 48s 139ms/step - accuracy: 0.4297 - loss: 1.7051 - val_accuracy: 0.2515 - val_loss: 2.6389
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 42s 133ms/step - accuracy: 0.5721 - loss: 1.2083 - val_accuracy: 0.5144 - val_loss: 1.5093
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 137ms/step - accuracy: 0.6282 - loss: 1.0555 - val_accuracy: 0.5362 - val_loss: 1.4560
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 137ms/step - accuracy: 0.6607 - loss: 0.9633 - val_accuracy: 0.6068 - val_loss: 1.2049
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 44s 141ms/step - accuracy: 0.6873 - loss: 0.8996 - val_accuracy: 0.6197 - val_loss: 1.0911
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 138ms/step - accuracy: 0.7045 - loss: 0.8474 - val_accuracy: 0.6519 - val_loss: 1.0377
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 138ms/step - accuracy: 0.7176 - loss: 0.8131 - val_accuracy: 0.6378 - val_loss: 1.1284
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 47s 150ms/step - accuracy: 0.7256 - loss: 0.7879 - val_accuracy: 0.6902 - val_loss: 0.9070
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 49s 156ms/step - accuracy: 0.7350 - loss: 0.7579 - val_accuracy: 0.6233 - val_loss: 1.1508
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 47s 148ms/step - accuracy: 0.7465 - loss: 0.7297 - val_accuracy: 0.6723 - val_loss: 0.9750
```

# Evaluating accuracy and confusion matrix; visualize predictions




Transfer (fine-tuned) test accuracy: 0.10000000149011612

Custom CNN test accuracy: 0.6708999872207642

79/79 ━━━━━━━━ 33s 388ms/step
Confusion matrix shape: (10, 10)


```
Transfer (fine-tuned) test accuracy: 0.10000000149011612
Custom CNN test accuracy: 0.6708999872207642
79/79 ━━━━━━━━━━━━━━━━━━━━ 33s 388ms/step
Confusion matrix shape: (10, 10)
```

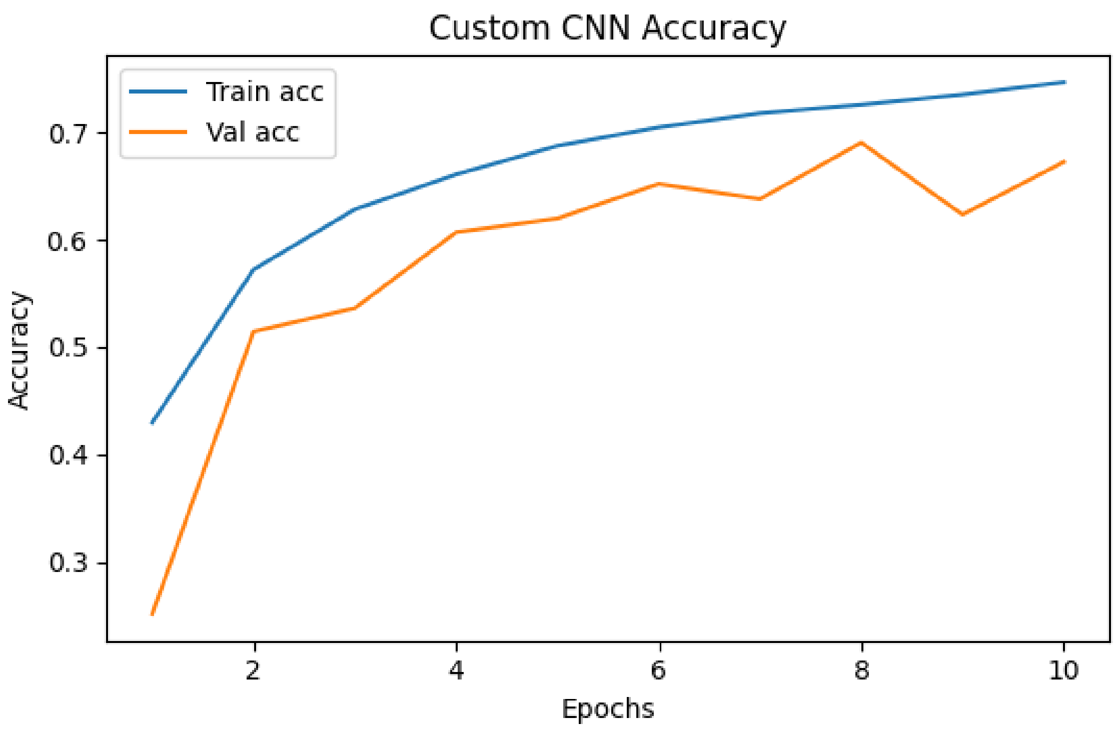
# Compare Performance with custom CNN (Continued)

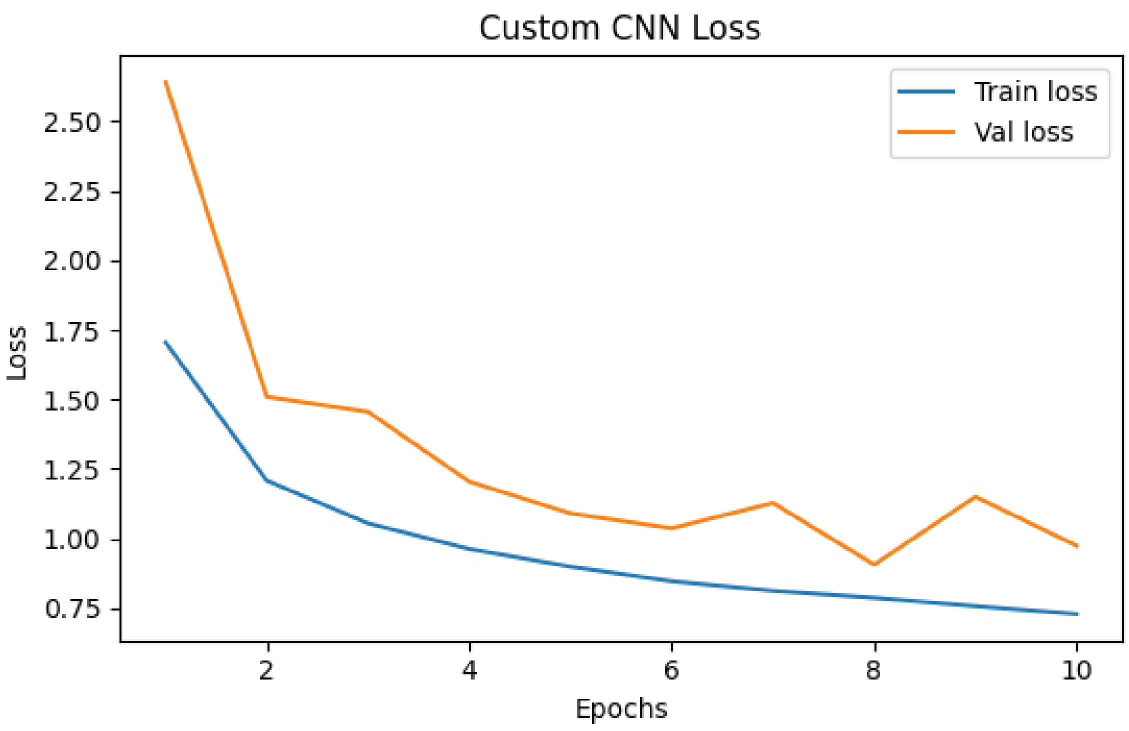
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.00 | 0.00 | 0.00 | 1000 |
| automobile | 0.00 | 0.00 | 0.00 | 1000 |
| bird | 0.00 | 0.00 | 0.00 | 1000 |
| cat | 0.10 | 1.00 | 0.18 | 1000 |
| deer | 0.00 | 0.00 | 0.00 | 1000 |
| dog | 0.00 | 0.00 | 0.00 | 1000 |
| frog | 0.00 | 0.00 | 0.00 | 1000 |
| horse | 0.00 | 0.00 | 0.00 | 1000 |
| ship | 0.00 | 0.00 | 0.00 | 1000 |
| truck | 0.00 | 0.00 | 0.00 | 1000 |
| accuracy | | | 0.10 | 10000 |
| macro avg | 0.01 | 0.10 | 0.02 | 10000 |
| weighted avg | 0.01 | 0.10 | 0.02 | 10000 |

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 414ms/step
```

**Sample predictions**

# Plot training history (accuracy/loss) for all models




**Accuracy:**

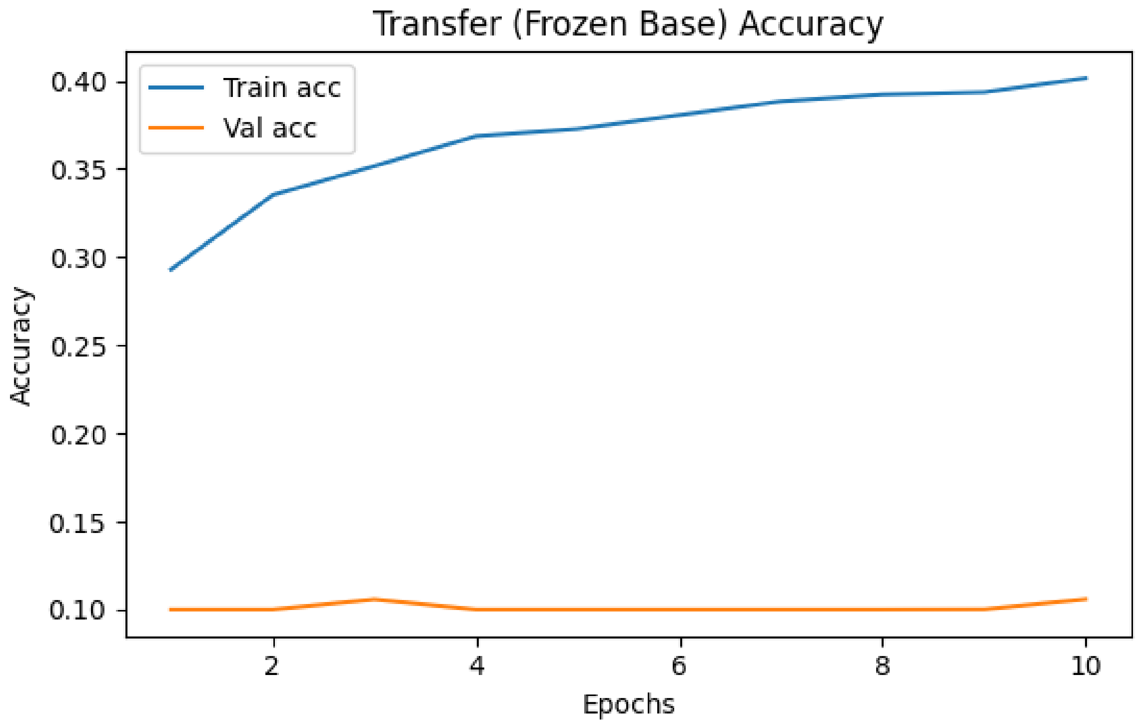
- Training: Starts at ~40%, increases to ~75% by epoch 10
- Validation: Starts at ~25%, increases to ~67% by epoch 10
- Gap: ~8% difference (training higher)

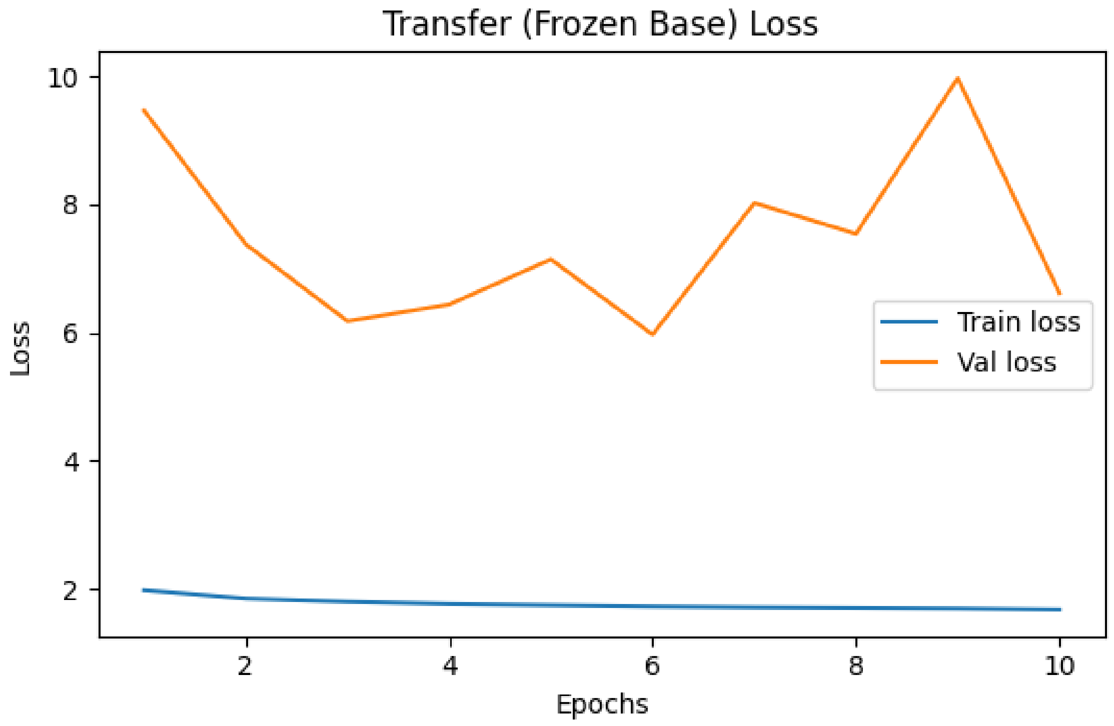
**Loss:**

- Training: Starts at ~1.7, decreases to <1.0 by epoch 10
- Validation: Starts at ~2.5, decreases to ~1.05 by epoch 10
- Gap: Validation loss remains higher

**Overall: Best performance.** Both metrics improve, with moderate overfitting.

# Plot training history (accuracy/loss) for all models




**Accuracy:**

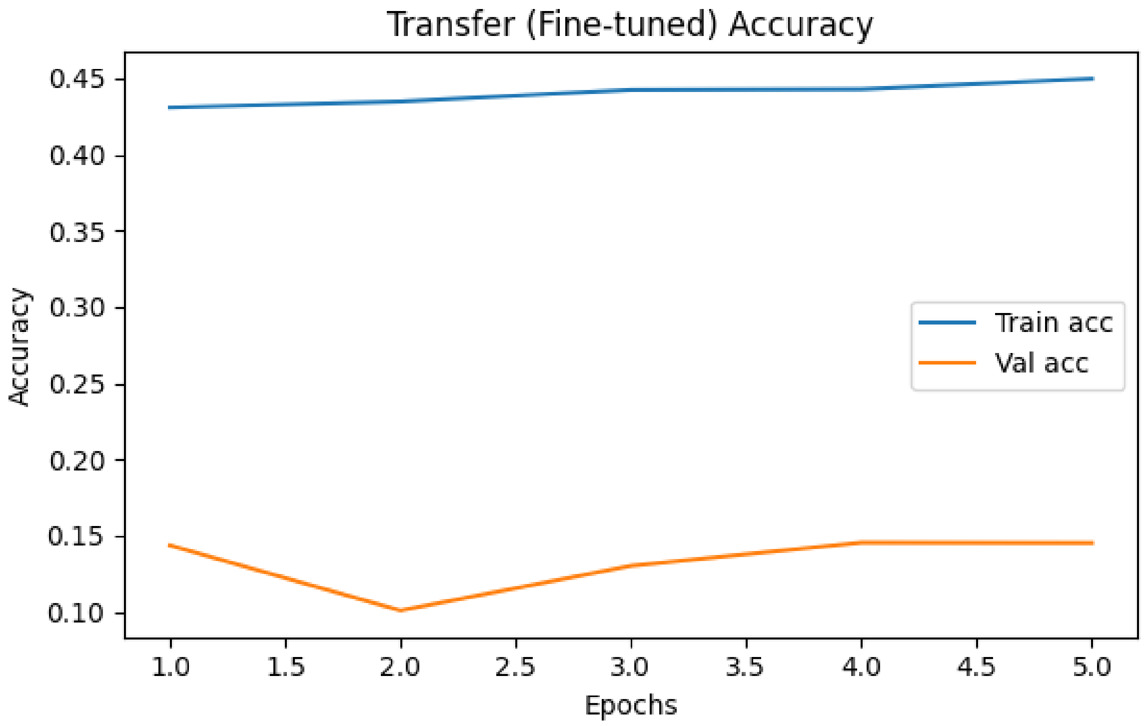
- Training: Starts at ~29%, increases to ~40% by epoch 10
- Validation: Stays very low (~10–12%), near random chance
- Gap: Large and growing (training ~40% vs validation ~10%)

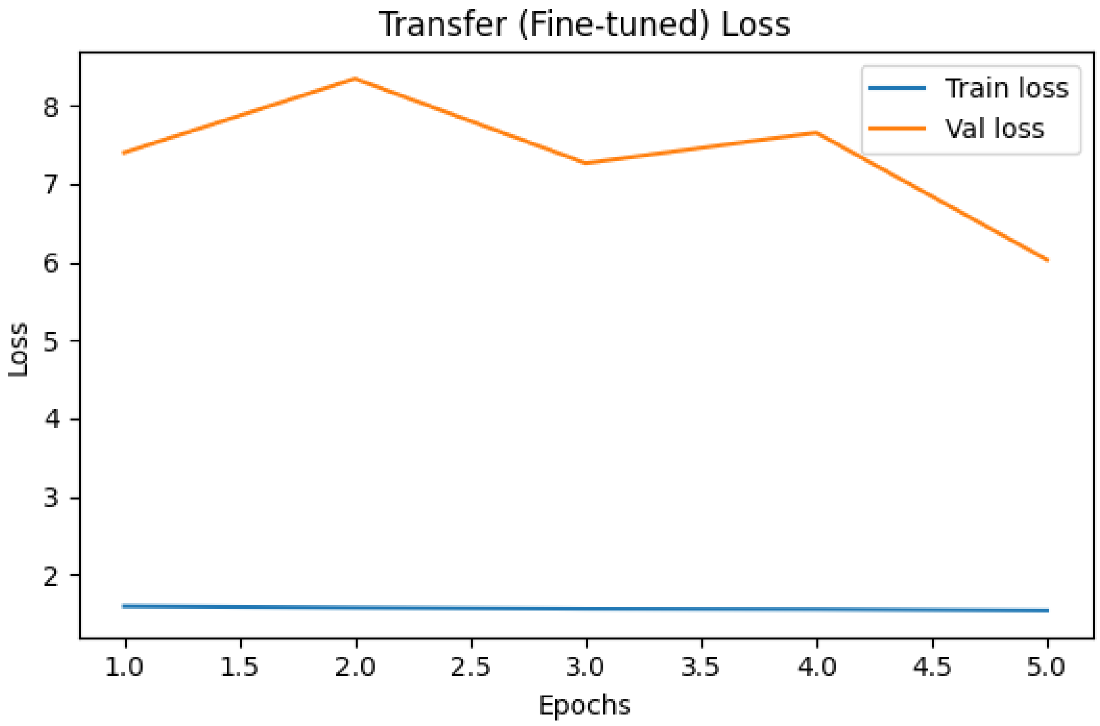
**Loss:**

- Training: Starts at ~2.0, stabilizes around 1.8–1.9
- Validation: Starts at ~9.5, fluctuates between 6–10 with spikes
- Gap: Very large (validation loss 3–5x higher)

**Overall: Severe overfitting.** Training improves but validation does not, indicating poor generalization.

# Plot training history (accuracy/loss) for all models




**Accuracy:**

- Training: Starts at ~43%, stabilizes around ~45% by epoch 5
- Validation: Starts at ~15%, drops to ~10% at epoch 2, recovers to ~15% by epoch 5
- Gap: Very large (~30% difference)

**Loss:**

- Training: Starts at ~1.8, stays stable around 1.7–1.8
- Validation: Starts at ~7.5, spikes to >8.0 at epoch 2, decreases to ~6.0 by epoch 5
- Gap: Very large (validation loss 3–4x higher)

**Overall: Severe overfitting.** Fine-tuning did not improve generalization; validation metrics are poor.

# Summary & Discussion

```
1. TEST SET ACCURACY (Generalization Performance)
________________________________________________________________________________

  • Custom CNN:                        0.6709 (67.09%)
  • Transfer Learning (Fine-tuned):    0.1000 (10.00%)
  • Performance Difference:            57.09% points

  ANALYSIS:
     - Custom CNN substantially outperforms (by 57.1% points) transfer learning
     - Custom CNN achieves ~67% accuracy, indicating good generalization
     - Transfer model achieves ~10% accuracy, indicating near random chance (failing to generalize)
```

**Part 1.1:**

- Test Set Accuracy

```
2. VALIDATION ACCURACY (Final Epoch - Model Selection Metric)
________________________________________________________________________________

  • Custom CNN:                        0.6723 (67.23%)
  • Transfer (Frozen Base):            0.1059 (10.59%)
  • Transfer (Fine-tuned):             0.1452 (14.52%)

  ANALYSIS:
     - Custom CNN: ~67% validation accuracy (best performer)
     - Transfer (Frozen): ~11% - near random chance (~10-12% for 10 classes)
     - Transfer (Fine-tuned): ~15% - very poor performance (close to random)
     - Fine-tuning: slight improvement (~4% points change)
```

**Part 1.2:**

- Validation Accuracy

# Summary & Discussion

```
3. OVERFITTING ANALYSIS (Train vs Validation Gap)
──────────────────────────────────────────────────────────────────────

   ANALYSIS:

   • Custom CNN:
     - Training: ~75% | Validation: ~67%
     - Gap: ~7% points (MODERATE OVERFITTING)
     - Status: acceptable - model generalizes reasonably well

   • Transfer (Frozen Base):
     - Training: ~40% | Validation: ~11%
     - Gap: ~30% points (SEVERE OVERFITTING)
     - Status: problematic - model memorizes training data significantly

   • Transfer (Fine-tuned):
     - Training: ~45% | Validation: ~15%
     - Gap: ~30% points (EXTREME OVERFITTING)
     - Status: critical issue - model memorizes training data but fails to generalize

   SUMMARY:
      - Custom CNN shows the least overfitting (~7% gap) - best generalization
      - Transfer models show severe/extreme overfitting (worst: ~30% gap)
      - Transfer models memorize training data but fail to generalize to unseen data
```

**Part 1.3:**

- Overfitting Analysis

# Summary & Discussion

```
4. TRAINING CONFIGURATION

────────────────────────────────────────────────────────────────────────────────

   • Custom CNN:                          10 epochs
   • Transfer Model - Frozen Base:        10 epochs
   • Transfer Model - Fine-tuning:         5 epochs (total 15 epochs)

   ANALYSIS:
      - Custom CNN achieved best results in just 10 epochs
      - Transfer models required more epochs (15 total) but performed worse
      - Efficiency: Custom CNN is more data-efficient
```

**Part 1.4:**

- Training Configuration

# Summary & Discussion

```
5. MODEL COMPLEXITY
────────────────────────────────────────────────────────────────────────────────

  • Transfer Model Parameters:        2,270,794
  • Custom CNN Parameters:            622,282
  • Parameter Ratio:                  3.6x (Transfer model is 3.6x larger)

  ANALYSIS:
    - Transfer model is significantly larger (3.6x) than Custom CNN (more complex)
    - Transfer model has 3.6x more parameters but performs worse
    - This demonstrates that more parameters (3.6x) do not guarantee better performance
    - Custom CNN: parameter-efficient (good efficiency)
    - Transfer model: parameter-inefficient (poor efficiency)

  SUMMARY:
    - Custom CNN achieves better performance with 622,282 parameters
    - More parameters (3.6x) ≠ better performance
    - Custom CNN demonstrates superior parameter efficiency
```

# Summary & Discussion

```
5. MODEL COMPLEXITY

────────────────────────────────────────────────────────────────────────────────

   • Transfer Model Parameters:        2,270,794
   • Custom CNN Parameters:            622,282
   • Parameter Ratio:                  3.6x (Transfer model is 3.6x larger)

   ANALYSIS:
      - Transfer model is significantly larger (3.6x) than Custom CNN (more complex)
      - Transfer model has 3.6x more parameters but performs worse
      - This demonstrates that more parameters (3.6x) do not guarantee better performance
      - Custom CNN: parameter-efficient (good efficiency)
      - Transfer model: parameter-inefficient (poor efficiency)

   SUMMARY:
      - Custom CNN achieves better performance with 622,282 parameters
      - More parameters (3.6x) ≠ better performance
      - Custom CNN demonstrates superior parameter efficiency
```

# THE END


| | |
|---|---|
| STUDENTS | AINEDEMBE DENIS<br>+256 788-674576 dembedenisjb@gmail.com, ainedembe.denis@stud.umu.ac.ug |
| | MUSINGUZI BENSON<br>+256 782 942245, musiben@gmail.com, musinguzi.benson@stud.umu.ac.ug |
| LECTURER | Dr. Sibitenda Harriet<br>+256 777 056581 hsibitenda@umu.ac.ug |