Computer Vision and Pattern Recognition

Practical Work 1: Introduction to Neural Networks

Object Detection and Classification

ADNET Willem

ENSIMAG - 3A - MMIS

December 2024

Abstract

This report presents a comprehensive study on deep learning approaches for object detection and classification. We implement and compare multiple convolutional neural network architectures, ranging from simple custom networks to advanced pre-trained models like ResNet18. The study encompasses both classification and bounding box regression tasks on a dataset containing three categories: airplanes, motorcycles, and faces. Our experiments demonstrate the effectiveness of transfer learning and the importance of architectural choices in achieving robust object detection performance.

Contents

1 Introduction

This practical work explores the fundamentals of deep learning for computer vision, specifically focusing on object detection tasks. The main objectives are:

- Understanding PyTorch framework and neural network training pipelines
- Implementing and comparing different CNN architectures
- Exploring transfer learning with pre-trained models
- Extending classification networks to bounding box regression
- Analyzing model performance and failure cases

The experiments were conducted on the Apple Silicon GPU (MPS). All models were trained on a custom dataset containing images from three categories: faces, motorcycles, and airplanes.

2 Dataset and Experimental Setup

2.1 Dataset Description

The dataset used in this work consists of images from three categories:

- **Faces**: Human facial images
- **Motorcycles**: Various motorcycle images
- **Airplanes**: Different airplane models and angles

The dataset is split into three subsets:

- **Training set**: Used for model parameter optimization
- **Validation set**: Used for hyperparameter tuning and model selection
- **Test set**: Used for final performance evaluation

All images are normalized to 224×224 pixels, following the ImageNet standard. Bounding box annotations are provided for each object and normalized to the $[0,1] \times [0,1]$ coordinate space.

2.2 Training Configuration

The standard training configuration across all experiments includes:

- Batch size: 32
- Default epochs: 20 (extended to 50-200 for specific experiments)
- Optimizer: Adam
- Loss functions: Cross-entropy (classification) + MSE (bounding box regression)

3 Part 1: Simple Classification Network

3.1 Initial Observations

The first experiments were conducted with the `SimpleDetector` architecture, a basic convolutional neural network designed for object classification. Initial training with the last epoch model revealed significant performance issues:

Table 1: Initial Model Performance (Last Epoch)

Dataset	**Overall**	**Face**	**Motorcycle**	**Airplane**
Train	48.3%	0.0%	94.4%	27.1%
Validation	46.8%	0.0%	92.2%	29.6%
Test	47.1%	0.0%	93.2%	37.0%

Key observations:

- The model shows severe bias toward predicting motorcycles
- Face detection completely fails (0% accuracy)
- Airplane detection is weak (≈30% accuracy)
- The model suffers from class imbalance issues

3.2 Model Selection Strategy

To address these issues, we implemented a best-model selection strategy based on validation performance. The model with the highest validation accuracy is saved. In case of a tie, the model with the lowest validation loss is selected. This approach significantly improved performance:

- After implementing best-model selection, test accuracy increased to **95%**
- Training for 50 epochs showed the model reached peak performance around epoch 30-40
- Extended training to 200 epochs achieved **100% accuracy** on all sets

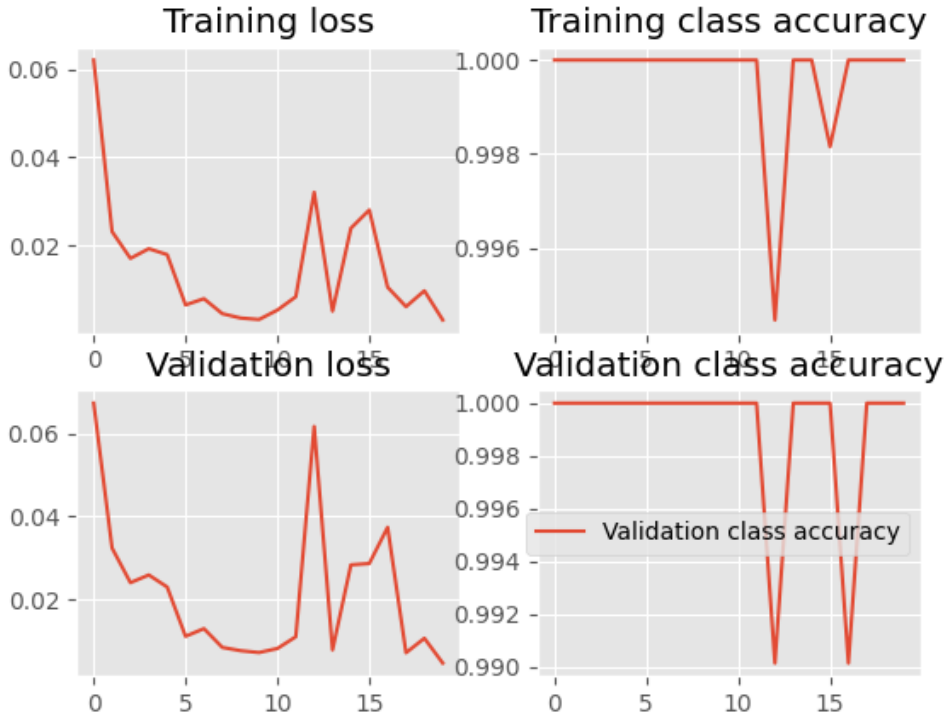


Figure 1: Training and validation loss/accuracy evolution over 20 epochs for SimpleDetector

3.3 Impact of Training Set Size

We conducted experiments with varying training set sizes to understand the relationship between data quantity and model performance:

Table 2: Performance vs. Training Set Size

Train Set Size	Epochs	Test Accuracy	Val Accuracy
1%	20-50	69%	73%
5%	50	85%	89%
20%	20	97%	96%
20%	40	99%	98%

Conclusion: A larger training set significantly improves performance, even with fewer epochs. With only 20% of the data, the model achieves near-perfect accuracy in 40 epochs.

3.4 Regularization Techniques

3.4.1 Effect of Dropout and Batch Normalization

We analyzed the impact of regularization techniques by training models with and without dropout and batch normalization layers.

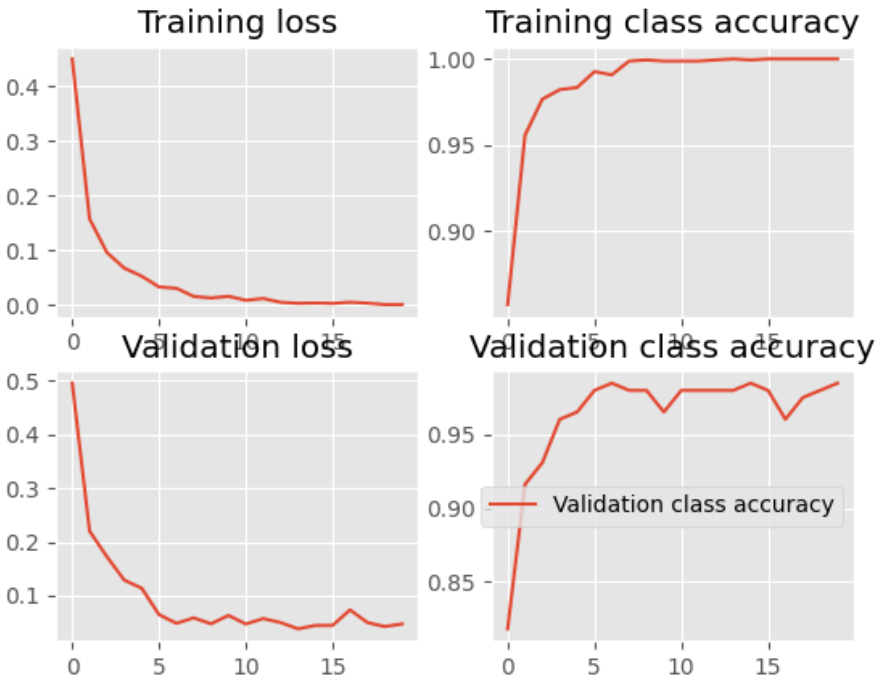


Figure 2: Training dynamics with dropout and batch normalization removed. Note the increased instability in validation metrics.

Observations:

- Without regularization, training is less stable
- Validation loss and accuracy show more fluctuation
- Risk of overfitting increases significantly
- Batch normalization provides smoother convergence

3.5 Web Image Testing

We tested the trained model on images downloaded from the internet:

Table 3: Performance on Web Images

Category	Correct	Total	Accuracy
Face	3	4	75%
Motorcycle	3	3	100%
Airplane	3	3	100%
Overall	**9**	**10**	**90%**

Failure case analysis: The model failed on an image containing multiple faces. This is expected since the training data only contained single-object images. This highlights the importance of training data diversity for model generalization.

4 Part 2: Advanced Architectures

4.1 Network Architectures

We implemented and compared four different architectures:

1. **SimpleDetector**: Basic CNN with few layers
2. **DeeperDetector**: Extended architecture with more convolutional layers
3. **VGGInspired**: Architecture inspired by VGG11 with 512 features instead of 4096
4. **ResNetObjectDetector**: ResNet18 with custom classification head

4.1.1 Architecture Visualizations

The network architectures were visualized using PlotNeuralNet[1], a LaTeX-based tool for drawing neural network architectures. The following figures present detailed visualizations of each model architecture, showing the progression from simple to complex designs.

[1] PlotNeuralNet: https://github.com/HarisIqbal88/PlotNeuralNet

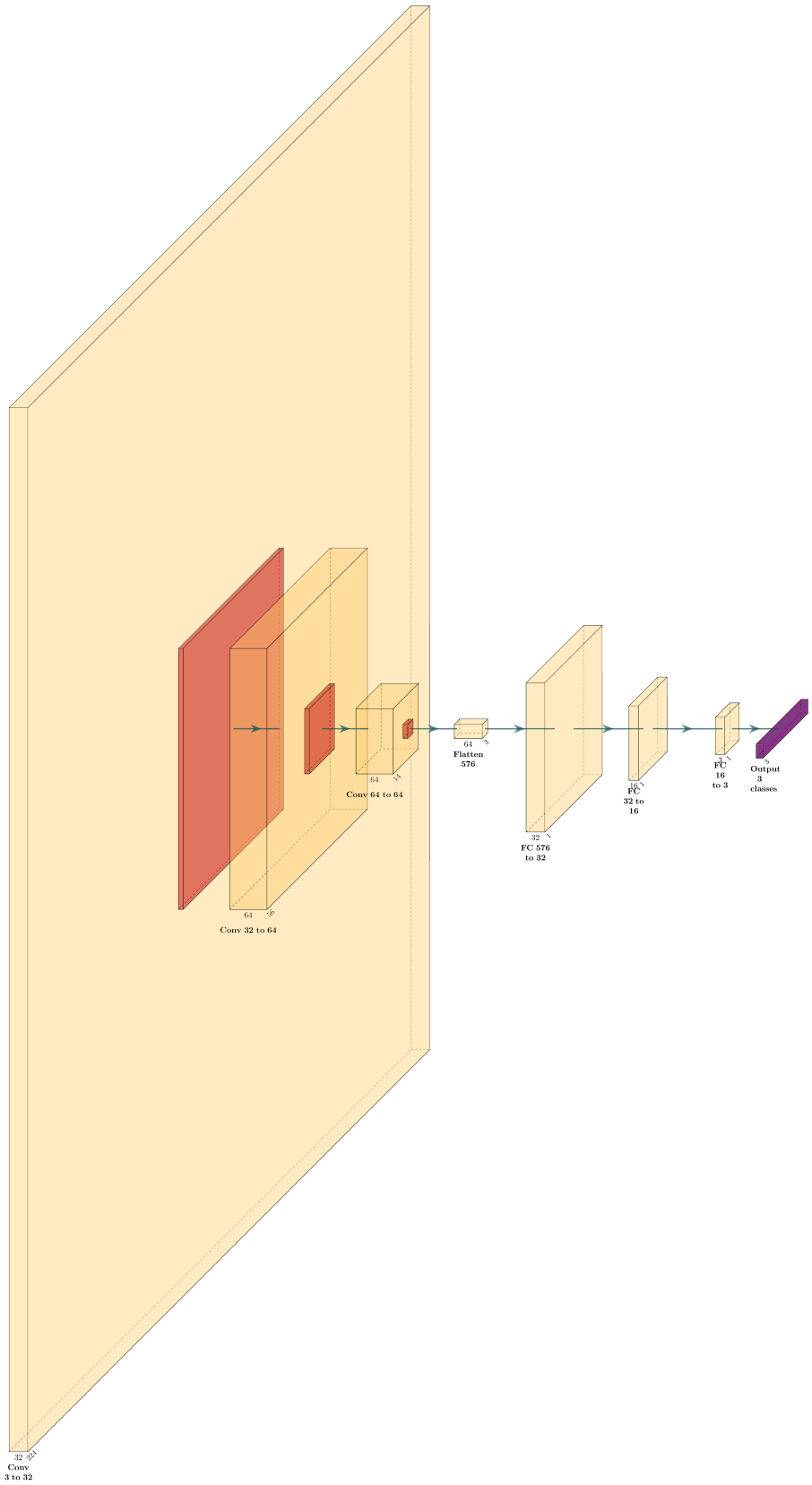


Figure 3: SimpleDetector architecture: A basic CNN with 3 convolutional blocks followed by 3 fully connected layers. Features: 32→64→64 channels with progressive pooling, resulting in 576 flattened features.

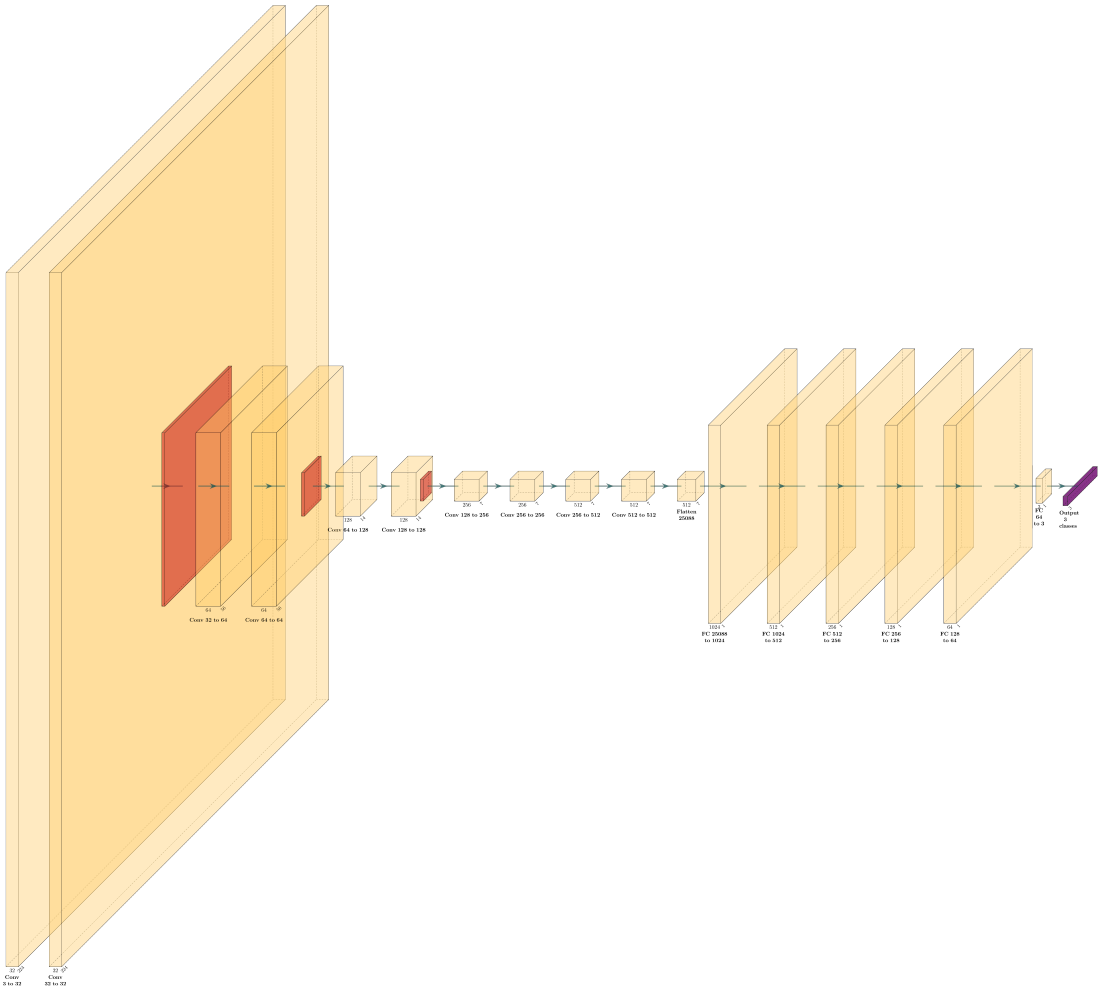


Figure 4: DeeperDetector architecture: An extended CNN with 10 convolutional layers organized in 4 blocks. Features progression: 32→64→128→256→512 channels with 4 pooling operations. The model uses 6 fully connected layers (25088→1024→512→256→128→64→3) for classification.

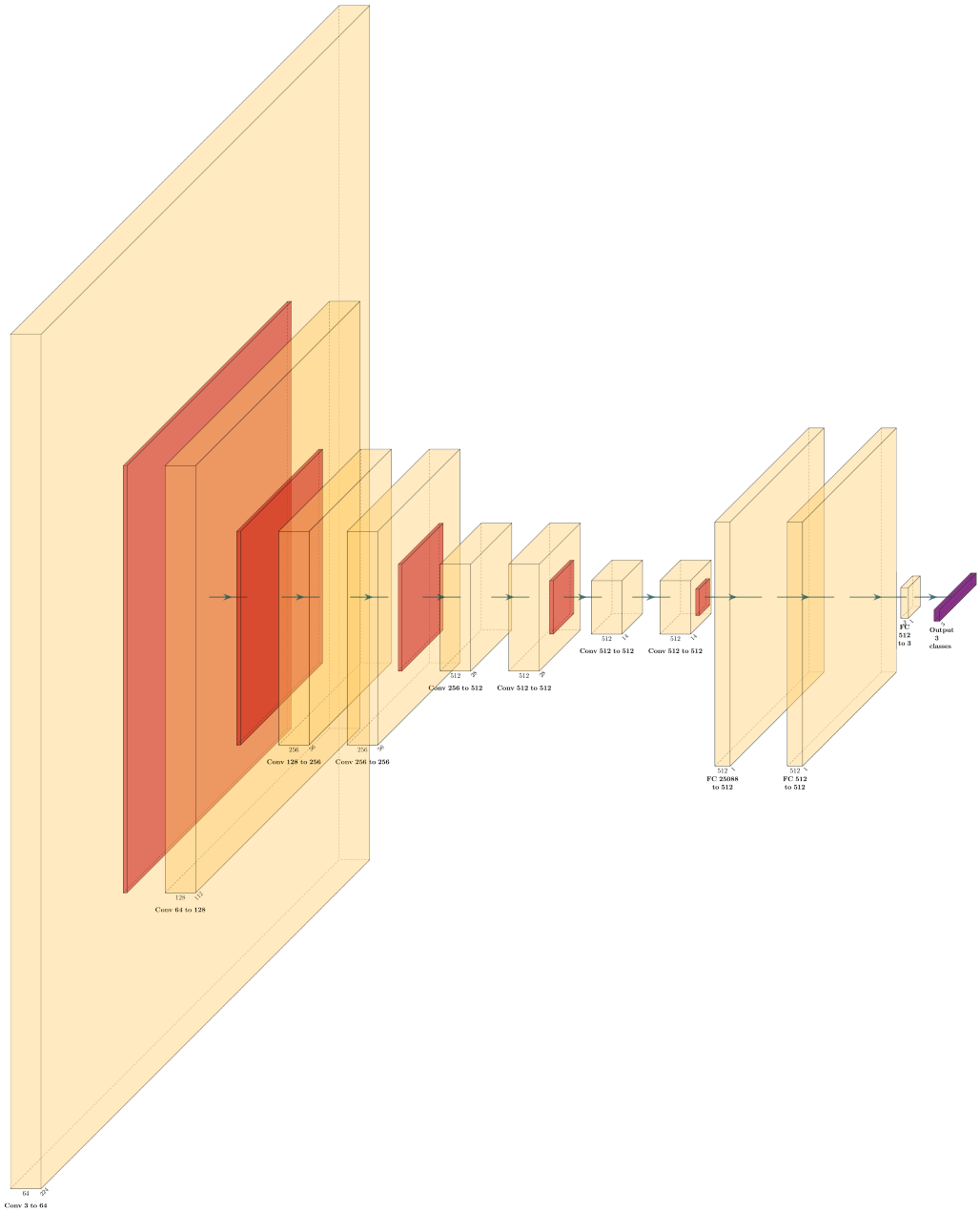


Figure 5: VGGInspired architecture: Based on VGG11 with simplified fully connected layers. Uses 8 convolutional layers with channel progression: 64→128→256×2→512×4. The FC layers are reduced from VGG's original 4096 to 512 features to prevent overfitting on the smaller dataset.

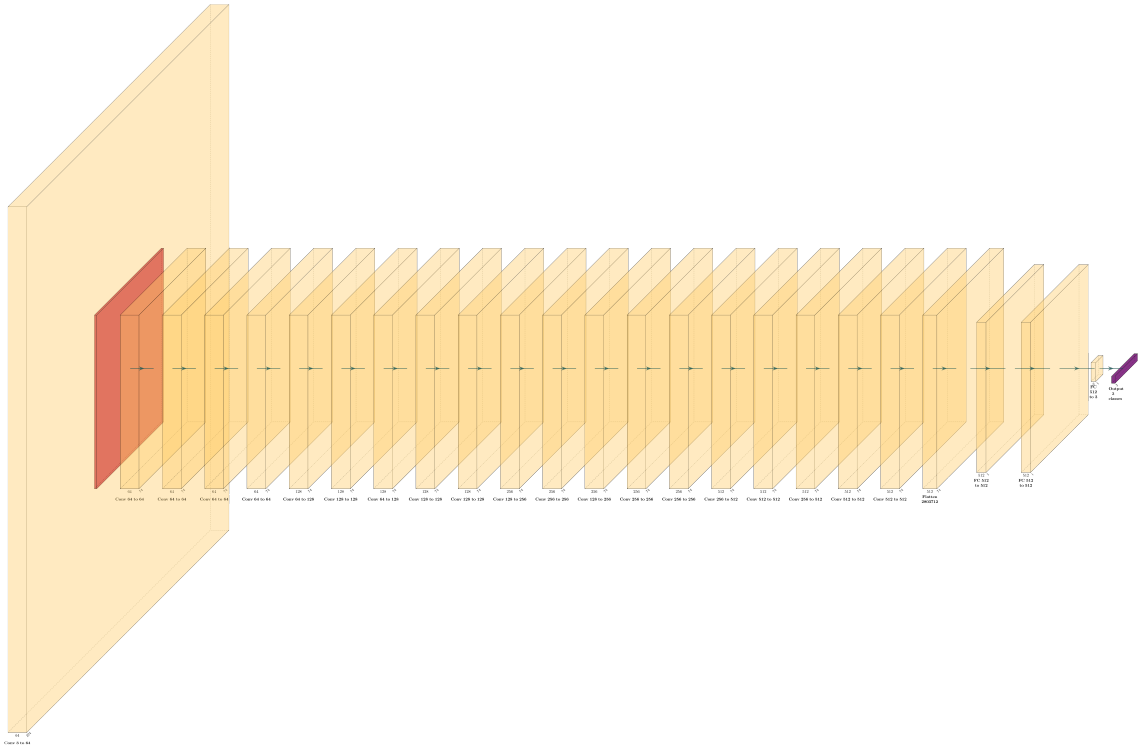


Figure 6: ResNet18 (Frozen) architecture: Pre-trained ResNet18 backbone with frozen convolutional features. The network contains 20 convolutional layers from the pre-trained model, followed by custom fully connected layers (512→512→3) for task-specific classification.

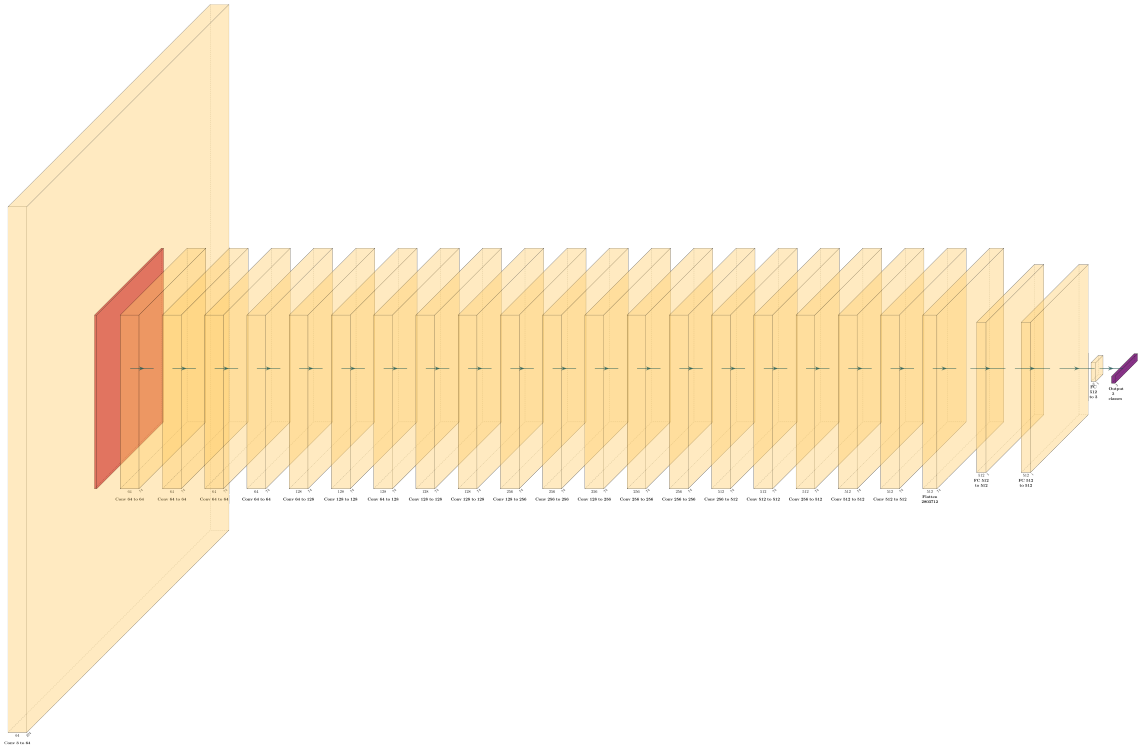


Figure 7: ResNet18 (Unfrozen) architecture: Same ResNet18 backbone as frozen version, but with all convolutional layers fine-tuned during training. This allows the entire network to adapt to the specific object detection task, particularly beneficial for bounding box regression.

4.2 Comprehensive Model Comparison

All models were trained for 20 epochs with identical hyperparameters to ensure fair comparison.

4.2.1 Training Performance

Table 4: Training Loss Comparison

Model	Final Loss	Mean Loss	Std Dev	Convergence
SimpleDetector	0.0107	0.1066	±0.1322	Moderate
DeeperDetector	0.0001	0.0431	±0.1122	Good
VGGInspired	0.0000	0.0278	±0.0470	Good
ResNet	0.0001	0.0055	±0.0160	Excellent

Table 5: Training Accuracy Comparison

Model	Final Accuracy	Mean Accuracy	Performance
SimpleDetector	99.88%	99.10% ±1.65%	Good
DeeperDetector	100.0%	98.38% ±5.15%	Very Good
VGGInspired	100.0%	98.95% ±2.06%	Very Good
ResNet	100.0%	99.99% ±0.02%	Excellent

4.2.2 Validation Performance

Table 6: Validation Performance Comparison

Model	Val Accuracy	Val Loss	Generalization
SimpleDetector	99.01%	0.0329	Good
DeeperDetector	99.51%	0.0414	Very Good
VGGInspired	99.01%	0.0617	Good
ResNet	100.0%	0.0001	Excellent

4.2.3 Computational Efficiency

Table 7: Epoch Time Comparison (20 epochs)

Model	Training Time
SimpleDetector	1m 38s
DeeperDetector	3m 02s
VGGInspired	7m 02s
ResNet	1m 58s

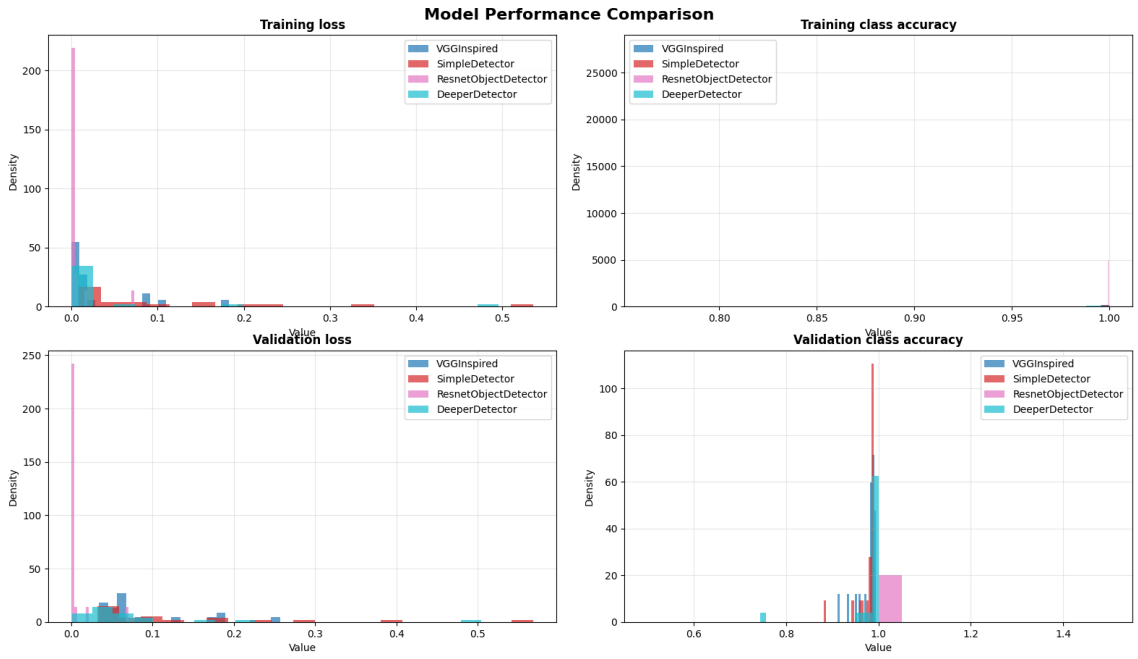


Figure 8: Comprehensive comparison of model performance metrics

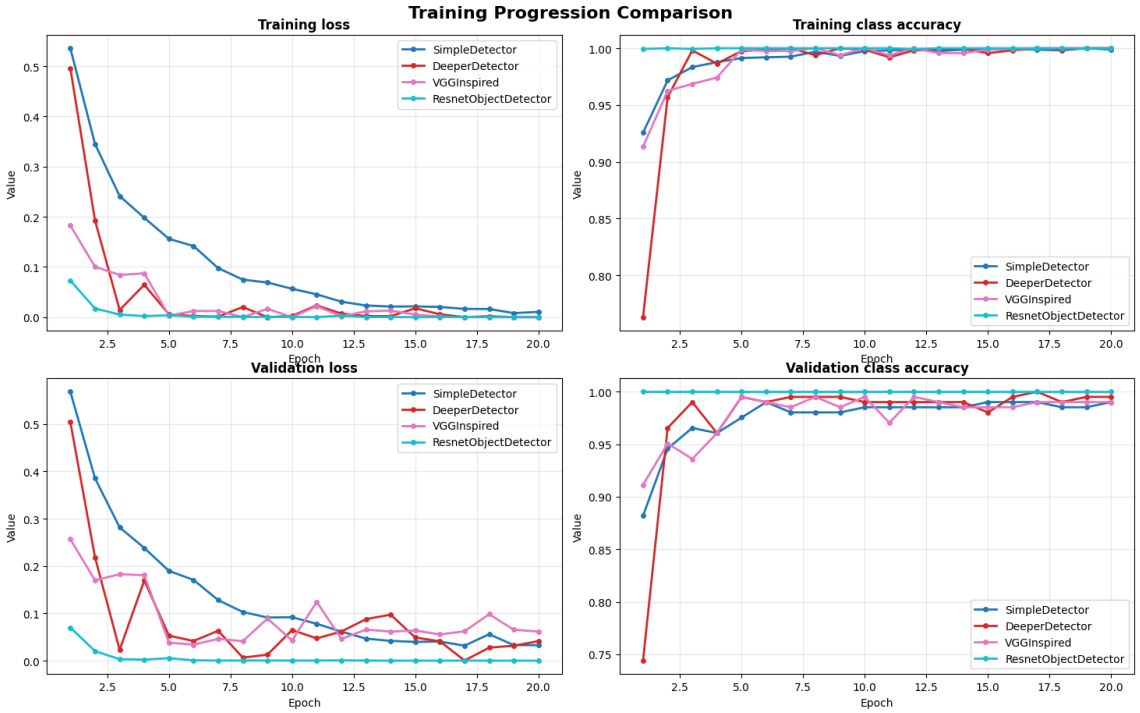


Figure 9: Training progression for all models over 20 epochs

4.3 Key Findings

4.3.1 Training Convergence

- **ResNet**: Best overall performance with lowest loss variance (± 0.0160) and near-perfect accuracy from early epochs due to pre-trained weights
- **VGGInspired & DeeperDetector**: Good convergence but require more epochs than ResNet
- **SimpleDetector**: Moderate performance with higher variance, indicating some training instability

4.3.2 Generalization Capability

- **ResNet**: Perfect validation accuracy with excellent generalization
- **DeeperDetector**: Best among non-pre-trained models
- **VGGInspired**: Good performance but slightly more prone to overfitting
- **SimpleDetector**: Largest gap between training and validation performance

4.3.3 Transfer Learning Impact

- Pre-trained ResNet achieves excellent results with minimal training
- Transfer learning significantly reduces required training time
- From-scratch models require substantially more training for comparable results

4.3.4 Computational Efficiency

- **ResNet**: Best performance/computation ratio
- **SimpleDetector**: Fastest but lowest final performance
- **VGGInspired**: Highest computational cost (4.3× slower than SimpleDetector) with marginal gains over DeeperDetector

5 Part 3: Bounding Box Regression

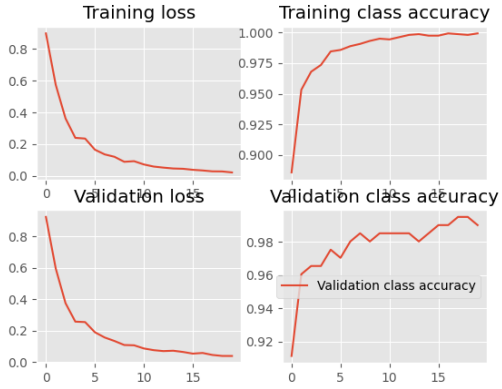
5.1 Multi-Task Learning Architecture

We extended our classification networks to perform both classification and bounding box regression simultaneously. This multi-task learning approach shares convolutional features between two task-specific heads:

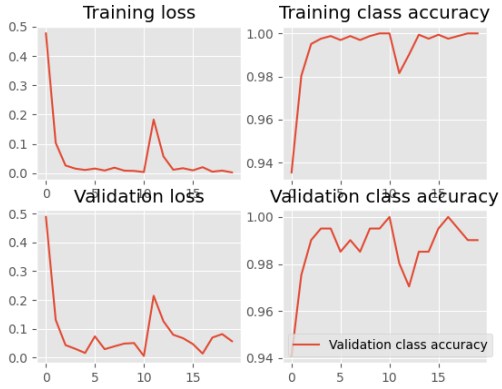
- **Classification head**: Original fully connected layers for category prediction
- **Regression head**: New branch with structure: Linear → ReLU → Linear → ReLU → Linear → Sigmoid

The sigmoid activation ensures bounding box coordinates remain in the normalized [0,1] range. Notably, dropout is omitted from the regression branch to maintain precise coordinate predictions.

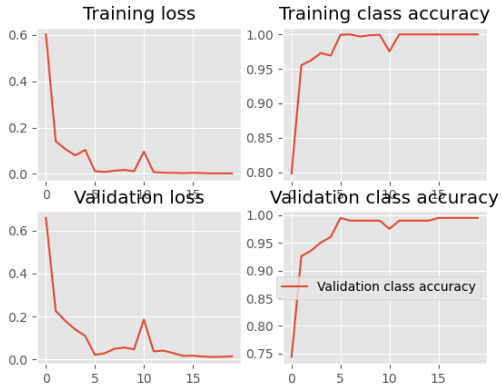
5.2 Training Dynamics



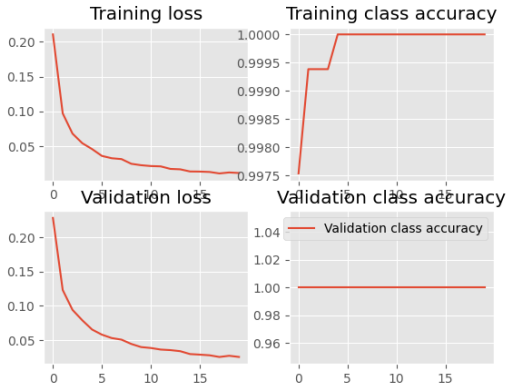
(a) SimpleDetector



(b) DeeperDetector



(c) VGGInspired



(d) ResNetObjectDetector

Figure 10: Training convergence with bounding box regression for all architectures

5.3 Transfer Learning: Frozen vs. Unfrozen Features

A critical experiment compared frozen ResNet features (transfer learning without fine-tuning) versus unfrozen features (full model fine-tuning).

Table 8: Frozen vs. Unfrozen ResNet: Bounding Box Performance

Category	Model	Mean Distance	Mean IoU	IoU $\geq$ 0.5	IoU $\geq$ 0.7
Face	Frozen	0.2799	0.4242	40.0%	5.0%
Face	Unfrozen	0.1196	0.6879	91.2%	52.6%
Motorcycle	Frozen	0.2442	0.4242	40.0%	5.0%
Motorcycle	Unfrozen	0.1039	0.5718	70.3%	17.6%
Airplane	Frozen	0.2520	0.5339	64.4%	12.3%
Airplane	Unfrozen	0.0926	0.7895	100.0%	91.8%

Key insight: Unfreezing the backbone dramatically improves bounding box localization. While classification remains excellent in both cases, regression benefits significantly from feature

fine-tuning. The improvement is most pronounced for faces (47.6% increase in IoU ≥ 0.7) and airplanes (79.5% increase).

5.4 Comprehensive Bounding Box Performance

All models were trained for 20 epochs with bounding box regression enabled.

5.4.1 Overall Performance Metrics

Table 9: Bounding Box Regression Performance (Test Set)

Model	**Mean IoU**	**Mean Distance**	**IoU ≥ 0.5**	**IoU ≥ 0.7**
SimpleDetector	0.803	0.083	97.1%	77.5%
DeeperDetector	0.833	0.071	100.0%	91.2%
VGGInspired	0.852	0.062	100.0%	97.1%
ResNet (Frozen)	0.660	0.156	85.3%	47.1%
ResNet (Unfrozen)	0.758	0.104	97.5%	78.9%

5.4.2 Per-Category Analysis

Table 10: Detailed Per-Category Bounding Box Performance

Model	**Category**	**Mean IoU**	**Distance**	**Good (%)**	**Very Good (%)**
SimpleDetector	Face	0.646	0.141	89.5	31.6
SimpleDetector	Motorcycle	0.848	0.071	100.0	93.2
SimpleDetector	Airplane	0.881	0.051	100.0	97.3
DeeperDetector	Face	0.754	0.092	100.0	77.2
DeeperDetector	Motorcycle	0.872	0.059	100.0	97.3
DeeperDetector	Airplane	0.855	0.065	100.0	95.9
VGGInspired	Face	0.809	0.071	100.0	91.2
VGGInspired	Motorcycle	0.851	0.068	100.0	98.6
VGGInspired	Airplane	0.886	0.048	100.0	100.0
ResNet (Frozen)	Face	0.499	0.209	50.9	15.8
ResNet (Frozen)	Motorcycle	0.726	0.139	100.0	60.8
ResNet (Frozen)	Airplane	0.718	0.132	97.3	57.5
ResNet (Unfrozen)	Face	0.688	0.120	91.2	52.6
ResNet (Unfrozen)	Motorcycle	0.781	0.104	100.0	86.5
ResNet (Unfrozen)	Airplane	0.790	0.093	100.0	91.8

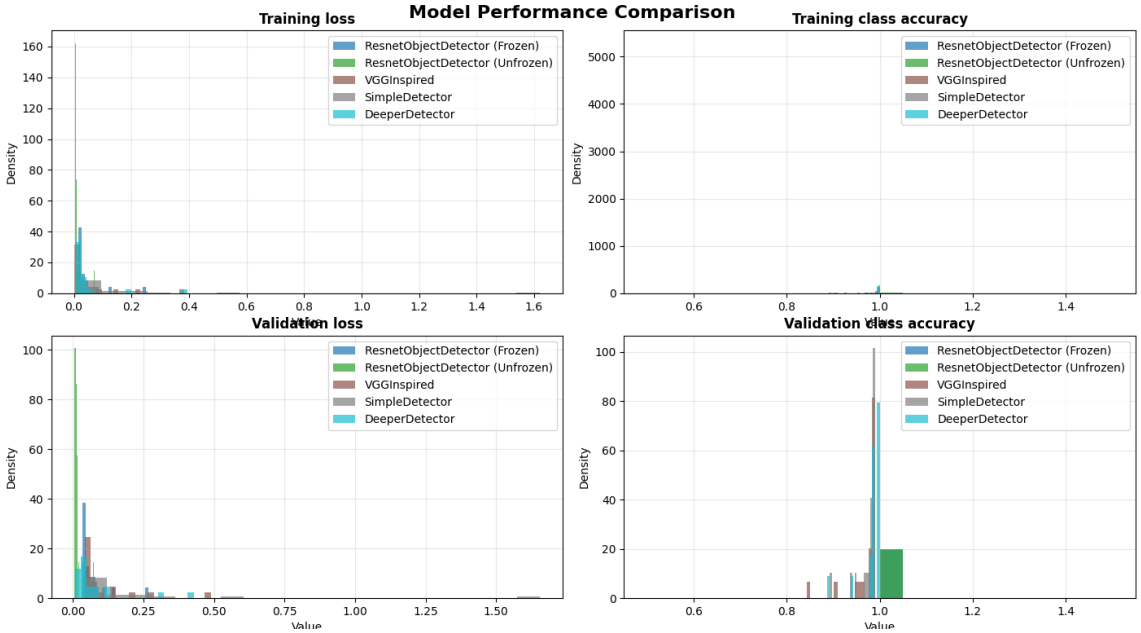


Figure 11: Comparative histograms of model performance with bounding box regression

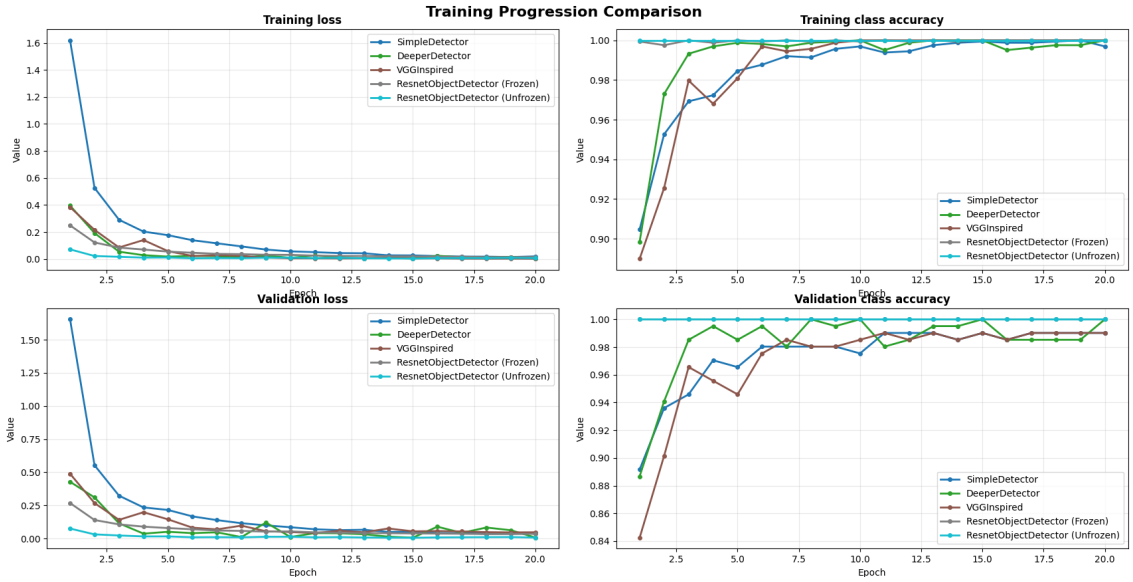


Figure 12: Training progression comparison for all models with bounding box regression

5.5 Key Observations

- **VGGInspired**: Achieves the best overall bounding box performance with 100% good detection rate and 97.1% very good detection rate, including perfect airplane localization (100% IoU ≥ 0.7)
- **DeeperDetector**: Excellent performance with 100% good detection rate and 91.2% very good detection rate, demonstrating that custom deeper architectures can rival sophisticated pre-trained models for this task
- **Transfer Learning Paradox**: Surprisingly, custom architectures (VGGInspired, DeeperDetector) consistently outperform both frozen and unfrozen ResNet for bounding box regression, despite ResNet's superior classification performance

- **Face Localization Challenge**: All models show weakest performance on faces. SimpleDetector achieves only 31.6% very good detection for faces, while even the best model (VGGInspired) reaches 91.2%
- **Frozen Features Limitation**: ResNet with frozen features performs notably poorly (47.1% very good detection overall), highlighting that bounding box regression requires feature adaptation to the specific task
- **Category Imbalance**: Airplane localization is generally easiest (most models achieve $\geq$95.9% very good detection), followed by motorcycles, then faces

5.6 Failure Case Analysis

Testing on web images revealed an interesting failure case:

Figure 13: Failure case: ResNet (unfrozen) correctly classifies the airplane but produces an incorrect bounding box. This suggests the regression head requires more specialized training or architecture adaptation.

The model correctly identifies the object category but fails to accurately localize it. This indicates that while the classification head leverages pre-trained features effectively, the regression head struggles to adapt these features for precise spatial localization.

6 Discussion and Conclusions

6.1 Summary of Findings

This comprehensive study on object detection revealed several important insights:

6.1.1 Classification Performance

- Pre-trained ResNet18 demonstrates superior performance for classification tasks, achieving near-perfect accuracy with minimal training
- Transfer learning significantly accelerates convergence and improves generalization
- Simple custom architectures can achieve excellent results (>95%) with sufficient training data
- Model selection based on validation performance is crucial; last-epoch models often underperform

6.1.2 Bounding Box Regression

- Custom deeper architectures (VGGInspired, DeeperDetector) surprisingly outperform transfer learning approaches for bounding box regression
- Unfreezing pre-trained features is essential for regression tasks; frozen features yield poor localization (47% vs. 79% very good detection)
- Multi-task learning with shared features is effective but requires careful architecture design
- Face localization remains the most challenging category across all models

6.1.3 Training Dynamics

- Regularization (dropout, batch normalization) is critical for training stability
- Training set size has a strong impact on performance; even 20% of data yields 97% accuracy
- Model complexity vs. training time presents a trade-off: VGGInspired takes 4.3× longer than SimpleDetector for marginal gains in classification
- Early stopping based on validation metrics prevents overfitting

6.2 Architectural Considerations

6.2.1 For Classification

- **Best choice**: ResNet with transfer learning (fastest convergence, highest accuracy)
- **Resource-constrained**: SimpleDetector (fastest training, acceptable performance)
- **From-scratch training**: DeeperDetector (good balance of performance and efficiency)

6.2.2 For Bounding Box Regression

- **Best accuracy**: VGGInspired (97.1% very good detection, perfect airplane localization)
- **Best efficiency/accuracy**: DeeperDetector (91.2% very good detection, 43% faster than VGG)
- **Transfer learning**: Requires unfrozen features; consider task-specific adaptations

6.3 Limitations and Future Work

Several limitations were identified during this study:

1. **Single-object limitation**: Models fail on images with multiple objects, as training data contained only single instances
2. **Category imbalance**: Face localization consistently underperforms, suggesting need for category-specific architectures or data augmentation
3. **Fixed input size**: 224×224 normalization may lose important details for precise localization
4. **Limited architecture exploration**: Modern architectures like EfficientNet, YOLO, or SSD could provide better performance

Potential improvements:

- Implement attention mechanisms to focus on object regions
- Explore anchor-based detection methods (YOLO, Faster R-CNN)
- Apply data augmentation to increase robustness
- Implement multi-scale feature fusion for better localization
- Address class imbalance through weighted loss functions or oversampling

6.4 Practical Recommendations

Based on our experimental results, we recommend:

1. **Start with transfer learning**: Pre-trained models provide excellent starting points, especially for classification
2. **Always validate unfrozen training**: For regression tasks, unfreezing features is often essential despite increased computational cost
3. **Prioritize validation-based model selection**: Last-epoch models frequently underperform; save and evaluate based on validation metrics
4. **Monitor per-category performance**: Aggregate metrics can hide category-specific failures
5. **Balance complexity and efficiency**: VGGInspired's 4× longer training time may not justify the marginal improvement over DeeperDetector for many applications
6. **Consider task-specific architectures**: What works best for classification may not be optimal for regression

6.5 Conclusion

This practical work successfully demonstrated the implementation and comparison of multiple deep learning architectures for object detection. While transfer learning with ResNet18 proved highly effective for classification (100% accuracy in 10-20 epochs), custom architectures like VGGInspired achieved superior bounding box localization (97.1% very good detection vs. 78.9% for unfrozen ResNet).

The experiments highlighted the importance of architectural choices, regularization techniques, and training strategies in achieving robust performance. The surprising result that custom architectures outperform transfer learning for bounding box regression suggests that pre-trained features, while excellent for classification, may require substantial task-specific adaptation for precise spatial localization tasks.

Overall, this work provides a comprehensive foundation for understanding neural network design for computer vision tasks and demonstrates both the power and limitations of current approaches to object detection.

Acknowledgments

This work was conducted as part of the Computer Vision and Pattern Recognition course at ENSIMAG (Grenoble INP). We acknowledge the use of the ENSIMAG GPU cluster for computational resources.

Network architecture visualizations were created using PlotNeuralNet[2], developed by Haris Iqbal.

[2] https://github.com/HarisIqbal88/PlotNeuralNet

References

[1] Iqbal, H. (2018). PlotNeuralNet: Python and LaTeX package for drawing neural networks. https://github.com/HarisIqbal88/PlotNeuralNet

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

A Command Reference

A.1 Training Commands

You must install uv to ensure everything is in the correct environment. Follow the installation instructions at https://docs.astral.sh/uv/getting-started/installation/:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Listing 1: Install uv (Unix/macOS)

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 |
    iex"
```

Listing 2: Install uv (Windows)

Alternatively, you can use package managers like Homebrew (`brew install uv`), pip (`pip install uv`), or conda (`conda install -c conda-forge uv`).

Once uv is installed, set up the project environment:

```
uv sync
```

Listing 3: Synchronize project dependencies

```
source .venv/bin/activate
```

Listing 4: Activate virtual environment

```
uv run python train.py --model simple --save-model true --epoch-size 20
```

Listing 5: Train SimpleDetector model

```
uv run python train.py --model resnet --save-model true --epoch-size 20
```

Listing 6: Training with ResNet

A.2 Evaluation Commands

```
uv run python eval.py output/best_model.pth
```

Listing 7: Evaluate trained model

A.3 Prediction Commands

```
uv run python predict.py --directory output/test_data.csv --show-all-images true
```

Listing 8: Predict on test set

```
uv run python predict.py --filename path/to/image.jpg --model best --save-file
    true
```

Listing 9: Predict on single image

A.4 Comparison Commands

```
uv run python compare_models.py
```

Listing 10: Compare all models

A.5 Network Visualization

To create all the Python files that represent the models you must run:

```
uv run python src/draw_network/archs_creation/create_files.py
```

Listing 11: Generate network representation files

Files will be created or updated in the directory: **src/draw_network/archs/**

Then you can run the following command that will create the PDF, PNG and keep the LaTeX files in the directory: **src/draw_network/network_diagrams/latex_files**. It will also display all the PDF files.

```
./draw_networks.sh
```

Listing 12: Generate network diagrams