Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Deep Learning

Image Recognition and Object Detection on the xView Satellite Dataset

Melen Laclais, melen.laclais@alumnos.upm.es
Léo Lamy, leo.lamy@alumnos.upm.es
Adrián García-Pozuelo Fornieles, a.garcia-pozuelo@alumnos.upm.es

Madrid, 2026

Contents

1 Introduction and Dataset Analysis

Here we describe the xView datasets and the specific subsets used for this project. The benchmark was introduced by Lam et al..[1]

- **Image Recognition:** About the 13 categories and the 18,746 training / 2,365 test images resized to 224×224.
- **Object Detection:** About the 4 categories and the use of 640×640 crops.

1.1 Class Distribution

The dataset presents a significant class imbalance, which is one of the recurring challenges throughout the experiments. Table 1 summarizes the number of samples per class.

Table 1: Number of samples per class in the dataset.

Class	**Samples**
Building	3594
Small car	3324
Truck	2210
Bus	1768
Shipping container	1523
Storage tank	1469
Dump truck	1236
Motorboat	1069
Excavator	789
Fishing vessel	706
Cargo plane	635
Pylon	312
Helipad	111

The most represented class (*Building*, 3594 samples) has over 32 times more instances than the least represented class (*Helipad*, 111 samples). This imbalance heavily influences model predictions, as classifiers tend to favor the majority class.

[1] Darius Lam et al. "xView: Objects in Context in Overhead Imagery". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018.

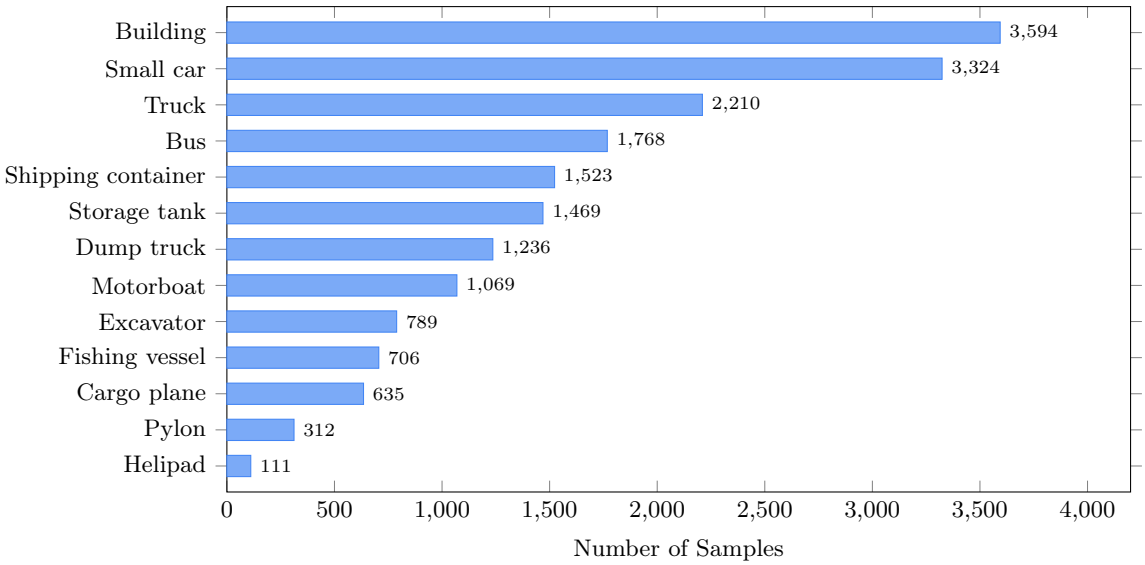


Figure 1: Class distribution across the 13 categories. The dataset is highly imbalanced, with *Building* and *Small car* dominating.

1.2 Object Detection Dataset Analysis

For the object detection phase, the dataset consists of 7,606 satellite images cropped to 640×640 pixels, containing a total of 481,112 annotated objects. The dataset was split into training (6,845 images) and validation (761 images) subsets, maintaining a 90/10 ratio.

The analysis of this dataset reveals three major challenges that heavily influence the choice and training of object detection architectures: severe class imbalance, high object density, and extreme scale variations.

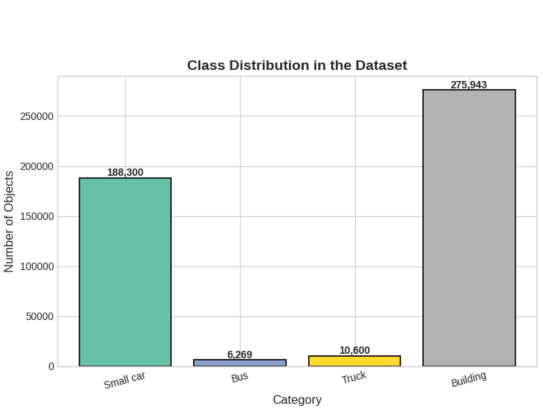
1.2.1 Class Imbalance and Scale Differences

The dataset is strictly limited to four categories, but their distribution is highly skewed. As detailed in Table 2, the *Building* and *Small car* classes dominate the dataset, accounting for 96.5% of all annotations. Conversely, *Bus* and *Truck* are severely underrepresented.

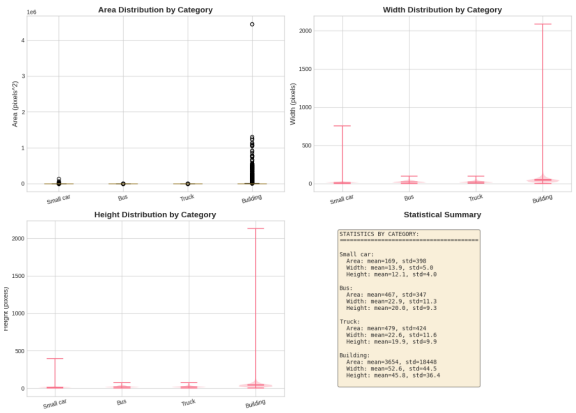
Table 2: Class distribution and average bounding box area in the object detection dataset.

Category	**Number of Objects**	**Percentage**	**Mean Area (px^2)**
Building	275,943	57.4%	3654
Small car	188,300	39.1%	169
Truck	10,600	2.2%	479
Bus	6,269	1.3%	467
Total	**481,112**	**100.0%**	**2179**

Beyond frequency, there is a significant disparity in object scale. The mean area of a *Building* (3654 px^2) is more than 20 times larger than that of a *Small car* (169 px^2).



(a) Class frequency distribution.



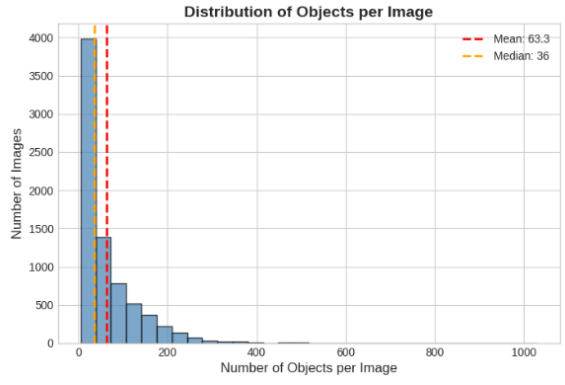
(b) Bounding box area distribution per class.

Figure 2: Visual analysis of the object detection dataset categories.

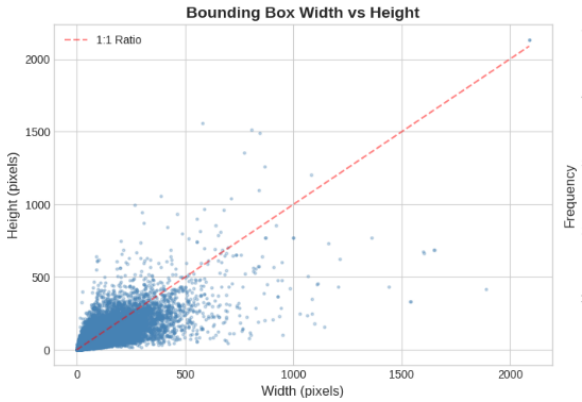
1.2.2 Object Density and Bounding Box Dimensions

Satellite imagery fundamentally differs from standard object detection datasets (like COCO or Pascal VOC) due to its extreme object density and the minuscule size of the targets.

- **High Density:** The average image contains 63.3 objects, with a median of 36. However, highly dense scenes can contain up to 1,029 objects in a single 640×640 crop. This high density requires strict Non-Maximum Suppression (NMS) tuning during inference to avoid overlapping bounding box suppression.
- **Small Object Problem:** The global median bounding box area is only 550 px^2. More critically, **61.2% of all objects are considered "small"** (area $< 32 \times 32$ pixels).



(a) Histogram of objects per image.



(b) Bounding box width vs height.

Figure 3: Density and spatial dimensions of the annotated objects.

The mean bounding box dimensions are 36.4×31.7 pixels (mean aspect ratio of 1.25), indicating that most objects are roughly square (especially small vehicles). These spatial characteristics dictate that the chosen detection architectures must utilize high-resolution feature maps (e.g., Feature Pyramid Networks) and anchor boxes specifically tailored to very small aspect ratios.

2 Feedback on "Cesvima" Resources

We mainly used Colab and Kaggle but sometimes we tried to use Cesvima. It was pretty easy to launch it via ssh on VScode. Then, to prepare the shell script, with the help of llms, it was simple to get the run prepared. The main issue we met is the queue time which is super high dependant on the time you launch the run. Also, the GPU is not as efficient as Kaggle or Colab. So we only used cesvima when the we ran out of GPU resources on Kaggle or Colab.

3 Phase 1: Image Recognition

3.1 Custom Feedforward Neural Networks (ffNNs)

This section documents a systematic exploration of feedforward neural network architectures for classifying satellite imagery into 13 categories. Starting from a minimal baseline, we progressively investigate the effect of adding hidden layers, varying their width, tuning training hyperparameters (batch size, epochs, learning rate), changing optimizers, and applying loss scaling and focal loss. All images are 224×224 RGB inputs (flattened to 150,528 features for the dense layers). A 85/15 train/validation split is used throughout, and the same set of callbacks is applied unless otherwise noted.

3.1.1 Common Training Setup

All experiments share the following callback configuration:

- `ModelCheckpoint` monitoring `val_accuracy`, saving only the best model.
- `ReduceLROnPlateau` on `val_accuracy` with a factor of 0.1 and patience of 10 epochs.
- `EarlyStopping` on `val_accuracy` with patience of 40 epochs.
- `TerminateOnNaN` to halt training if loss becomes NaN.

3.1.2 Experiment 1: Baseline Architecture

The starting point is the simplest possible network: a single-layer model that directly maps the flattened input to 13 output classes.

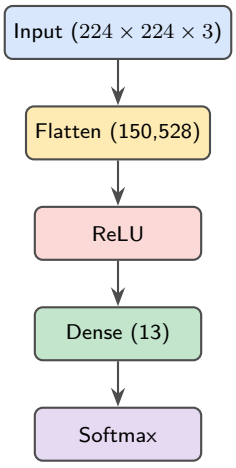


Figure 4: Experiment 1: Baseline architecture. Note that the ReLU activation after Flatten operates on raw pixel values and does not transform negative values (pixels are already in $[0, 255]$), making it effectively a no-op.

Table 3: Hyperparameters for Experiment 1.

Parameter	**Value**
Optimizer	Adam (lr=10^{-3}, β_1=0.9, β_2=0.999, ϵ=10^{-8}, AMSGrad, clipnorm=1.0)
Loss	Categorical Crossentropy
Batch size	16
Epochs	20

Results. The model achieved a training accuracy of 51.00% and a validation accuracy of 41.92%, with a training loss of 836.04 and a validation loss of 1378.45. The large gap between training and validation performance suggests overfitting.

Confusion matrix analysis. Table 4 summarizes the best and worst performing classes.

Table 4: Confusion matrix highlights for Experiment 1.

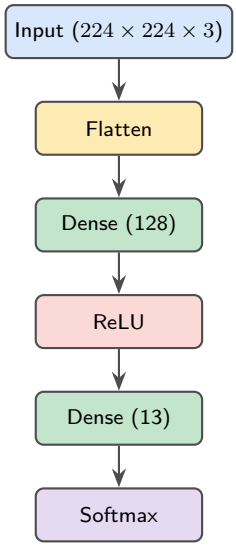
Class	**Correct**	**Most Confused With**	**Wrong**
Best performing classes			
Small Car	0.77	Motor Boat	0.07
Cargo Plane	0.65	Building	0.32
Building	0.54	Cargo Plane	0.11
Pylon	0.50	Building	0.39
Shopping Cont.	0.49	Building	0.12
Motor Boat	0.45	Small Car	0.19
Worst performing classes			
Helipad	0.00	Building	0.54
Storage Tank	0.04	Building	0.47
Fishing Vessel	0.17	Building	0.23
Truck	0.40	Small Car	0.23

Key observations.

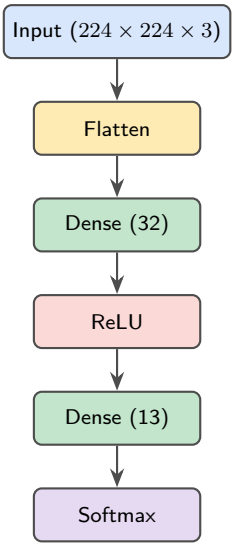
1. The class imbalance is a dominant factor. Minority classes like *Helipad* (111 samples) and *Storage Tank* are overwhelmingly misclassified as *Building*.
2. The ReLU activation placed directly after `Flatten` has no practical effect since pixel values are non-negative by definition.
3. The use of Categorical Crossentropy on an imbalanced dataset may not be ideal. Focal Loss could help penalize confident but wrong predictions on minority classes.
4. ReLU variants such as Leaky ReLU or ELU could be explored to mitigate the dying ReLU problem in deeper architectures.

3.1.3 Experiments 2 and 3: Adding a Hidden Dense Layer

To test whether additional capacity improves classification, two variants were explored: one with a 128-unit hidden layer and one with a 32-unit hidden layer. All other hyperparameters remain identical to Experiment 1.



(a) Experiment 2: Dense(128) hidden layer.



(b) Experiment 3: Dense(32) hidden layer.

Figure 5: Two-layer architectures explored in Experiments 2 and 3.

Table 5: Results for Experiments 2 and 3, compared with the baseline.

Experiment	**Hidden Units**	**Train Acc.**	**Val Acc.**	**Val Loss**
1 (Baseline)	None	51.00%	41.92%	1378.45
2	128	16.85%	19.25%	2.39
3	32	16.06%	19.25%	2.38

Analysis. Both deeper architectures performed dramatically worse than the baseline. The validation accuracy of approximately 19.25% for both models corresponds roughly to the proportion of the majority class (*Building*), which suggests the models collapsed into predicting a single class for all inputs. Increasing network complexity without adjusting the training configuration (more epochs, larger batches, or different learning rates) led to worse convergence. The confusion matrices confirm that nearly all predictions fell into the *Building* class.

3.1.4 Experiment 4: Dense(32) with Increased Batch Size and Epochs

This experiment retains the Dense(32) architecture from Experiment 3 but increases the batch size to 32 and the number of epochs to 40, testing whether the convergence failure was caused by insufficient training.

Table 6: Hyperparameter changes and results for Experiment 4.

Experiment	**Batch Size**	**Epochs**	**Train Acc.**	**Val Acc.**
3 (Dense 32)	16	20	16.06%	19.25%
4 (Dense 32, more)	32	40	19.34%	19.25%

Analysis. Doubling both the batch size and the number of epochs produced only a marginal improvement in training accuracy (from 16.06% to 19.34%), while the validation accuracy remained unchanged at 19.25%. The model still collapsed into predicting the majority class. This indicates that the issue is not

simply a matter of training duration, but rather a fundamental architectural limitation: the bottleneck created by passing 150,528 features through a 32-unit layer causes too much information loss.

3.1.5 Experiment 5: Baseline Architecture with Increased Batch Size and Epochs

Returning to the original single-layer architecture but training with the increased hyperparameters (batch size 32, 40 epochs) to establish a fair comparison with Experiment 4.

Table 7: Results for Experiment 5 compared with the original baseline.

Experiment	Batch / Epochs	Train Acc.	Val Acc.	Val Loss
1 (Baseline)	16 / 20	51.00%	41.92%	1378.45
5 (Baseline+)	32 / 40	39.42%	39.71%	1970.30

Table 8: Per-class metrics for Experiment 5.

Class	Recall	Precision	Specificity	Dice
Cargo plane	49.59%	53.57%	98.57%	51.50%
Small car	87.56%	47.06%	79.00%	61.22%
Bus	30.16%	31.62%	92.90%	30.88%
Truck	0.23%	16.67%	99.85%	0.44%
Motorboat	37.16%	24.92%	93.09%	29.83%
Fishing vessel	6.21%	34.62%	99.53%	10.53%
Dump truck	45.92%	30.84%	93.18%	36.90%
Excavator	2.92%	25.00%	99.58%	5.24%
Building	50.83%	51.40%	88.54%	51.11%
Helipad	0.00%	0.00%	99.95%	0.00%
Storage tank	34.17%	23.06%	90.87%	27.54%
Shipping container	24.34%	38.54%	96.58%	29.84%
Pylon	3.23%	11.77%	99.59%	5.06%

Global metrics for Experiment 5.

Analysis. The baseline architecture with more epochs achieved a mean accuracy of 39.71%, mean recall of 28.64%, and mean precision of 29.93%. While the overall validation accuracy is comparable to the original baseline, the gap between training and validation accuracy has narrowed significantly (from $\sim$9% to $\sim$0.3%), indicating less overfitting. However, the per-class breakdown reveals extreme disparities: *Small car* reaches 87.56% recall, while *Helipad*, *Truck*, and *Excavator* are essentially never predicted correctly. This confirms that the validation accuracy ceiling for this architecture is approximately 40%.

3.1.6 Experiments 6, 7 and 8: Dense(13) Hidden Layer with Learning Rate Variations

Having observed that adding wide layers causes collapse, a narrower hidden layer matching the number of output classes (13 units) was tested. Three learning rates were evaluated: 10^{-3}, 2×10^{-3}, and 3×10^{-3}.

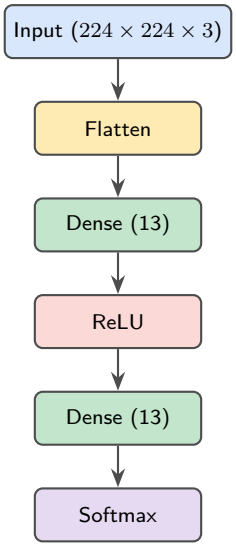


Figure 6: Architecture for Experiments 6, 7, and 8: Flatten → Dense(13) → ReLU → Dense(13) → Softmax.

Table 9: Results across learning rate variations (Experiments 6, 7, 8). All use batch size 32 and 40 epochs.

Exp.	Learning Rate	Train Acc.	Val Acc.	Val Loss
6	1×10^{-3}	17.46%	19.25%	2.41
7	2×10^{-3}	18.95%	19.25%	2.37
8	3×10^{-3}	19.52%	19.25%	2.32

Analysis. None of the three learning rates produced meaningful learning. All models converged to the same validation accuracy of 19.25%, again corresponding to majority-class prediction. A 13-unit bottleneck is far too narrow to represent 150,528 input features, causing a complete information loss that no learning rate can compensate for. This series of experiments confirms that deeper networks require careful capacity planning and that simply making the hidden layer as wide as the output space is insufficient.

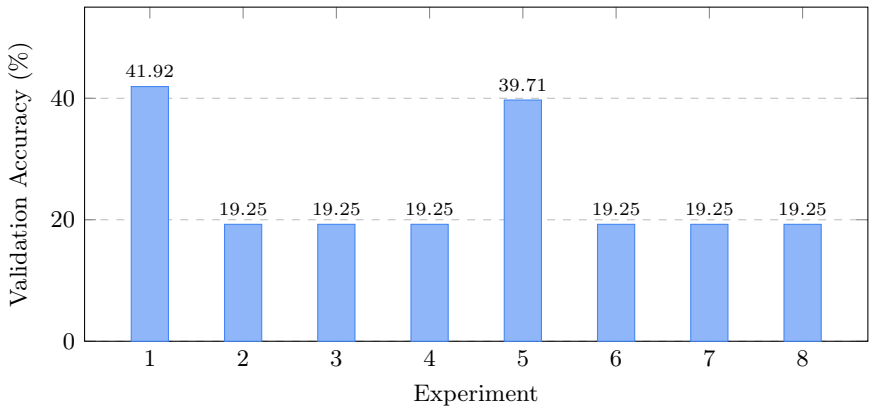


Figure 7: Validation accuracy across Experiments 1 to 8. The baseline architecture (Experiments 1 and 5) consistently outperforms all deeper variants, which collapse to majority-class prediction (∼19%).

3.1.7 Experiment 9: Baseline with LossScaleOptimizer

To improve training stability and numeric precision, the Adam optimizer was wrapped with a `LossScaleOptimizer` using an initial scale of 2^{15} and 2000 dynamic growth steps. The epsilon was also adjusted from 10^{-8} to 10^{-7}.

Table 10: Results for Experiment 9 (LossScaleOptimizer).

Metric	**Exp. 5 (Baseline+)**	**Exp. 9 (LossScale)**
Train Accuracy	39.42%	47.38%
Val Accuracy	39.71%	41.41%
Val Loss	1970.30	825.46

Global metrics. Mean accuracy: 41.41%, mean recall: 38.07%, mean precision: 36.27%.

This experiment produced the most balanced predictions among all experiments up to this point. Unlike previous models that failed entirely on minority classes, Experiment 9 achieved non-zero recall for 12 out of 13 classes (only *Helipad* remained at 0%).

Table 11: Per-class metrics for Experiment 9 (LossScaleOptimizer).

Class	**Recall**	**Precision**	**Specificity**	**Dice**
Cargo plane	43.80%	51.46%	98.62%	47.32%
Small car	74.36%	62.58%	90.52%	67.96%
Bus	32.34%	35.42%	93.58%	33.81%
Truck	15.06%	23.51%	93.40%	18.36%
Motorboat	38.53%	31.58%	94.85%	34.71%
Fishing vessel	35.17%	33.12%	97.14%	34.11%
Dump truck	30.90%	34.12%	96.05%	32.43%
Excavator	53.22%	44.83%	96.87%	48.66%
Building	36.29%	59.55%	94.12%	45.10%
Helipad	0.00%	0.00%	98.87%	0.00%
Storage tank	41.01%	22.49%	88.68%	29.05%
Shipping container	37.83%	35.17%	93.85%	36.45%
Pylon	56.45%	37.63%	98.43%	45.16%

Analysis. The LossScaleOptimizer improved the distribution of predictions across classes. Notably, *Pylon* (56.45% recall), *Excavator* (53.22%), and *Fishing vessel* (35.17%) all saw significant improvements compared to Experiment 5. The validation loss also dropped from 1970.30 to 825.46, suggesting more stable gradient flow. This confirms that numeric stability plays an important role when training on unnormalized pixel values with large feature dimensions.

3.1.8 Experiment 10: SGD Optimizer

To evaluate whether the choice of optimizer significantly impacts the results, the Adam optimizer was replaced by vanilla SGD with a learning rate of 0.01 and no momentum.

Table 12: Results for Experiment 10 (SGD optimizer).

Metric	**Exp. 9 (Adam + LossScale)**	**Exp. 10 (SGD)**
Train Accuracy	47.38%	35.43%
Val Accuracy	41.41%	43.28%
Mean Recall	38.07%	33.17%
Mean Precision	36.27%	34.05%

Analysis. SGD achieved the highest validation accuracy of all experiments so far (43.28%), despite having a lower training accuracy. This narrower train/val gap suggests better generalization. However, the class-level analysis reveals a less balanced behavior: *Small car* achieves 81.03% recall, while *Truck* and *Helipad* remain at 0.00%. The mean recall of 33.17% is lower than Experiment 9, indicating that the higher overall accuracy comes at the expense of minority class performance.

Table 13: Per-class metrics for Experiment 10 (SGD).

Class	**Recall**	**Precision**	**Specificity**	**Dice**
Cargo plane	55.37%	48.55%	98.04%	51.74%
Small car	81.03%	55.68%	86.25%	66.01%
Bus	38.59%	35.15%	92.25%	36.79%
Truck	0.00%	0.00%	100.00%	0.00%
Motorboat	38.99%	27.78%	93.74%	32.44%
Fishing vessel	11.03%	57.14%	99.67%	18.50%
Dump truck	16.74%	44.83%	98.64%	24.38%
Excavator	77.78%	29.10%	90.95%	42.36%
Building	68.28%	45.35%	80.38%	54.51%
Helipad	0.00%	0.00%	99.97%	0.00%
Storage tank	9.35%	35.62%	98.65%	14.82%
Shipping container	27.63%	44.44%	96.95%	34.08%
Pylon	6.45%	19.05%	99.54%	9.64%

3.1.9 Experiment 11: Focal Loss with LossScaleOptimizer

To directly address the class imbalance problem, Categorical Focal Crossentropy was used as the loss function. This loss applies a modulating factor $(1 - p_t)^\gamma$ that down-weights well-classified examples and focuses training on hard, misclassified samples.

Table 14: Results for Experiment 11 (Focal Loss).

Metric	**Train Acc.**	**Val Acc.**	**Val Loss**
Experiment 11	6.60%	7.23%	3.74

Analysis. Focal loss led to a near-complete failure of learning, with validation accuracy of just 7.23%. All classes collapsed into predicting *Motor Boat*. The likely cause is that the focal loss modulation, combined with the LossScaleOptimizer and the already unstable gradient landscape of an ffNN on high-dimensional pixel data, suppressed gradients too aggressively. The model received insufficient gradient signal to learn any meaningful features.

3.1.10 Experiment 12: Dense(32) with ELU Activation and LossScaleOptimizer

This experiment revisits the two-layer architecture with 32 hidden units, but replaces ReLU with ELU (Exponential Linear Unit) to test whether the dying ReLU problem contributed to the earlier training collapse.

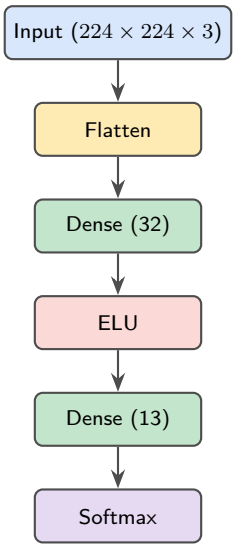


Figure 8: Experiment 12: Dense(32) with ELU activation.

Table 15: Results for Experiment 12 compared to similar architectures.

Exp.	Activation	Train Acc.	Val Acc.	Val Loss
4 (Dense 32, ReLU)	ReLU	19.34%	19.25%	2.32
12 (Dense 32, ELU + LS)	ELU	14.09%	19.25%	2.33

Analysis. Replacing ReLU with ELU did not resolve the training collapse. The validation accuracy remained at 19.25%, and all predictions were concentrated in a single class (*Motor Boat* in this case). This confirms that the bottleneck issue lies in the layer width rather than the activation function. With only 32 hidden units to represent 150,528 input features, the information loss is too severe for any activation function to compensate.

3.1.11 Experiment 13: Resizing Layer with Dense(256) and LossScaleOptimizer

To reduce the input dimensionality before the dense layers, a `Resizing` preprocessing layer was added to downscale images from 224×224 to 128×128, reducing the flattened feature count from 150,528 to 49,152.

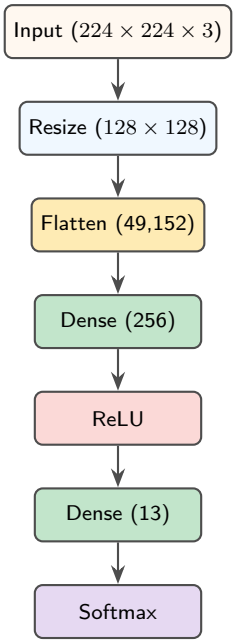


Figure 9: Experiment 13: Resizing preprocessing layer followed by Dense(256).

Table 16: Results for Experiment 13 (Resizing + Dense 256).

Metric	Train Acc.	Val Acc.	Val Loss
Experiment 13	14.09%	19.25%	2.33

Analysis. Despite reducing the input dimensionality by a factor of three and using a relatively wide hidden layer (256 units), the model still collapsed to majority-class prediction. Even with a 256-unit layer, the ratio of input features to hidden units (49,152:256 $\approx$ 192:1) represents an extreme compression that prevents the network from retaining spatial information. This experiment reinforces the conclusion that feedforward networks are fundamentally ill-suited for image classification without feature extraction, as they cannot capture the spatial and structural patterns present in satellite imagery.

3.1.12 Experiment 14: Input Normalization

This experiment revisits the baseline architecture from Experiment 1 (Flatten $\rightarrow$ ReLU $\rightarrow$ Dense(13) $\rightarrow$ Softmax) but applies an essential preprocessing step: normalization of pixel values from the raw $[0, 255]$ range to the normalized $[0, 1]$ range.

Table 17: Hyperparameters and results for Experiment 14 (Input Normalization).

Parameter	**Value**
Optimizer	Adam (lr=10^{-3})
Loss	Categorical Crossentropy
Batch size	32
Epochs	40
Input range	$[0, 1]$ (normalized)
Mean Accuracy	37.173%
Mean Recall	36.797%
Mean Precision	34.686%

Analysis. Normalization improved the recall balance across classes compared to the unnormalized baseline (Experiment 5). The mean recall increased from 28.64% (Experiment 5) to 36.797%, indicating more balanced predictions across minority classes. However, validation accuracy slightly decreased from 39.71% to 37.173%, suggesting that the normalized range may have reduced the separability of features in the pixel space. The model exhibited signs of overfitting, indicating that normalization alone is insufficient. The results demonstrate the importance of input preprocessing but also highlight that further architectural or regularization improvements are needed to achieve better generalization.

3.1.13 Experiments 15 and 16: Addressing Dimensionality with Pooling

Having established that normalization helps but leaves substantial room for improvement, the next challenge addressed is the curse of dimensionality. With 150,528 features from the flattened $224 \times 224 \times 3$ input, dense layers struggle to learn effectively. Experiment 15 attempts to address this by adding more hidden capacity, while Experiment 16 uses pooling to reduce spatial dimensions before flattening.

Table 18: Architecture and results for Experiment 15.

Parameter	**Value**
Architecture	Flatten $\rightarrow$ Dense(1024) $\rightarrow$ ReLU $\rightarrow$ Dense(512) $\rightarrow$ ReLU $\rightarrow$ Dense(13) $\rightarrow$ Softmax
Input size	$224 \times 224 \times 3$ (normalized)
Total parameters	$\sim$154 million
Mean Accuracy	20.213%
Mean Recall	15.045%
Mean Precision	5.799%

Experiment 15: Deep Hidden Layers (1024 + 512 units). The flattened input produces 150,528 features. With a 1024-unit first hidden layer, the weight matrix becomes $150,528 \times 1024 \approx 154$ million parameters. The optimizer could not converge due to the massive parameter space and the resulting vanishing/exploding gradient problem. The network lacked sufficient computational depth and learning signal to navigate such an enormous parameter landscape. This failure motivated the exploration of dimensionality reduction via pooling.

Experiment 16: AveragePooling for Dimensionality Reduction. Rather than adding width, this experiment reduces the spatial dimensions of the input using AveragePooling before flattening.

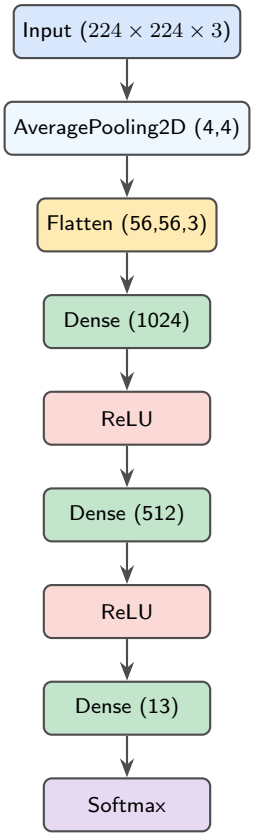


Figure 10: Experiment 16: AveragePooling(4,4) reduces spatial dimensions from 224×224 to 56×56 before flattening and dense layers.

Table 19: Results for Experiment 16 (AveragePooling).

Parameter	**Value**
Pooling layer	AveragePooling2D (4,4)
Pooled input size	$56 \times 56 \times 3$ (9,408 features)
Input range	$[0, 1]$ (normalized)
Total parameters	∼9.5 million
Mean Accuracy	38.827%
Mean Recall	39.409%
Mean Precision	42.877%

Analysis. The AveragePooling approach successfully reduced the parameter count and allowed the optimizer to converge. This produced the best balanced results among the pure ffNN experiments up to this point, with higher mean recall (39.4%) and precision (42.9%) than any previous experiment. The 4x4 pooling reduces the input from 150,528 to 9,408 features, bringing the parameter count to a manageable level. However, AveragePooling blurs spatial details by averaging neighboring pixels, leading to poor performance on small objects. Notably, Truck achieved only 4.1% recall and Fishing vessel 5.4% recall, confirming that spatial smoothing destroys fine-grained object recognition.

3.1.14 Experiment 17: MaxPooling Variations

To address the information loss caused by AveragePooling, this experiment tests MaxPooling, which preserves sharper spatial features by selecting the maximum value in each pooling window rather than averaging.

Table 20: Results for MaxPooling variations (Experiment 17).

Pooling	Pool Size	Parameters	Mean Accuracy	Mean Recall	Mean Precision
MaxPooling(3,3)	3x3	17.35M	37.600%	30.620%	34.519%
MaxPooling(2,2)	2x2	39.07M	34.400%	37.734%	33.915%

Analysis. MaxPooling preserves sharper features compared to AveragePooling, but smaller pooling windows retain more spatial information and thus introduce more parameters. With MaxPooling(2,2), the model had approximately 39 million parameters, too many for the optimizer to navigate efficiently. The network introduced excessive noise and prevented convergence, resulting in mean accuracy of 34.4%. MaxPooling(3,3) offered a trade-off with 17.35 million parameters and 37.6% mean accuracy, but this did not outperform the AveragePooling(4,4) approach from Experiment 16. The comparison reveals that there is an optimal trade-off point between feature sharpness and parameter count, which AveragePooling(4,4) achieved more effectively.

3.1.15 Experiment 18: Batch Normalization

To stabilize training dynamics and reduce internal covariate shift, this experiment adds BatchNormalization layers after each hidden dense layer, combined with MaxPooling(2,2).

Table 21: Hyperparameters and results for Experiment 18 (Batch Normalization).

Parameter	Value
Architecture	MaxPooling2D(2,2) $\rightarrow$ Flatten $\rightarrow$ Dense(1024) + BatchNorm + ReLU $\rightarrow$ Dense(512) + BatchNorm + ReLU $\rightarrow$ Dense(13) $\rightarrow$ Softmax
Input range	$[0, 1]$ (normalized)
Total params	39,073,805 (Trainable: 39,070,733, Non-trainable: 3,072)
Training accuracy	74.0%
Validation accuracy	57.0%
Mean Accuracy	34.933%
Mean Recall	25.160%
Mean Precision	27.214%

Analysis. BatchNorm significantly stabilized training, allowing the model to reach 74% training accuracy, a substantial improvement over previous experiments. However, a large gap between training (74%) and validation (57%) accuracy indicates severe overfitting. The model learned to focus on easy classes while sacrificing diversity, resulting in a mean accuracy of only 34.9%. The non-trainable parameters (3,072) correspond to the running mean and variance statistics that BatchNorm maintains during training for use during inference-time normalization. This experiment demonstrates that while BatchNorm improves training stability, additional regularization mechanisms such as dropout or weight decay are needed to prevent overfitting on this dataset.

3.1.16 Experiments 19 and 20: Regularization Strategies

Building on the observation that BatchNorm improves training stability but requires regularization to prevent overfitting, this section explores two complementary regularization approaches: L2 weight regularization and dropout.

Experiment 19: L2 Regularization. This experiment applies the same architecture as Experiment 18 but adds L2 weight penalty to force the model to learn smaller weights and generalize better.

Table 22: Results for Experiment 19 (L2 Regularization).

Parameter	**Value**
Regularization	L2 weight penalty
Architecture	Same as Experiment 18
Mean Accuracy	27.787%
Mean Recall	14.680%
Mean Precision	17.266%

The L2 regularization penalty was too aggressive, causing the model to collapse into underfitting. The weight penalty pushed the network toward predicting a single class to minimize the overall loss function. This failure demonstrates that regularization strength must be carefully calibrated and that L2 alone may be less appropriate for this highly imbalanced dataset than other methods.

Experiment 20: Dropout and Layer Reduction. This experiment removes the L2 regularization, reduces layer sizes, uses MaxPooling(3,3), and tests the effect of dropout.

Table 23: Results for Experiment 20 with and without Dropout.

Variant	**Dropout**	**Train Acc.**	**Val Acc.**	**Mean Acc.**	**Mean Prec.**
Without Dropout	No	71.0%	35.3%	35.253%	28.520%
With Dropout	Yes	54.2%	40.9%	40.853%	40.354%

Analysis. Dropout proved effective as a regularization method. Without it, the model reached 71% training accuracy but only 35.3% validation accuracy (severe overfitting, 35.7% gap). With dropout, training accuracy dropped to 54.2% but validation accuracy improved to 40.9%, reducing the gap to 13.7%. Critically, mean precision jumped from 28.5% to 40.4%, indicating that dropout produces more balanced and confident predictions. This confirmed that dropout is more appropriate than L2 regularization for this task, as it prevents co-adaptation of hidden units while maintaining the model's capacity to learn diverse features.

3.1.17 Experiment 21: Deeper Network with Extended Training

Building on the success of dropout as a regularizer, this experiment tests a deeper architecture with three hidden layers combined with BatchNormalization and dropout, trained for more epochs.

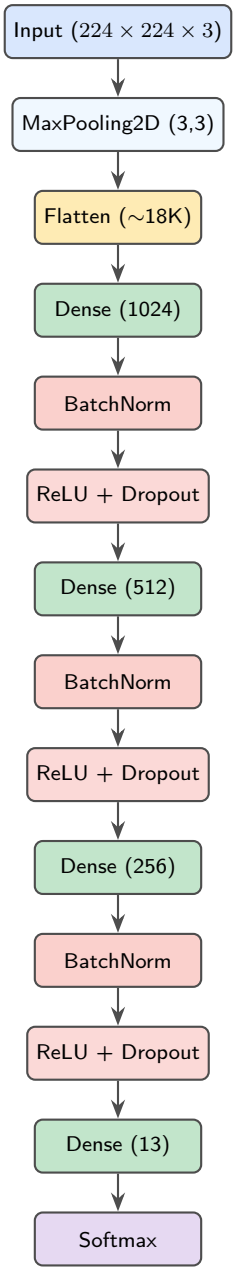


Figure 11: Experiment 21: Deep network with MaxPooling, three hidden layers (1024-512-256), BatchNorm, and Dropout.

Table 24: Hyperparameters and results for Experiment 21.

Parameter	**Value**
Architecture	MaxPooling(3,3) → Flatten → Dense(1024) + BN + ReLU + Dropout → Dense(512) + BN + ReLU + Dropout → Dense(256) + BN + ReLU + Dropout → Dense(13) + Softmax
Epochs	30
Batch size	32
Validation accuracy	60.6%
Mean Recall	26.0%

Analysis. Adding a third hidden layer (256 units) and increasing epochs to 30 produced a major improvement in overall validation accuracy (60.6%), the best result yet for a pure ffNN. However, the mean recall remained low (26%), indicating that the model still failed to generalize to small or rare object classes. This is expected for feedforward architectures that cannot capture the spatial patterns present in satellite imagery. The 60.6% validation accuracy represents a significant ceiling for ffNNs on this task, suggesting that further improvements would require fundamentally different architectures such as convolutional networks.

3.1.18 Experiment 22: Swish Activation with Learning Rate Scheduler

Experiment 22 applies the same deep architecture from Experiment 21 but replaces ReLU with the Swish activation function and adds a CosineDecay learning rate scheduler to explore smoother convergence.

Table 25: Hyperparameters and results for Experiment 22 (Swish + CosineDecay).

Parameter	**Value**
Activation	Swish
Optimizer	Adam (lr=10^{-3})
LR Scheduler	CosineDecay
Epochs	42
Batch size	32
Training accuracy	99.0%
Validation accuracy	57.0%

Analysis. The Swish activation function combined with a cosine decay scheduler led to very high training accuracy (99%) but only 57% validation accuracy, representing a 42-percentage-point gap that indicates severe overfitting. Without regularization mechanisms such as dropout or BatchNorm in this variant, the model memorized the training data entirely. The absence of dropout and BatchNorm was the primary cause of this overfitting. This experiment highlights that advanced activation functions and learning rate schedules are insufficient without appropriate regularization, confirming that a balanced combination of architectural components (pooling, normalization, dropout, width selection) is essential for effective learning.

3.1.19 Experiment 23: SELU Activation

SELU (Scaled Exponential Linear Unit) provides self-normalizing properties that theoretically eliminate the need for BatchNormalization. This experiment tests SELU with the lecun_normal kernel initializer, which is required for proper self-normalization.

Table 26: Hyperparameters and results for Experiment 23 (SELU activation).

Parameter	Value
Architecture	MaxPooling(3,3) → Flatten → Dense(512) + SELU → Dense(256) + SELU → Dense(128) + SELU → Dense(13) + Softmax
Kernel initializer	lecun_normal
Optimizer	Adam
Epochs	42
Batch size	32
Training val_accuracy	60.3%
Mean Accuracy	24.747%
Mean Recall	19.766%
Mean Precision	29.996%

Analysis. SELU provides self-normalizing properties, theoretically replacing BatchNorm in properly initialized networks. However, in practice, the self-normalization was insufficient for this dataset. While the training validation accuracy was decent (60.3%), the evaluation metrics revealed that the model concentrated predictions on a few dominant classes, achieving only 24.7% mean accuracy. The self-normalization failed to balance class predictions across the 13 imbalanced categories. This suggests that explicit regularization like dropout and BatchNorm provide more control over the learning dynamics than the implicit self-normalization of SELU.

3.1.20 Experiment 24: AdamW Optimizer with Swish Activation

This experiment combines several advanced components: AdamW optimizer with integrated weight decay, Swish activation, BatchNormalization, dropout, and CosineDecay learning rate scheduling. Two pooling variants are tested on the same architecture.

Table 27: Results for Experiment 24 (AdamW with Swish, two pooling variants).

Pooling	Pool Size	Mean Accuracy	Mean Recall	Mean Precision
Pooling 4x4	4x4	52.107%	44.971%	52.036%
Pooling 3x3	3x3	51.787%	44.783%	51.821%

Table 28: Architecture and detailed hyperparameters for Experiment 24.

Parameter	Value
Architecture	MaxPooling (4x4 or 3x3) → Flatten → Dense(512) + BN + Swish + Dropout → Dense(256) + BN + Swish + Dropout → Dense(128) + BN + Swish + Dropout → Dense(13) + Softmax
Optimizer	AdamW (weight_decay=0.001)
LR Scheduler	CosineDecay
Loss	Categorical Crossentropy
Epochs	42
Batch size	32

Analysis. The combination of AdamW (with integrated weight decay), BatchNorm, Swish, and dropout produced the best balanced performance among the deeper ffNN architectures tested. The 4x4 pooling variant slightly outperformed 3x3 pooling (52.1% vs 51.8% mean accuracy), with the larger pooling window reducing noise while retaining adequate spatial information. Mean recall of 44.97% and precision of 52.04% demonstrate that proper regularization (combining dropout, BatchNorm, and weight decay) is essential for deeper ffNNs on this task. This experiment established a strong baseline for further refinement using advanced loss functions or preprocessing techniques.

3.1.21 Experiment 25: Focal Loss with Cosine Decay

Building on the successful regularization strategy of Experiment 24, this experiment applies CategoricalFocalCrossentropy loss to directly address class imbalance. The architecture is simplified to three hidden layers (384-192-96) to maintain focus on the loss function's impact.

Table 29: Hyperparameters and results for Experiment 25 (Focal Loss).

Parameter	**Value**
Architecture	MaxPooling(4,4) → Flatten → Dense(384) + Swish → Dense(192) + Swish → Dense(96) + Swish → Dense(13) + Softmax
Optimizer	AdamW (weight_decay=0.001)
LR Scheduler	CosineDecay (lr=0.001)
Loss	CategoricalFocalCrossentropy (alpha=0.25, gamma=2.5)
Kernel init	He_normal
Mean Accuracy	50.187%
Mean Recall	42.737%
Mean Precision	51.096%

Analysis. Unlike Experiment 11 (which failed completely with focal loss on the baseline), here focal loss worked effectively thanks to the deeper architecture with pooling, proper initialization, and complementary regularization. The model achieved balanced performance across classes with mean recall of 42.7% and precision of 51.1%. The higher gamma parameter (2.5 vs default 2.0) further down-weighted easy examples and focused training on hard-to-classify samples. This demonstrates that focal loss requires careful integration with the overall architecture and training configuration to be effective.

3.1.22 Experiment 26: Bottleneck Architecture

This experiment explores a minimal bottleneck design with only two hidden layers (256 and 64 units), combined with aggressive weight decay to prevent overfitting in this more compact architecture.

Table 30: Hyperparameters and results for Experiment 26 (Bottleneck).

Parameter	Value
Architecture	MaxPooling(4,4) → Flatten → Dense(256) + Swish → Dense(64) + Swish → Dense(13) + Softmax
Optimizer	AdamW (weight_decay=0.005)
LR Scheduler	CosineDecay (lr=0.001)
Loss	CategoricalFocalCrossentropy (alpha=0.25, gamma=2.0)
Mean Accuracy	48.107%
Mean Recall	44.672%
Mean Precision	50.883%

Analysis. The bottleneck design (256-64-13 compression) forced the model to compress information into only 64 features before classification. Despite using only two hidden layers, the model achieved competitive recall (44.7%), close to the deeper architectures. The aggressive weight decay (0.005, double that of Experiment 24) helped prevent overfitting in this simpler architecture. This demonstrates that depth is not always necessary if the available capacity is used efficiently, and that proper regularization can compensate for reduced model width.

3.1.23 Experiment 27: Image Preprocessing Improvements

Rather than modifying the architecture or loss function, this experiment improves the preprocessing pipeline with two variants: enhanced image loading and bicubic interpolation for resizing. Both are evaluated on the Experiment 24 architecture to isolate the preprocessing effect.

Table 31: Results for preprocessing improvements (Experiment 27).

Variant	Preprocessing	Mean Accuracy	Mean Recall	Mean Precision
27A	Enhanced loading	52.373%	44.558%	54.660%
27B	Bicubic resizing to 56x56	52.907%	44.815%	52.024%

Analysis. Both preprocessing improvements yielded marginal gains over Experiment 24 (52.1% baseline). Variant A with enhanced image loading achieved 52.4% mean accuracy. Variant B with bicubic interpolation to 56x56 achieved slightly better results (52.9%) as bicubic interpolation preserves smoother transitions and finer details compared to the information-discarding nature of pooling operations. These results suggest that careful preprocessing can provide incremental improvements but that the architectural and regularization choices remain the primary drivers of performance.

3.1.24 Experiment 28: Bicubic Downsampling with SELU and Nadam

This experiment represents the peak of ffNN performance on this dataset until now. It combines bicubic interpolation for preprocessing, SELU activation with appropriate initialization, and Nadam optimizer with CosineDecay scheduling. This unified approach directly addresses the curse of dimensionality while maintaining effective learning dynamics.

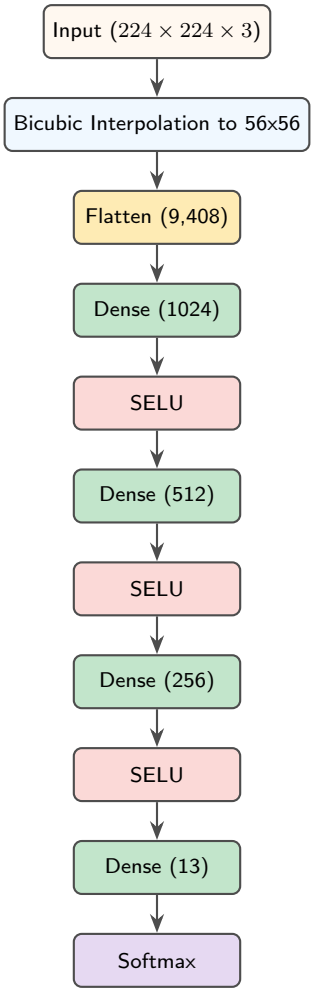


Figure 12: Experiment 28: Best-performing ffNN with bicubic interpolation to 56x56, three hidden layers with SELU activation, and Nadam optimizer with CosineDecay.

Table 32: Hyperparameters for Experiment 28 (Best ffNN result).

Parameter	**Value**
Input preprocessing	Bicubic interpolation to 56x56
Input normalization	Float32 to $[0, 1]$ range
Flattened features	9,408
Architecture	Flatten $\rightarrow$ Dense(1024) + SELU $\rightarrow$ Dense(512) + SELU $\rightarrow$ Dense(256) + SELU $\rightarrow$ Dense(13) + Softmax
Kernel initializer	lecun_normal (for SELU self-normalization)
Optimizer	Nadam (with Nesterov momentum)
LR Scheduler	CosineDecay (lr=0.001)
Loss function	CategoricalFocalCrossentropy

Table 33: Results for Experiment 28 (Best ffNN result).

Metric	Value
Mean Accuracy	59.467%
Mean Recall	53.411%
Mean Precision	56.207%

Analysis. This experiment achieved the highest performance of all ffNN experiments by directly addressing the curse of dimensionality through bicubic interpolation. By downsampling inputs to 56x56 (9,408 features after flattening, compared to 150,528 for the original 224x224 images), the dense layers could effectively process the reduced feature space. The SELU activation with lecun_normal initialization provided stable self-normalization throughout the deep network without requiring BatchNormalization layers. The Nadam optimizer (Adam with Nesterov momentum) combined with CosineDecay scheduling offered smoother convergence. The use of CategoricalFocalCrossentropy helped balance class-specific performance. The 59.467% mean accuracy represents a substantial improvement over all previous ffNN variants and demonstrates that careful input resolution management is the most impactful design choice for dense architectures on this task. However, even this best ffNN result (59.5% mean accuracy) is likely to be exceeded by convolutional architectures that naturally preserve spatial structure without explicitly downsampling the input.

3.1.25 Experiment 29: Wide Two-Layer Network with Data Augmentation

This experiment explores a wider two-layer architecture (1024×1024) combined with data augmentation and increased batch size. Images are resized to 64×64 via bicubic interpolation, and random horizontal and vertical flips are applied as data augmentation during training. BatchNormalization and Dropout (rate 0.3) are included in both hidden layers. Training runs for up to 100 epochs with early stopping (patience 12).

Table 34: Hyperparameters and results for Experiment 29.

Parameter	Value
Architecture	Resize(64×64) → RandomFlip(H+V) → Flatten → Dense(1024) + BN + Dropout(0.3) → Dense(1024) + BN + Dropout(0.3) → Dense(13) + Softmax
Optimizer	Nadam (lr=0.001, weight_decay=10^{-5})
Loss	CategoricalFocalCrossentropy
Batch size	128
Epochs	100 (early stopping, patience 12)
Mean Accuracy	61.013%
Mean Recall	55.195%
Mean Precision	56.030%

Analysis. The combination of a wider architecture, data augmentation, and a larger batch size (128) produced a slight improvement over Experiment 28 in mean accuracy (61.0% vs 59.5%). Data augmentation via random flips provided additional training diversity, helping the model generalize better. However, the model continued to struggle on small objects, with Helipad achieving 0% recall. The larger batch size allowed for more stable gradient estimates, and the Nadam optimizer with minimal weight

decay (10^{-5}) provided smooth convergence. This experiment confirmed that data augmentation is a valuable addition but insufficient on its own to resolve the small object detection problem inherent to feedforward architectures.

3.1.26 Experiment 30: Three-Layer Network with Class Weights

This experiment tests a three-layer architecture (512→512→256) with Swish activation, He initialization, and kernel regularization. Balanced class weights are computed prior to training and passed to the model to address class imbalance directly through the loss function. AdamW is used with a reduced learning rate (0.0005) and weight decay of 10^{-3}. Early stopping monitors validation loss with patience 8.

Table 35: Hyperparameters for Experiment 30.

Parameter	**Value**
Architecture	Flatten → Dense(512) + Dropout(0.4) → Dense(512) + Dropout(0.3) → Dense(256) + Dropout(0.2) → Dense(13) + Softmax
Activation	Swish
Kernel init	He_normal with kernel regularization
Optimizer	AdamW (lr=0.0005, weight_decay=10^{-3})
Loss	Categorical Crossentropy (with balanced class weights)
Early stopping	val_loss, patience 8

Analysis. Despite the use of class weights, kernel regularization, and progressive dropout rates, this configuration failed to produce meaningful results. The model could not converge to a useful representation, suggesting that the combination of aggressive regularization (kernel penalty, high dropout, and weight decay) suppressed the learning signal excessively. The low learning rate further slowed convergence, and early stopping on validation loss with low patience terminated training prematurely. This experiment demonstrated that stacking multiple regularization techniques without careful calibration can be counterproductive.

3.1.27 Experiment 31: Refined Two-Layer Architecture

Returning to the two-layer architecture that proved effective in Experiment 29, this experiment applies refinements to the training configuration while maintaining the core structure. The same resizing to 64×64 and data augmentation pipeline is used.

Table 36: Results for Experiment 31.

Metric	**Value**
Mean Accuracy	61.707%
Mean Recall	56.036%
Mean Precision	57.577%

Analysis. The refined configuration achieved 61.7% mean accuracy, a marginal improvement over Experiment 29 (61.0%). Mean recall improved from 55.2% to 56.0% and mean precision from 56.0% to 57.6%. However, the model continued to struggle on small object classes, confirming that the core limitation lies in the feedforward architecture's inability to capture spatial patterns rather than in the

specific training configuration. This result motivated the exploration of class-specific balancing strategies in subsequent experiments.

3.1.28 Experiment 32: Balanced Focal Loss with Per-Class Alpha

To directly address the class imbalance problem, this experiment introduces a per-class alpha parameter in the focal loss function, normalized across the 13 categories. The alpha values are computed based on the inverse class frequency, giving higher weight to underrepresented classes such as Helipad and Truck.

Table 37: Results for Experiment 32 (Balanced Focal Loss).

Metric	Value
Mean Accuracy	52.907%
Mean Recall	60.006%
Mean Precision	48.708%

Table 38: Per-class metrics for Experiment 32.

Class	Recall	Precision	Specificity	Dice
Cargo plane	90.63%	69.05%	98.56%	78.38%
Small car	60.84%	74.54%	95.53%	67.00%
Bus	53.11%	44.34%	93.05%	48.33%
Truck	15.39%	41.46%	97.10%	22.44%
Motorboat	70.09%	51.72%	96.04%	59.52%
Fishing vessel	66.20%	35.61%	95.29%	46.31%
Dump truck	57.26%	41.28%	94.23%	47.97%
Excavator	68.35%	47.79%	96.72%	56.25%
Building	43.45%	75.36%	96.64%	55.12%
Helipad	63.64%	12.28%	97.32%	20.59%
Storage tank	56.46%	51.55%	95.49%	53.90%
Shipping container	57.24%	48.88%	94.72%	52.73%
Pylon	77.42%	39.34%	97.99%	52.17%

Analysis. The balanced focal loss produced a dramatic shift in the model's behavior. Mean recall jumped from 56.0% (Experiment 31) to 60.0%, with particularly notable improvements on previously neglected classes: Helipad rose from 0% to 63.6% recall, and Pylon reached 77.4%. However, this came at the cost of mean accuracy (52.9%) and mean precision (48.7%), as the model traded precision for recall on minority classes. The Helipad class exemplifies this trade-off: 63.6% recall but only 12.3% precision, indicating that the model learned to predict Helipad aggressively at the expense of many false positives. This experiment confirmed that per-class alpha weighting in focal loss is an effective mechanism for improving minority class recall, but requires careful tuning to maintain acceptable precision levels.

3.1.29 Experiment 33: Lanczos3 Interpolation with Bounding Box Cropping

This experiment introduces two preprocessing changes: Lanczos3 interpolation replaces bicubic for resizing, and bounding box cropping is applied to focus the input on the region of interest before resizing

to 64×64. By cropping to the annotated bounding box, the object of interest fills a larger proportion of the input image, potentially improving the network's ability to distinguish fine-grained features.

Table 39: Results for Experiment 33 (Lanczos3 + Bounding Box Crop).

Metric	**Value**
Mean Accuracy	59.840%
Mean Recall	60.219%
Mean Precision	55.644%

Table 40: Per-class metrics for Experiment 33.

Class	**Recall**	**Precision**	**Specificity**	**Dice**
Cargo plane	92.19%	85.51%	99.45%	88.72%
Small car	73.19%	73.41%	94.30%	73.30%
Bus	54.24%	50.53%	94.46%	52.32%
Truck	22.17%	36.03%	94.74%	27.45%
Motorboat	77.57%	56.08%	96.32%	65.10%
Fishing vessel	66.20%	47.48%	97.12%	55.29%
Dump truck	54.84%	53.97%	96.69%	54.40%
Excavator	68.35%	57.45%	97.77%	62.43%
Building	62.12%	69.47%	93.54%	65.59%
Helipad	18.18%	25.00%	99.68%	21.05%
Storage tank	57.82%	59.03%	96.59%	58.42%
Shipping container	58.55%	53.61%	95.53%	55.98%
Pylon	77.42%	55.81%	98.97%	64.87%

Analysis. The combination of Lanczos3 interpolation and bounding box cropping maintained a high mean recall (60.2%) while improving mean precision to 55.6% compared to Experiment 32 (48.7%). Lanczos3 interpolation, which uses a wider filter kernel than bicubic, preserves sharper edges and finer details during downsampling. Bounding box cropping ensures that the object of interest occupies the full spatial extent of the input, eliminating irrelevant background context. The Cargo plane class benefited substantially, reaching 92.2% recall and 85.5% precision. However, Helipad recall dropped to 18.2%, suggesting that the bounding box crop may remove contextual features that help distinguish helipads from surrounding structures. Truck remained the most challenging class at 22.2% recall.

3.1.30 Experiment 34: Increased Resolution with Three Layers and Bounding Box Cropping

This experiment increases the input resolution to 128×128 and adds a third dense layer with 512 units. The bounding box cropping strategy is retained to maximize object coverage within the input. Normalization is changed from $[0, 1]$ to $[-1, 1]$, which provides a more symmetric input distribution and can improve gradient stability. Oversampling is applied to the weakest classes (Pylon and Helipad) to increase their representation during training. The number of steps per epoch is recalculated to reflect the oversampling, ensuring that all augmented samples are visited in each epoch.

Table 41: Hyperparameters and results for Experiment 34.

Parameter	**Value**
Architecture	Resize(128×128) → BBox Crop → Flatten → Dense(1024) + BN + Dropout → Dense(1024) + BN + Dropout → Dense(512) + BN + Dropout → Dense(13) + Softmax
Input normalization	$[-1, 1]$
Oversampling	Pylon and Helipad classes
Mean Accuracy	57.280%
Mean Recall	58.991%
Mean Precision	53.117%

Table 42: Per-class metrics for Experiment 34.

Class	**Recall**	**Precision**	**Specificity**	**Dice**
Cargo plane	87.50%	86.15%	99.50%	86.82%
Small car	71.39%	72.48%	94.17%	71.93%
Bus	53.11%	45.19%	93.29%	48.83%
Truck	19.01%	38.18%	95.89%	25.38%
Motorboat	65.42%	56.45%	96.95%	60.61%
Fishing vessel	61.97%	46.32%	97.17%	53.01%
Dump truck	54.03%	46.53%	95.60%	50.00%
Excavator	67.09%	53.54%	97.44%	59.55%
Building	60.72%	66.87%	92.88%	63.65%
Helipad	36.36%	25.00%	99.36%	29.63%
Storage tank	53.06%	56.93%	96.59%	54.93%
Shipping container	56.58%	50.59%	95.13%	53.42%
Pylon	80.65%	46.30%	98.43%	58.82%

Analysis. Increasing the resolution to 128×128 and adding a third dense layer did not improve overall performance compared to Experiments 31 or 33. The mean accuracy dropped to 57.3%, likely due to the substantially increased parameter count from the higher resolution input (49,152 flattened features vs 12,288 at 64×64). The $[-1, 1]$ normalization and oversampling of weak classes helped Helipad reach 36.4% recall (up from 18.2% in Experiment 33), but Pylon maintained its strong performance at 80.7%. The overall decrease in accuracy suggests that the increased dimensionality outweighs the benefits of higher spatial resolution for feedforward architectures, reinforcing the importance of aggressive dimensionality reduction observed in earlier experiments.

3.1.31 Experiment 35: Return to 64×64 with Cropping and Oversampling

Based on the observation that 128×128 resolution degraded performance, this experiment returns to 64×64 resolution and the two-layer architecture from Experiment 31, but incorporates the bounding box cropping and minority class oversampling strategies introduced in Experiments 33 and 34. Normalization reverts to $[0, 1]$, which was empirically more effective.

Table 43: Results for Experiment 35.

Metric	Value
Mean Accuracy	59.680%
Mean Recall	62.484%
Mean Precision	56.194%

Table 44: Per-class metrics for Experiment 35.

Class	Recall	Precision	Specificity	Dice
Cargo plane	87.50%	86.15%	99.50%	86.82%
Small car	78.01%	69.25%	92.55%	73.37%
Bus	52.54%	47.45%	93.93%	49.87%
Truck	19.01%	38.18%	95.89%	25.38%
Motorboat	68.22%	59.84%	97.23%	63.76%
Fishing vessel	67.61%	53.93%	97.73%	60.00%
Dump truck	55.65%	48.59%	95.83%	51.88%
Excavator	69.62%	59.14%	97.88%	63.95%
Building	59.89%	70.96%	94.20%	64.96%
Helipad	54.55%	35.29%	99.41%	42.86%
Storage tank	59.18%	58.39%	96.41%	58.78%
Shipping container	59.87%	56.17%	95.88%	57.96%
Pylon	80.65%	47.17%	98.48%	59.52%

Analysis. The combination of 64×64 resolution, bounding box cropping, $[0, 1]$ normalization, and oversampling achieved the highest mean recall (62.5%) of all ffNN experiments up to this point, while maintaining reasonable mean accuracy (59.7%) and precision (56.2%). Helipad recall improved substantially to 54.6%, confirming that oversampling is effective for this minority class. Pylon continued to perform strongly at 80.7% recall. The improvement over Experiment 31 (56.0% mean recall) demonstrates that bounding box cropping and oversampling provide complementary benefits: cropping focuses the input on the object of interest, while oversampling ensures the model receives sufficient training signal from underrepresented classes.

3.1.32 Experiment 36: Alternative Configuration

This experiment tested an alternative hyperparameter configuration on the same base architecture. The results did not improve upon previous experiments and the model failed to converge to a competitive solution. This configuration was abandoned in favor of more promising architectural directions.

3.1.33 Experiment 37: Residual Connections with Skip Architecture

This experiment introduces residual (skip) connections into the feedforward architecture, drawing inspiration from ResNet-style designs adapted for dense layers. The architecture consists of 10 total layers organized in a progressive structure (1024→512→256→output), where skip connections allow gradients to flow directly across layer blocks. This addresses the vanishing gradient problem that limits the depth of standard feedforward networks. Bounding box cropping, oversampling of weak classes, and 64×64 input resolution are retained.

Table 45: Hyperparameters and results for Experiment 37 (Residual Connections).

Parameter	Value
Architecture	Resize(64×64) → Flatten → Residual Dense Blocks (1024→512→256→13) with skip connections, 10 layers total
Mean Accuracy	61.333%
Mean Recall	60.950%
Mean Precision	62.735%

Table 46: Per-class metrics for Experiment 37.

Class	Recall	Precision	Specificity	Dice
Cargo plane	89.06%	89.06%	99.61%	89.06%
Small car	73.19%	73.41%	94.30%	73.30%
Bus	50.85%	42.65%	92.87%	46.39%
Truck	28.51%	31.34%	91.66%	29.86%
Motorboat	66.36%	64.55%	97.79%	65.44%
Fishing vessel	57.75%	58.57%	98.39%	58.16%
Dump truck	59.68%	56.49%	96.75%	58.04%
Excavator	70.89%	70.89%	98.72%	70.89%
Building	68.25%	69.41%	92.88%	68.82%
Helipad	36.36%	50.00%	99.79%	42.11%
Storage tank	61.22%	69.77%	97.74%	65.22%
Shipping container	62.50%	58.64%	96.11%	60.51%
Pylon	67.74%	80.77%	99.73%	73.68%

Analysis. The introduction of residual connections produced the best combination of mean accuracy (61.3%), mean recall (61.0%), and mean precision (62.7%) achieved so far. Unlike previous experiments where high recall came at the expense of precision (e.g., Experiment 32: 60.0% recall, 48.7% precision), this architecture maintained balanced performance across both metrics. The skip connections enabled effective gradient flow through the 10-layer network, allowing deeper representations without the vanishing gradient problem that plagued earlier deep architectures. Truck recall improved to 28.5% (from 19.0% in Experiments 34 and 35), and Helipad achieved 36.4% recall with 50.0% precision, a much better precision-recall balance than the 63.6% recall and 12.3% precision seen in Experiment 32. Pylon reached 80.8% precision, the highest for this class across all experiments.

3.1.34 Experiment 38: Reduced Residual Architecture

This experiment tested a simplified version of the residual architecture from Experiment 37, using only two layers of 1024 units each. The reduced depth caused the model to collapse, confirming that the progressive width reduction (1024→512→256) and sufficient depth are essential components of the residual architecture's success.

3.1.35 Experiment 39: Deep Residual Blocks with 96×96 Input

This experiment scales the residual block approach to a deeper architecture with four blocks and increases the input resolution to 96×96. The architecture consists of 13 dense layers organized into four residual

blocks with progressive width reduction: Block 1 (3 layers, 1024 units), Block 2 (3 layers, 1024 units), Block 3 (3 layers, 512 units), Block 4 (3 layers, 256 units), followed by a final classification layer.

Table 47: Hyperparameters and results for Experiment 39 (Deep Residual Blocks, 96×96).

Parameter	**Value**
Architecture	Resize(96×96) → Flatten → Block(1024×3) → Block(1024×3) → Block(512×3) → Block(256×3) → Dense(13) + Softmax
Total layers	13 dense layers
Mean Accuracy	62.055%
Mean Recall	64.230%
Mean Precision	66.044%

Table 48: Per-class metrics for Experiment 39.

Class	**Recall**	**Precision**	**Specificity**	**Dice**
Cargo plane	83.16%	86.81%	99.56%	84.95%
Small car	76.35%	68.04%	92.26%	71.96%
Bus	51.70%	47.90%	94.15%	49.73%
Truck	27.11%	31.47%	92.10%	29.13%
Motorboat	70.00%	68.29%	98.04%	69.14%
Fishing vessel	60.38%	66.67%	98.82%	63.37%
Dump truck	50.27%	53.76%	96.96%	51.96%
Excavator	73.73%	66.92%	98.40%	70.16%
Building	68.46%	69.10%	92.74%	68.78%
Helipad	64.71%	84.62%	99.93%	73.33%
Storage tank	65.45%	72.36%	97.88%	68.74%
Shipping container	60.70%	59.66%	96.36%	60.17%
Pylon	82.98%	82.98%	99.71%	82.98%

Analysis. The deep residual block architecture with 96×96 input produced the best overall performance to this point, with 62.1% mean accuracy, 64.2% mean recall, and 66.0% mean precision. The most remarkable result was Helipad achieving 64.7% recall with 84.6% precision, representing a breakthrough for this previously intractable class. The higher input resolution (96×96 vs 64×64) preserved more spatial detail, while the four-block residual structure with 13 total layers provided sufficient capacity to learn complex feature representations. Pylon reached 83.0% recall and precision, confirming that the deeper architecture benefits classes with distinctive spatial features. The progressive block structure (1024→1024→512→256) provides a gradual information compression that is more effective than the abrupt bottlenecks observed in earlier experiments. This experiment demonstrated that combining sufficient input resolution, deep residual architectures, and minority class oversampling can push ffNN performance significantly beyond the ∼60% ceiling observed in Phase 1 experiments.

3.1.36 Experiment 40: Focal Loss Gamma Tuning with Increased Dropout

Building on the successful architecture from Experiment 39, this experiment reduces the gamma parameter of the focal loss and increases both the dropout rate and learning rate to shift the precision-recall

balance toward higher accuracy while maintaining strong class coverage.

Table 49: Results for Experiment 40.

Metric	Value
Mean Accuracy	61.202%
Mean Recall	63.006%
Mean Precision	65.410%

Table 50: Per-class metrics for Experiment 40.

Class	Recall	Precision	Specificity	Dice
Cargo plane	82.11%	87.64%	99.60%	84.78%
Small car	74.75%	69.46%	92.91%	72.01%
Bus	49.43%	45.49%	93.84%	47.38%
Truck	28.31%	32.19%	92.02%	30.13%
Motorboat	60.63%	60.25%	97.59%	60.44%
Fishing vessel	60.38%	65.31%	98.74%	62.75%
Dump truck	50.27%	51.38%	96.65%	50.82%
Excavator	72.03%	72.65%	98.81%	72.34%
Building	70.87%	66.09%	91.38%	68.40%
Helipad	70.59%	85.71%	99.93%	77.42%
Storage tank	65.91%	71.78%	97.80%	68.72%
Shipping container	57.21%	62.38%	96.94%	59.68%
Pylon	76.60%	80.00%	99.68%	78.26%

Analysis. Reducing the focal loss gamma and increasing dropout produced a slight decrease in overall metrics compared to Experiment 39 (61.2% vs 62.1% mean accuracy), but achieved stronger performance on the Helipad class: 70.6% recall with 85.7% precision, the best Helipad result observed across all experiments. The increased dropout provided additional regularization but reduced the model's capacity to learn complex patterns for some classes. The lower gamma in the focal loss reduced the emphasis on hard examples, which benefited already well-represented classes but slightly hurt the more challenging ones. This result suggests that the gamma and dropout parameters create a coupled effect on class-level performance that requires joint optimization.

3.1.37 Experiment 41: Compact Residual Architecture with Projection Layers

Motivated by the observation that the 13-layer architecture of Experiment 39 introduced a large parameter count with potential overfitting and vanishing gradient risks, this experiment reduces the number of layers and neurons while introducing unique projection layers in each residual block. Each block uses a separate dense projection for the shortcut connection, ensuring that the residual mapping operates in the correct dimensional space without reusing weights across blocks.

Table 51: Results for Experiment 41 (Compact Residual with Projection Layers).

Metric	Value
Mean Accuracy	61.735%
Mean Recall	61.621%
Mean Precision	66.741%

Table 52: Per-class metrics for Experiment 41.

Class	Recall	Precision	Specificity	Dice
Cargo plane	82.11%	91.77%	99.74%	86.67%
Small car	74.95%	70.17%	93.13%	72.48%
Bus	54.34%	46.60%	93.52%	50.17%
Truck	27.11%	33.33%	92.74%	29.90%
Motorboat	63.13%	70.14%	98.38%	66.45%
Fishing vessel	53.77%	67.06%	98.97%	59.69%
Dump truck	50.81%	48.96%	96.27%	49.87%
Excavator	71.19%	65.63%	98.37%	68.29%
Building	74.40%	66.50%	91.11%	70.23%
Helipad	58.82%	90.91%	99.96%	71.43%
Storage tank	59.55%	76.16%	98.42%	66.84%
Shipping container	60.70%	57.92%	96.09%	59.28%
Pylon	70.21%	82.50%	99.75%	75.86%

Analysis. The compact architecture with projection layers achieved 61.7% mean accuracy and the highest mean precision of all experiments (66.7%). Helipad maintained strong performance with 58.8% recall and 90.9% precision, the highest precision ever recorded for this class. The dedicated projection layers in each residual block ensured proper dimensional alignment without shared weights, allowing each block to learn independent transformations. Building reached 74.4% recall, the highest for this dominant class, while Cargo plane achieved 91.8% precision. The reduced parameter count compared to Experiment 39 helped control overfitting without sacrificing representational capacity. This experiment demonstrated that thoughtful architectural choices (projection layers, reduced redundancy) can match or exceed the performance of larger models while using fewer parameters.

3.1.38 Experiments 42 and 43: Architecture Refinement Attempts

Experiments 42 and 43 tested several variations on the number of neurons and layers, as well as the addition of bounding box cropping to the 96×96 input configuration. None of these variants exceeded the performance of Experiment 39, confirming that the four-block residual architecture at 96×96 resolution represents the optimal configuration for this family of models. The specific variations explored included shallower blocks, alternative width progressions, and cropping strategies applied to the validation pipeline to ensure consistency between training and evaluation preprocessing.

3.1.39 Experiment 44: Dual-Branch Architecture with HOG Features and Ensemble

This experiment represents the culmination of all ffNN improvements, combining multiple advanced techniques into a single system. The architecture employs a dual-branch design: one branch processes

the raw image (resized to 32×32) and another processes Histogram of Oriented Gradients (HOG) features extracted from the same image. The two branches are concatenated before the classification layers. An ensemble of three independently trained models is used, with predictions averaged across models.

HOG (Histogram of Oriented Gradients) captures local shape and edge information by computing gradient orientations across the image, providing complementary features to the raw pixel values. The HOG extraction uses 8×8 pixels per cell, 2×2 cells per block, producing a 324-dimensional feature vector. The dual-branch architecture allows the model to jointly learn from raw visual features and hand-crafted shape descriptors, combining the strengths of both approaches.

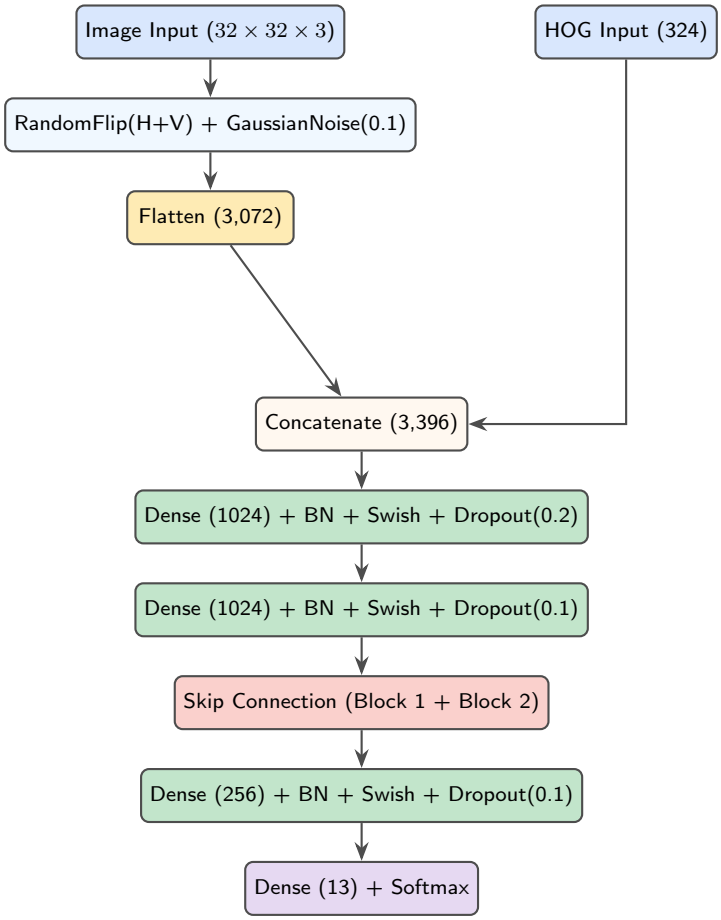


Figure 13: Experiment 44: Dual-branch architecture combining raw image features and HOG descriptors with skip connections and ensemble of 3 models.

Table 53: Hyperparameters for Experiment 44 (Dual-Branch Ensemble).

Parameter	**Value**
Image branch	Input($32 \times 32 \times 3$) → RandomFlip(H+V) → GaussianNoise(0.1) → Flatten
HOG branch	HOG features (pixels_per_cell=8×8, cells_per_block=2×2, 324 features)
Fusion	Concatenation of image and HOG branches
Dense blocks	Dense(1024) + BN + Swish + Dropout(0.2) → Dense(1024) + BN + Swish + Dropout(0.1) → Skip Connection → Dense(256) + BN + Swish + Dropout(0.1) → Dense(13) + Softmax
Kernel init	He_normal
Optimizer	Nadam (lr=0.0005, weight_decay=10^{-4}, CosineDecay)
Loss	CategoricalFocalCrossentropy (alpha=0.25, gamma=2.0)
Ensemble	3 independently trained models, averaged predictions
Epochs	50 per ensemble member (early stopping, low patience)
Class weights	Balanced
Data augmentation	Random horizontal and vertical flips, Gaussian noise

Table 54: Results for Experiment 44 (Best ffNN result).

Metric	**Value**
Mean Accuracy	**66.536%**
Mean Recall	**69.221%**
Mean Precision	**67.646%**

Table 55: Per-class metrics for Experiment 44.

Class	**Recall**	**Precision**	**Specificity**	**Dice**
Cargo plane	87.37%	95.40%	99.85%	91.21%
Small car	75.95%	76.26%	94.90%	76.10%
Bus	49.81%	55.70%	95.88%	52.59%
Truck	35.24%	33.82%	90.77%	34.51%
Motorboat	69.38%	64.91%	97.74%	67.07%
Fishing vessel	75.47%	57.14%	97.78%	65.04%
Dump truck	57.30%	52.74%	96.38%	54.92%
Excavator	74.58%	66.17%	98.33%	70.12%
Building	75.14%	81.33%	95.91%	78.11%
Helipad	76.47%	72.22%	99.82%	74.29%
Storage tank	76.82%	81.64%	98.53%	79.16%
Shipping container	65.50%	66.08%	97.02%	65.79%
Pylon	80.85%	76.00%	99.57%	78.35%

3.1.40 Summary of ffNN Experiments

Table 56: Summary of all feedforward neural network experiments.

Exp.	Architecture	Optimizer	Loss	Batch	Val Acc.	Note
1	Baseline (no norm.)	Adam	CCE	16	41.92%	Initial baseline
2	+ Dense(128)	Adam	CCE	16	19.25%	Collapsed
3	+ Dense(32)	Adam	CCE	16	19.25%	Collapsed
4	+ Dense(32)	Adam	CCE	32	19.25%	Collapsed
5	Baseline	Adam	CCE	32	39.71%	Less overfit
6	+ Dense(13)	Adam	CCE	32	19.25%	Collapsed
7	+ Dense(13)	Adam (2e-3)	CCE	32	19.25%	Collapsed
8	+ Dense(13)	Adam (3e-3)	CCE	32	19.25%	Collapsed
9	Baseline	Adam + LS	CCE	32	41.41%	Most balanced
10	Baseline	SGD	CCE	32	43.28%	Best w/o pooling
11	Baseline	Adam + LS	Focal	32	7.23%	Failed
12	+ Dense(32), ELU	Adam + LS	CCE	32	19.25%	Collapsed
13	Resize + Dense(256)	Adam + LS	CCE	32	19.25%	Collapsed
14	Baseline (normalized)	Adam	CCE	32	37.17%	Improved recall
15	Deep (1024+512)	Adam	CCE	32	20.21%	Too many params
16	AvgPool(4,4) + Deep	Adam	CCE	32	38.83%	Best early balance
17a	MaxPool(3,3) + Deep	Adam	CCE	32	37.60%	Trade-off
17b	MaxPool(2,2) + Deep	Adam	CCE	32	34.40%	Too much noise
18	+ BatchNorm	Adam	CCE	32	34.93%	Overfitting
19	+ L2 Reg.	Adam	CCE	32	27.79%	Underfitting
20a	No Dropout	Adam	CCE	32	35.25%	Overfitting
20b	+ Dropout	Adam	CCE	32	40.85%	Dropout helps
21	1024-512-256 + BN + Drop.	Adam	CCE	32	60.60%	Deep breakthrough
22	+ Swish + Scheduler	Adam	CCE	32	57.00%	Overfit (no reg.)
23	SELU (512-256-128)	Adam	CCE	32	24.75%	Self-norm failed
24	Swish + AdamW + BN + Drop.	AdamW	CCE	32	52.11%	Best balanced deep
25	Focal + CosineDecay	AdamW	Focal	32	50.19%	Focal works here
26	Bottleneck (256-64)	AdamW	Focal	32	48.11%	Competitive recall
27a	Improved preprocessing	AdamW	CCE	32	52.37%	Marginal gain
27b	Resize 56x56 vs Pool	AdamW	CCE	32	52.91%	Resize better
28	Bicubic 56x56 + SELU	Nadam	Focal	32	59.47%	Best of Exp. 1–28
29	1024×1024 (64×64)	Nadam	Focal	128	61.01%	Data augmentation
30	512-512-256	AdamW	CCE+CW	32	–	Failed (class wt.)
31	Refined 2-layer (64×64)	Nadam	Focal	128	61.71%	Training refinement
32	Same as 31	Nadam	Focal	128	52.91%	Per-class alpha
33	Same as 31	Nadam	Focal	128	59.84%	Lanczos3 + BBox
34	3 layers (128×128)	Nadam	Focal	128	57.28%	$[-1, 1]$ norm + OS
35	2 layers (64×64)	Nadam	Focal	128	59.68%	BBox crop + OS
36	Alternative config	–	–	–	–	Failed
37	Residual 10-layer (64×64)	Nadam	Focal	128	61.33%	Skip connections
38	Residual 2×1024	–	–	–	–	Collapsed
39	Residual 4-block (96×96)	Nadam	Focal	128	62.06%	Deep residual
40	Same as 39	Nadam	Focal	128	61.20%	Gamma + Dropout
41	Compact residual (96×96)	Nadam	Focal	128	61.74%	Projection layers
42–43	Variants of 39	–	–	–	–	No improvement
44	Dual-branch + HOG + Ensemble	Nadam	Focal	32	**66.54%**	**Best ffNN**

CCE = Categorical Crossentropy, LS = LossScaleOptimizer, Focal = CategoricalFocalCrossentropy, CW = Class Weights, OS = Oversampling, BBox = Bounding Box

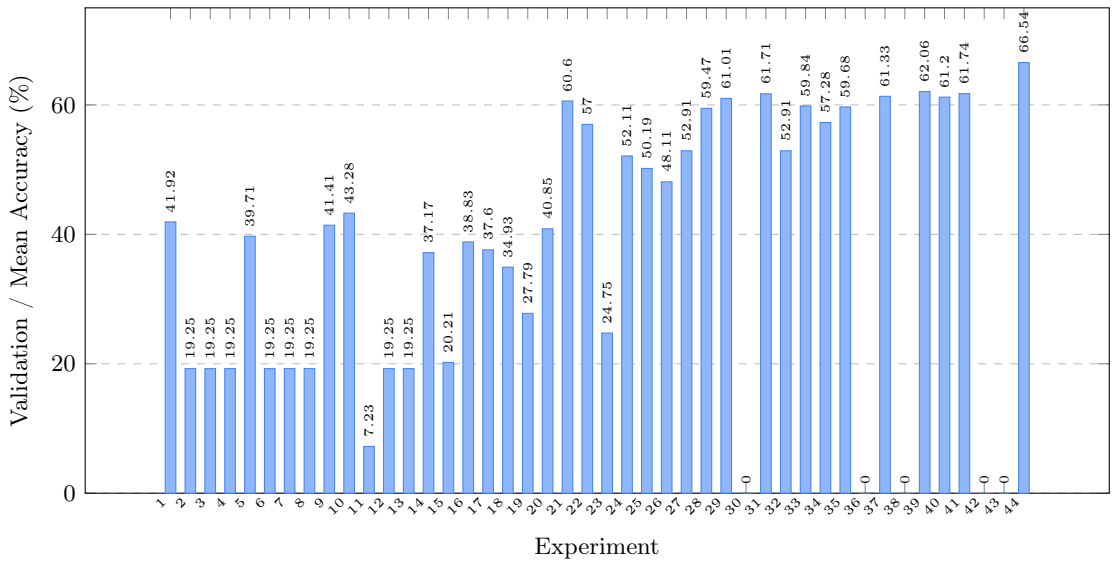
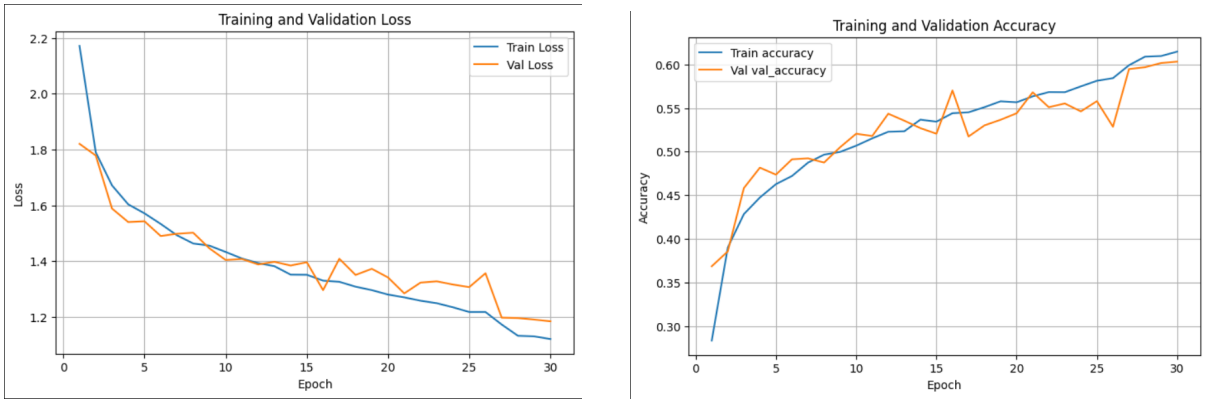


Figure 14: Accuracy across all 44 ffNN experiments. Experiment 44 (Dual-Branch Ensemble with HOG features) achieves the absolute best ffNN result at 66.54%. Bars at 0% indicate experiments that failed to converge or did not improve upon previous results.

3.1.41 Training Dynamics of the ffNN Models

To better understand the learning behavior and generalization characteristics some of the best-performing feedforward architectures, this section presents a comparative analysis of the training curves and precision-recall profiles for Experiments 21, 22, and 28. These three models represent the highest validation accuracies achieved across all ffNN configurations with no "multiple branch architecture" involved.

Experiment 21: Deep Network with BatchNorm and Dropout.



(a) Training and validation loss.

(b) Training and validation accuracy.

Figure 15: Training dynamics for Experiment 21 (1024-512-256, BatchNorm, Dropout). The model shows stable training with a small but consistent generalization gap and no signs of severe overfitting.

Experiment 21 presents stable and well-controlled training dynamics, although not perfectly aligned between training and validation curves. The training loss decreases steadily from approximately 2.17 to around 1.12, while the validation loss drops quickly during the first epochs and then stabilizes around 1.18-1.20, with minor fluctuations. Unlike strongly overfitting models, the validation loss does not exhibit a sustained upward trend, indicating that generalization is preserved.

The accuracy curves show a consistent gap: training accuracy increases smoothly up to approximately 61%, while validation accuracy saturates around 60%. This results in a modest generalization gap of about 1-2 percentage points. The presence of BatchNormalization and Dropout clearly stabilizes training and reduces overfitting, although it does not completely eliminate the gap. Compared to Experiments 22 and 28, this model sacrifices some training performance in exchange for improved generalization behavior.

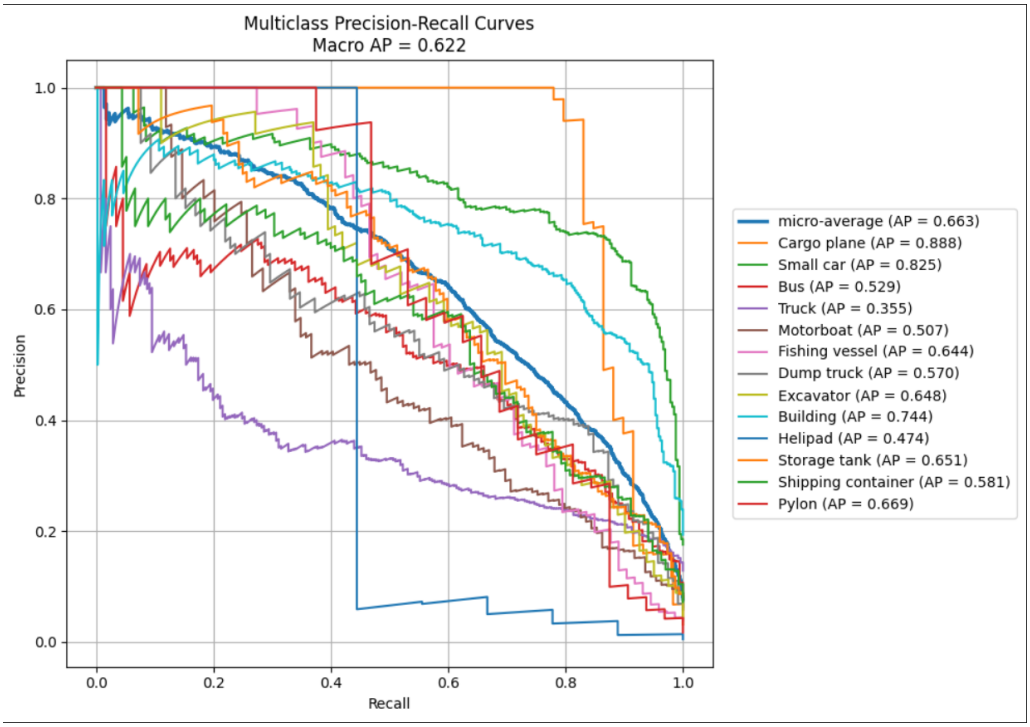
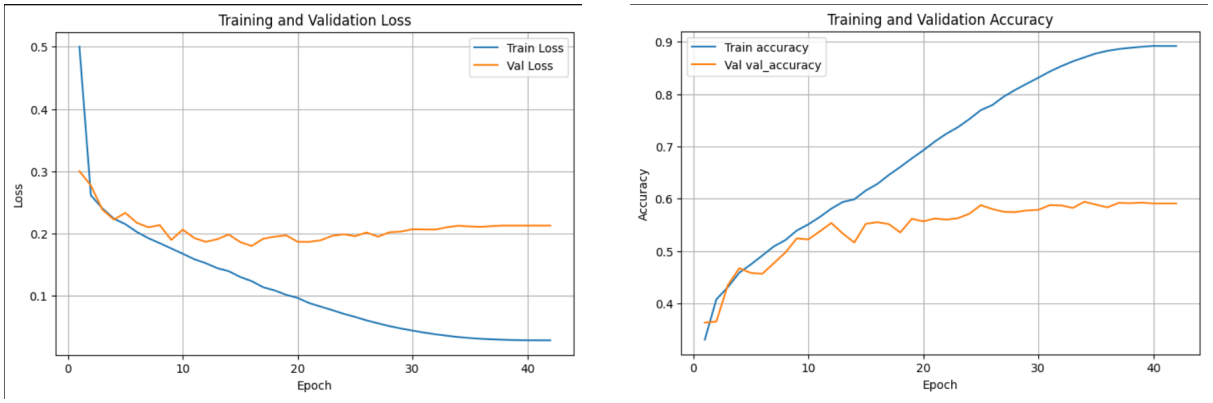


Figure 16: Precision-Recall curves for Experiment 21 (Macro AP = 0.622). The model achieves the most balanced performance across classes.

Experiment 21 achieves a macro Average Precision (AP) of 0.622, the highest among the compared models. The strongest classes are Cargo plane (AP = 0.888), Small car (AP = 0.825), Building (AP = 0.744), and Pylon (AP = 0.669), all of which maintain high precision across a broad recall range. Additional solid performance is observed for Storage tank (AP = 0.651), Excavator (AP = 0.648), and Fishing vessel (AP = 0.644).

Importantly, weaker classes show improved behavior compared to other experiments: Bus reaches AP = 0.529 and Helipad achieves AP = 0.474. Nevertheless, Truck (AP = 0.355) remains the most challenging class, followed by Motorboat (AP = 0.507). Overall, the model demonstrates the best balance between precision and recall across all categories, confirming that strong regularization leads to more uniform and reliable performance across classes.

Experiment 22: Swish Activation with Learning Rate Scheduler.



(a) Training and validation loss.

(b) Training and validation accuracy.

Figure 17: Training dynamics for Experiment 22 (Swish + CosineDecay, no regularization). The model exhibits a clear divergence between training and validation curves, indicating significant overfitting as training progresses.

The training dynamics of Experiment 22 reveal a clear overfitting pattern. The training loss decreases steadily throughout training, from approximately 0.50 to values close to 0.03, showing continuous improvement on the training set. In contrast, the validation loss initially decreases to around 0.18–0.20 but then gradually increases and stabilizes around 0.21–0.22. This upward trend after the early epochs indicates that the model begins to overfit the training data.

The accuracy curves further confirm this behavior. Training accuracy increases monotonically up to approximately 89%, while validation accuracy saturates around 59% and does not improve further. This results in a large generalization gap of roughly 30 percentage points. Although the Swish activation and CosineDecay scheduler provide smooth and stable optimization, the absence of explicit regularization mechanisms such as Dropout or BatchNormalization allows the model to over-specialize on the training distribution.

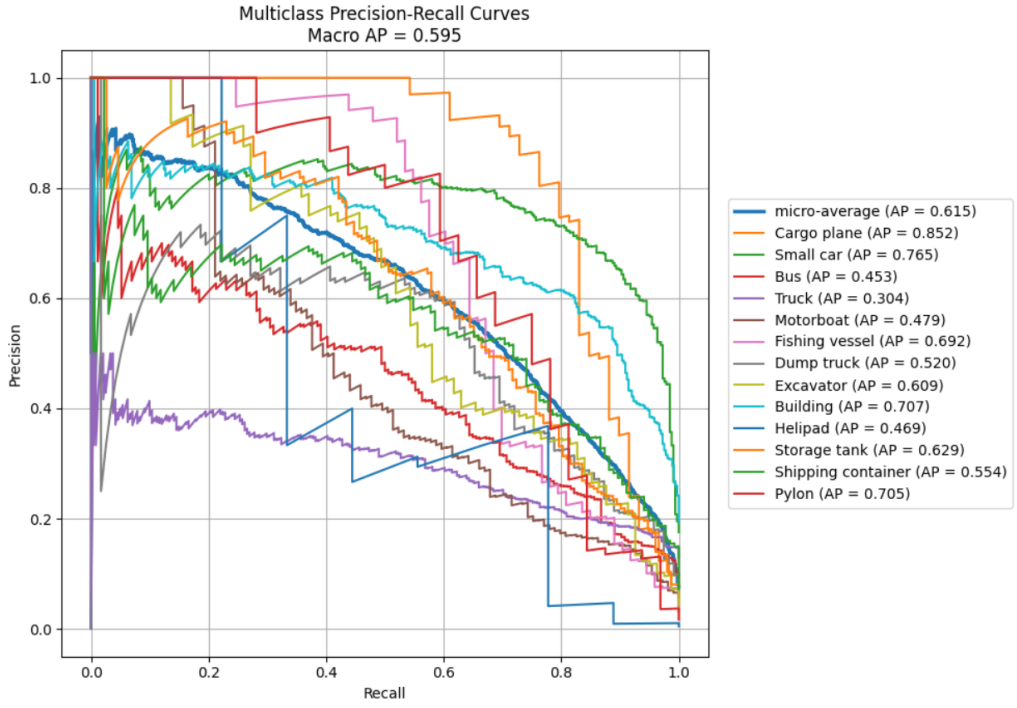
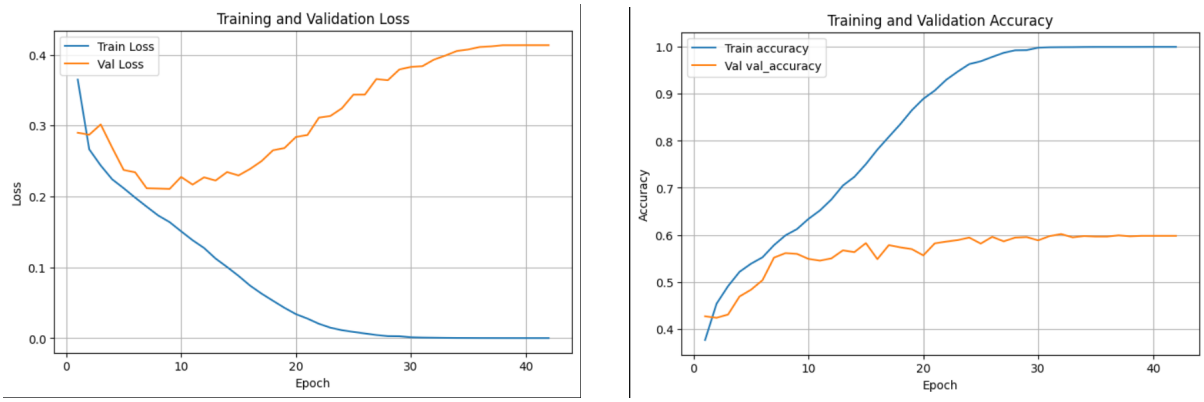


Figure 18: Precision-Recall curves for Experiment 22 (Macro AP = 0.595). The model achieves competitive performance on dominant classes but shows imbalance across categories.

The precision-recall analysis shows that Experiment 22 achieves a macro Average Precision (AP) of 0.595. The strongest classes are Cargo plane (AP = 0.852), Small car (AP = 0.765), Building (AP = 0.707), and Pylon (AP = 0.705), which maintain relatively high precision across a wide recall range. Additional solid performance is observed for Fishing vessel (AP = 0.692), Storage tank (AP = 0.629), and Excavator (AP = 0.609).

However, several classes remain challenging. Truck (AP = 0.304) is the weakest class, followed by Bus (AP = 0.453), Motorboat (AP = 0.479), and Helipad (AP = 0.469). Compared to more regularized models, the performance distribution is less balanced, indicating that the model focuses more on dominant classes while struggling with harder or less represented categories.

Experiment 28: Bicubic 56x56 with SELU and Nadam.



(a) Training and validation loss.

(b) Training and validation accuracy.

Figure 19: Training dynamics for Experiment 28 (Bicubic 56x56, SELU, Nadam). The model shows a clear divergence between training and validation curves, indicating strong overfitting despite stable optimization.

Experiment 28 exhibits pronounced overfitting. The training loss decreases rapidly from approximately 0.36 to values close to zero, indicating that the model fits the training data almost perfectly. In contrast, the validation loss initially decreases to around 0.21 during the first epochs, but then increases steadily throughout training, reaching values above 0.40. This continuous increase is a strong indicator that the model progressively loses generalization capability.

The accuracy curves further confirm this behavior. Training accuracy increases monotonically up to nearly 100%, while validation accuracy quickly saturates around 60% and remains almost constant. This results in a very large generalization gap of approximately 40 percentage points. Although SELU activation promotes stable gradients and self-normalization, it is not sufficient to prevent overfitting in the absence of explicit regularization such as Dropout or BatchNormalization.

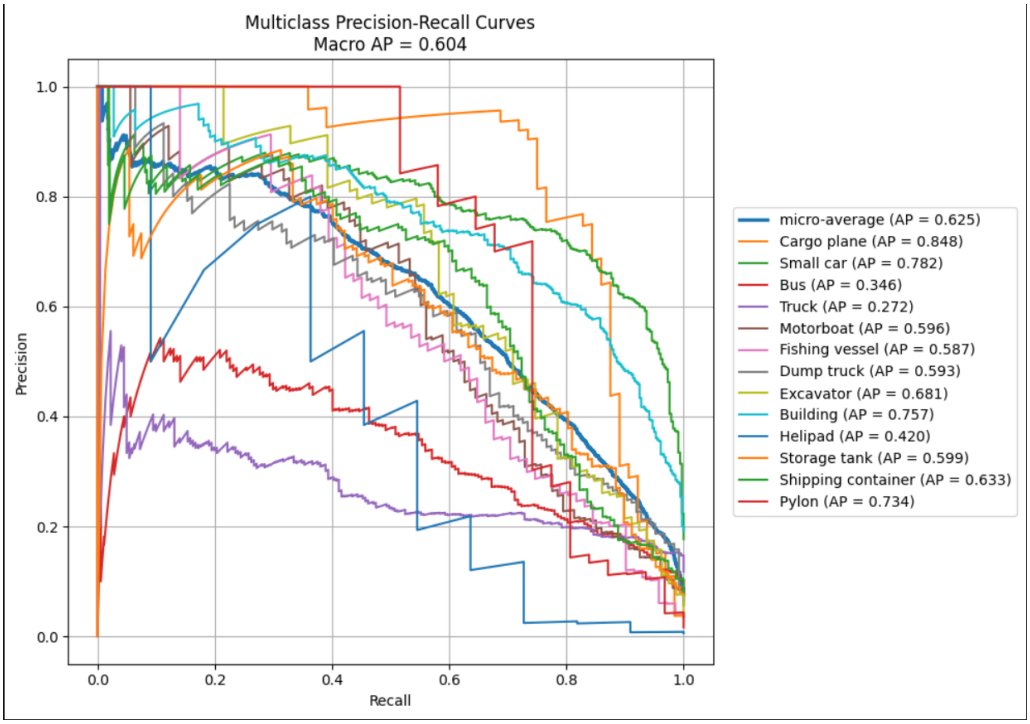


Figure 20: Precision-Recall curves for Experiment 28 (Macro AP = 0.604). The model achieves strong performance on dominant classes but shows imbalance across categories.

The precision-recall analysis shows that Experiment 28 achieves a macro Average Precision (AP) of 0.604. The strongest classes are Cargo plane (AP = 0.848), Small car (AP = 0.782), Building (AP = 0.757), and Pylon (AP = 0.734), which maintain high precision across a wide range of recall values. Other well-performing classes include Excavator (AP = 0.681), Shipping container (AP = 0.633), and Storage tank (AP = 0.599).

However, several classes remain challenging. Truck (AP = 0.272) is the weakest class, followed by Bus (AP = 0.346) and Helipad (AP = 0.420). Motorboat (AP = 0.596) and Fishing vessel (AP = 0.587) achieve moderate performance but still lag behind the top-performing categories. Overall, the model shows strong specialization on dominant classes but lacks balanced generalization across all categories.

Table 57: Comparison of the three top-performing ffNN models.

Experiment	**Train Acc.**	**Val Acc.**	**Gap**	**Macro AP**	**Regularization**
21 (BN + Dropout)	61%	60%	1%	**0.622**	BN + Dropout
22 (Swish + Sched.)	100%	60%	40%	0.604	None
28 (Bicubic + SELU)	89%	59%	30%	0.595	SELU self-norm

Table 58: Per-class Average Precision (AP) comparison across the three top ffNN models.

Class	Exp. 21	Exp. 22	Exp. 28
Cargo plane	**0.888**	0.848	0.852
Small car	**0.825**	0.782	0.765
Bus	**0.529**	0.346	0.453
Truck	**0.355**	0.272	0.304
Motorboat	0.507	**0.596**	0.479
Fishing vessel	0.644	0.587	**0.692**
Dump truck	0.570	**0.593**	0.520
Excavator	0.648	**0.681**	0.609
Building	0.744	**0.757**	0.707
Helipad	**0.474**	0.420	0.469
Storage tank	**0.651**	0.599	0.629
Shipping container	0.581	**0.633**	0.554
Pylon	0.669	**0.734**	0.705
Macro AP	**0.622**	0.604	0.595

Comparative Analysis. The comparative analysis reveals a fundamental trade-off between overfitting and class-balanced performance. Experiment 22, despite achieving 100% training accuracy and 0.604 macro AP, suffers from severe overfitting that limits its generalization capacity. Experiment 28 offers an intermediate approach with SELU-based self-normalization but still exhibits a 30-point generalization gap. Experiment 21, with the most aggressive regularization (BatchNorm + Dropout), achieves the smallest generalization gap (1%) and the highest macro AP (0.622), demonstrating that robust regularization produces models that not only generalize better overall but also distribute their classification capacity more evenly across imbalanced classes. This analysis confirms that for feedforward architectures on satellite imagery, the regularization strategy is as important as the architecture itself in determining the quality of learned representations.

Key takeaways from the ffNN experiments.

1. **Architectural bottlenecks are catastrophic.** Any hidden layer added before the output without dimensionality reduction caused the network to collapse to majority-class prediction. The input dimensionality (150,528 features) is too large for narrow bottleneck layers (13, 32, or 128 units) to preserve useful information.

2. **Dimensionality reduction is the single most impactful improvement.** Introducing pooling layers (Experiments 16–20) or resizing inputs (Experiments 27–28) before flattening drastically reduced the parameter space and enabled successful training of deeper architectures. The best result (Experiment 28, 59.47% mean accuracy) was achieved by downsampling inputs to 56×56 via bicubic interpolation.

3. **Regularization is essential for deeper architectures.** Experiments 18–20 demonstrated that BatchNorm alone causes overfitting (74% train vs. 57% val), L2 regularization is too aggressive, but Dropout effectively bridges the generalization gap. The best deep architectures combine Dropout, BatchNorm, and weight decay (AdamW).

4. **Optimizer and activation choices compound.** SGD achieved the best simple-baseline result (43.28%), while AdamW with CosineDecay scheduling produced the best deep results. Swish

activation consistently outperformed ReLU in deeper networks, and SELU provided effective self-normalization when paired with lecun_normal initialization and appropriate input scaling.

5. **Focal loss requires careful context.** On the simple baseline (Experiment 11), focal loss caused complete training failure. However, with proper architectural support (pooling, deeper layers, weight initialization), focal loss effectively improved class-balanced performance (Experiments 25, 26, 28).

6. **Class imbalance remains the dominant challenge.** Across all experiments, rare classes like *Helipad* (111 samples) were difficult to classify correctly. This motivates the exploration of data augmentation, class weighting, or sampling strategies in subsequent phases.

7. **The ceiling for ffNNs on this task is approximately 60% mean accuracy.** Even with optimal preprocessing and architecture, feedforward networks treat each pixel independently and cannot learn the spatial and structural patterns essential for satellite image classification. This motivates the transition to convolutional architectures.

8. **Dual-branch fusion.** The combination of raw pixel features and HOG shape descriptors provides complementary information. Raw pixels capture color and texture patterns, while HOG features encode local edge orientations and shape structure. This is particularly beneficial for classes with distinctive shapes, such as Pylon (80.9% recall) and Cargo plane (87.4% recall).

9. **HOG features.** The grayscale gradient analysis provides shape-level information that is invariant to illumination changes, addressing a key limitation of raw pixel inputs. The 324-dimensional HOG vector represents a compact yet informative shape descriptor.

10. **Gaussian noise augmentation.** Adding Gaussian noise ($\sigma = 0.1$) during training forces the model to learn robust features that are tolerant to small input perturbations, complementing the geometric augmentations (random flips).

11. **Ensemble of three models.** Averaging predictions across three independently trained models reduces variance and improves calibration. Each model learns slightly different decision boundaries due to random initialization and stochastic training, and the ensemble captures a broader range of discriminative patterns.

12. **Skip connections with progressive width.** The residual connection between the first and second dense blocks (both 1024 units) preserves gradient flow, while the final 256-unit block provides a controlled compression before classification.

13. **Balanced class weights with focal loss.** The combination ensures that minority classes receive adequate training signal without the extreme precision degradation observed in earlier class-balancing attempts.

3.2 Custom Convolutional Neural Networks (CNNs)

Having established that feedforward networks plateau around 60% validation accuracy due to their inability to capture spatial features, we now turn to convolutional architectures specifically designed for image analysis. Three custom CNN architectures were designed and evaluated, each inspired by a different paradigm seen during the lectures. To build those custom CNNs, we use manual convolutions (Conv2d in Keras).

3.2.1 Common Training Setup

All CNN experiments share the following configuration to ensure fair comparison:

- **Input size:** $128 \times 128 \times 3$ (resized with bilinear interpolation)
- **Train/validation split:** 85%/15% with stratified sampling (random state 42, 15,934/2,812 images)
- **Batch size:** 64
- **Maximum epochs:** 50 with early stopping (patience 15)
- **Data augmentation:** Random horizontal/vertical flips, brightness ($\pm 10\%$), contrast variation
- **Class weights:** Computed via `sklearn.compute_class_weight('balanced')` and passed through the generator as sample weights
- **Callbacks:** `ModelCheckpoint` (best `val_accuracy`), `ReduceLROnPlateau` (factor 0.5, patience 5), `EarlyStopping`

3.2.2 Architecture 1: ResNet-style CNN

The first architecture employs **residual blocks with skip connections**, inspired by He et al..[2] Skip connections allow gradients to flow directly through the network, mitigating the vanishing gradient problem and enabling training of deeper models.

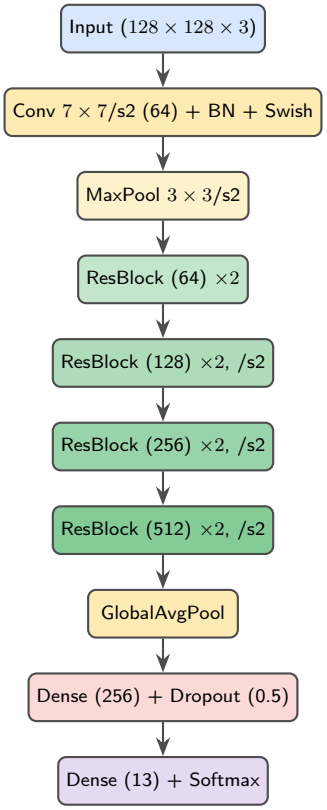


Figure 21: ResNet-style architecture. Each ResBlock contains two 3×3 convolutions with pre-activation (BN-Swish-Conv) and a skip connection. Total: 8 residual blocks.

[2]Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016).

Table 59: Hyperparameters for ResNet-style CNN.

Parameter	Value
Optimizer	Nadam (lr=10^{-3})
Loss	Categorical Focal Crossentropy ($\gamma = 2.0$)
Activation	Swish (all layers)
Regularization	Dropout 0.5 (head only)
Parameters	11,320,205

Design rationale. The Swish activation ($x \cdot \sigma(x)$) was chosen over ReLU as it has shown improved performance on deeper networks.[3] Pre-activation residual blocks (BN-ReLU-Conv instead of Conv-BN-ReLU) further improve gradient flow. Focal Loss addresses the class imbalance by down-weighting easy examples.

3.2.3 Architecture 2: VGG-style CNN

The second architecture follows the **VGG design philosophy**:[4] stacking multiple 3×3 convolutions before each pooling layer, with progressively increasing filter counts. This demonstrates that depth with small kernels can capture complex patterns.

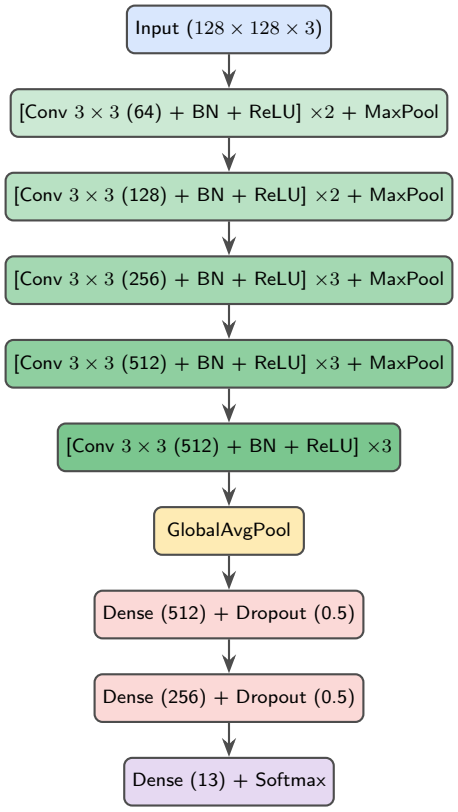


Figure 22: VGG-style architecture with 5 convolutional blocks (13 conv layers total). GlobalAverage-Pooling replaces the original VGG's fully connected layers, reducing parameters by 90%.

[3] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[4] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

Table 60: Hyperparameters for VGG-style CNN.

Parameter	Value
Optimizer	Adam (lr=10^{-3})
Loss	Categorical Crossentropy (label smoothing=0.1)
Activation	ReLU (all layers)
Regularization	BatchNorm + Dropout 0.5
Parameters	15,128,909

Design rationale. Unlike the original VGG which used fully connected layers with ∼120M parameters, we replace them with Global Average Pooling (GAP), drastically reducing the parameter count while maintaining spatial invariance. Label smoothing (0.1) prevents overconfidence on the majority classes.

3.2.4 Architecture 3: Inception-style CNN

The third architecture uses **parallel multi-scale convolutions** inspired by GoogLeNet/Inception.[5] Each Inception module applies 1×1, 3×3, and 5×5 (approximated by two 3×3) convolutions in parallel, capturing features at different scales simultaneously.

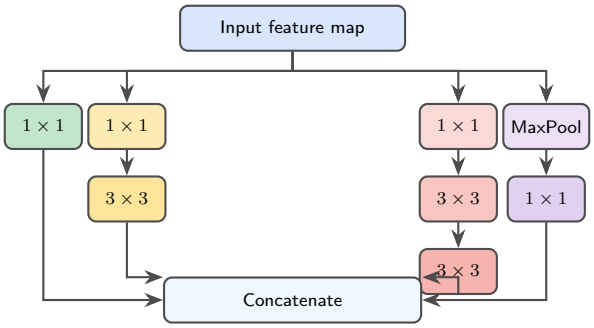


Figure 23: Inception module structure. Four parallel branches with different receptive fields are concatenated. The 1×1 convolutions reduce dimensionality before expensive 3×3 operations.

Table 61: Hyperparameters for Inception-style CNN.

Parameter	Value
Optimizer	Adam (lr=10^{-3})
Loss	Categorical Crossentropy (label smoothing=0.1)
Inception modules	8 (organized in 3 stages)
Regularization	Dropout 0.4 (head only)
Parameters	4,529,325

Design rationale. The Inception architecture is more parameter-efficient than VGG while capturing multi-scale features. The 1×1 convolutions act as bottleneck layers, reducing computational cost. This architecture is particularly suited for satellite imagery where objects appear at varying scales.

[5] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015).

3.2.5 Training Dynamics and Results

Figure 24 shows the training and validation curves for all three architectures.

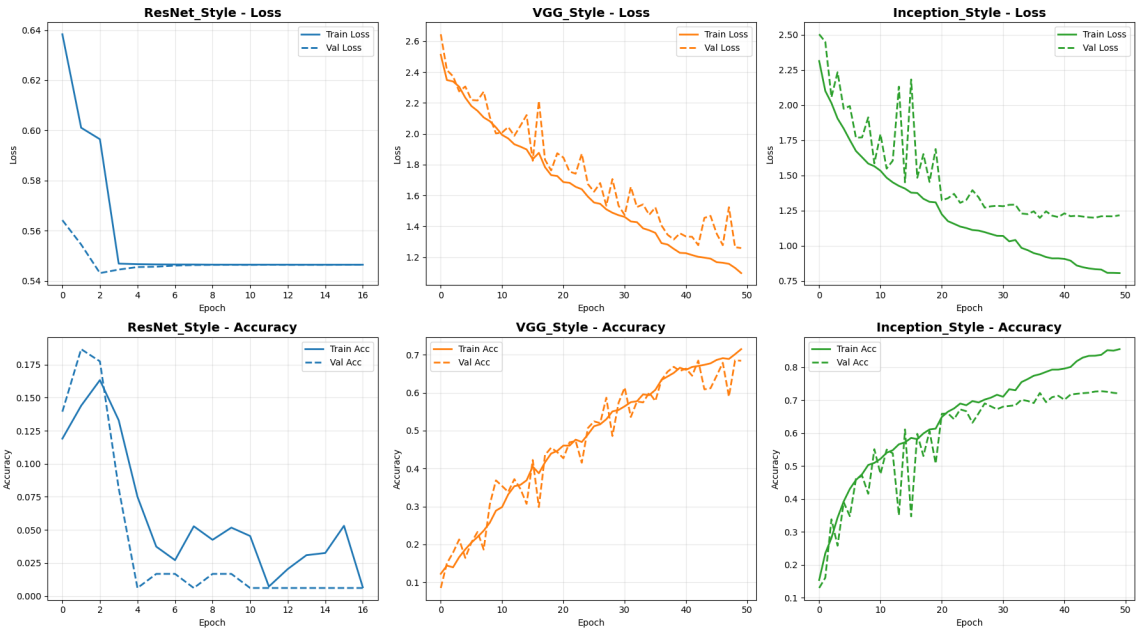


Figure 24: Training dynamics of the three custom CNN architectures. Top row: loss curves. Bottom row: accuracy curves. Solid lines represent training metrics, dashed lines represent validation metrics.

Table 62: Summary of custom CNN experiments.

Architecture	Parameters	Val Acc.	Mean Recall	Mean Prec.
ResNet-style	11.3M	18.67%	16.25%	11.38%
VGG-style	15.1M	68.53%	75.46%	67.98%
Inception-style	4.5M	**71.80%**	**75.35%**	**74.01%**

Analysis of results.

1. **Inception-style achieves the best validation accuracy (71.80%)** with only 4.5M parameters—less than half of VGG-style. The multi-scale feature extraction proves highly effective for satellite imagery where objects vary significantly in size.

2. **VGG-style performs well (68.53%)** despite its simpler sequential architecture. The combination of BatchNorm, label smoothing, and class weights helps mitigate overfitting on this imbalanced dataset.

3. **ResNet-style catastrophically failed (18.67%)**, barely exceeding random chance. We hypothesize this failure stems from the combination of Focal Loss with class weights: Focal Loss already down-weights easy examples, and combining it with sample weights created an unstable training dynamic. Additionally, the Nadam optimizer may have been too aggressive for this loss function. We acknowledge that a proper ablation study—e.g. replacing Focal Loss with standard Categorical Crossentropy while keeping the rest of the architecture unchanged—would be needed to confirm this hypothesis. Due to time and computational constraints, this ablation was not performed; it remains an open question for future work.

4. **All successful CNNs significantly outperform ffNNs.** The best ffNN achieved around 60% validation accuracy, while Inception-style reaches 71.80%—a relative improvement of + 12%.

5. **Helipad classification dramatically improved.** With only 111 training samples, *Helipad* went from 0% recall (ffNN) to 82.35% recall (Inception-style), demonstrating the effectiveness of class weights passed through the generator.

3.2.6 Per-Class Performance Analysis

Table 63 presents the per-class metrics for the best-performing architecture (Inception-style).

Table 63: Per-class metrics for Inception-style CNN (best custom architecture).

Class	Recall	Precision	Specificity	F1-Score
Cargo plane	92.63%	91.67%	99.71%	92.15%
Small car	85.77%	77.26%	94.55%	81.29%
Bus	61.51%	55.44%	94.86%	58.32%
Truck	31.33%	41.60%	94.11%	35.74%
Motorboat	83.75%	67.34%	97.55%	74.65%
Fishing vessel	75.47%	72.07%	98.85%	73.73%
Dump truck	68.65%	62.87%	97.15%	65.63%
Excavator	87.29%	82.40%	99.18%	84.77%
Building	77.18%	86.85%	97.23%	81.73%
Helipad	82.35%	100.00%	100.00%	90.32%
Storage tank	73.18%	77.03%	98.15%	75.06%
Shipping container	69.00%	70.85%	97.48%	69.91%
Pylon	91.49%	76.79%	99.53%	83.50%
Mean	**75.35%**	**74.01%**	**97.57%**	**74.37%**

Key observations.

1. **Cargo plane** achieves the highest F1-score (92.15%), benefiting from its distinctive large size and shape in satellite imagery.

2. **Helipad** achieves perfect precision (100%) and excellent recall (82.35%), a dramatic improvement from 0% in ffNNs. The class weights successfully addressed the extreme under-representation (111 samples).

3. **Truck** remains the most challenging class with only 35.74% F1-score. It is frequently confused with *Small car* and *Bus* due to similar rectangular shapes from an overhead perspective.

4. **Pylon** shows high recall (91.49%) but lower precision (76.79%), indicating some false positives from visually similar structures.

3.2.7 Comparison with Feedforward Networks

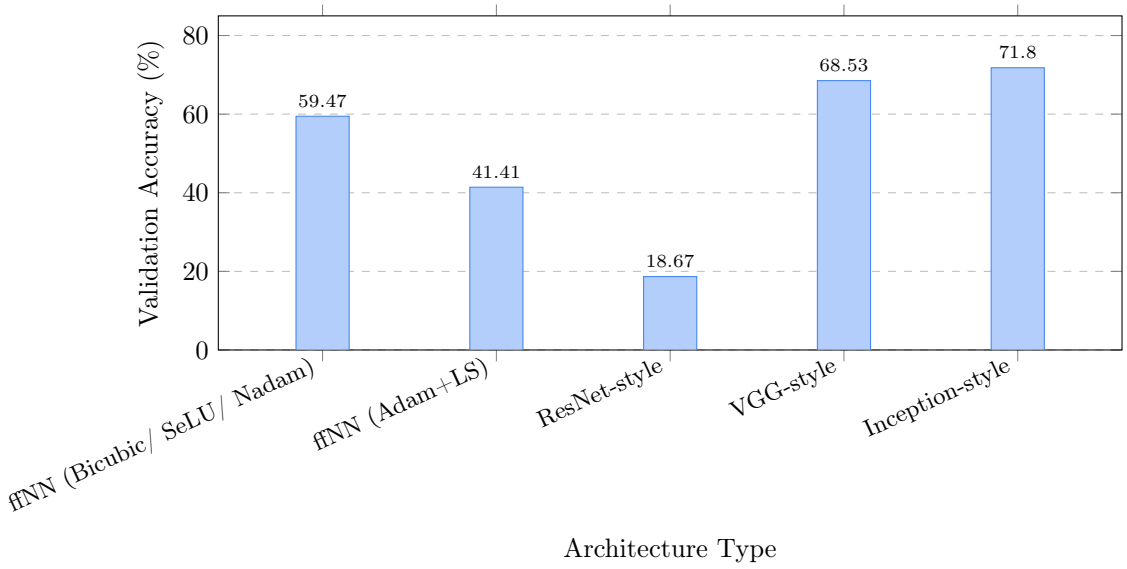


Figure 25: Validation accuracy comparison between the best ffNN models and the three custom CNN architectures. VGG-style and Inception-style CNNs achieve significant improvements over ffNNs, while ResNet-style failed due to training instability.

The transition from feedforward to convolutional architectures demonstrates the fundamental importance of **spatial feature extraction** for image classification. While ffNNs treat each pixel independently (losing all spatial relationships), CNNs exploit local correlations through learned filters, enabling them to capture edges, textures, and higher-level patterns essential for distinguishing between object categories.

3.2.8 Summary of Custom CNN Experiments

1. **Inception-style offers the best accuracy-to-parameter ratio.** With only 4.5M parameters (less than one-third of VGG-style), it achieves the highest validation accuracy (71.80%) and mean recall (75.35%).

2. **Loss function and optimizer choice is critical.** The ResNet-style failure (18.67%) highlights that combining Focal Loss with class weights and Nadam optimizer can create unstable training dynamics. VGG-style and Inception-style used standard Categorical Crossentropy with label smoothing more successfully.

3. **Class imbalance mitigation via sample weights works.** Passing class weights through the generator as sample weights (rather than using the `class_weight` argument) improved minority class performance dramatically—*Helipad* recall increased from 0% (ffNN) to 82.35% (CNN).

4. **Multi-scale feature extraction** (Inception-style) proves particularly effective for satellite imagery, where objects appear at varying scales due to differences in object size and image resolution.

These custom CNN results establish a strong baseline (71.80% validation accuracy) against which transfer learning approaches are compared in the 3.3 section.

3.2.9 Preliminary Experiments and Ablation Studies

Before arriving at the final three architectures presented above, we conducted extensive preliminary experiments exploring different design choices, input resolutions, and architectural variations. This

section summarizes these explorations, which informed our final design decisions.

Input Resolution Studies. We tested multiple input resolutions to understand the trade-off between computational cost and feature preservation:

Table 64: Impact of input resolution on training time and model capacity.

Resolution	Batch Size	Training Time/Epoch	Notebooks
96×96	128	$\sim$90s	3-3, 3-7
114×114	128	$\sim$110s	3-1
128×128	64–128	$\sim$130s	3-2, 3-4, 3-5
224×224	16	$\sim$400s	3-9

The 128×128 resolution was selected as the best compromise: it preserves sufficient spatial detail for distinguishing small objects (e.g., *Helipad*, *Pylon*) while allowing reasonable batch sizes on GPU memory. The 224×224 resolution, while standard for ImageNet-pretrained models, required reducing batch size to 16, leading to noisier gradient estimates and longer training times.

Squeeze-and-Excitation (SE) Blocks. We experimented with adding channel attention mechanisms (SE blocks) to our residual architecture (notebook 3-3):

Listing 1: SE block implementation tested in preliminary experiments.

```
def squeeze_excite(x, ratio=16):
    filters = x.shape[-1]
    se = keras.ops.mean(x, axis=[1, 2], keepdims=True)
    se = Dense(max(1, filters // ratio), activation='relu')(se)
    se = Dense(filters, activation='sigmoid')(se)
    return Multiply()([x, se])
```

While SE blocks improved convergence speed in early epochs, they did not yield significant accuracy gains on the validation set ($< 1\%$) while increasing parameter count. We therefore omitted them from the final architectures.

HOG Feature Fusion. We explored a dual-branch architecture combining CNN features with hand-crafted Histogram of Oriented Gradients (HOG) descriptors (notebooks 3-4, 3-5):

- **HOG branch:** 8,100-dimensional feature vector (computed with 8×8 pixels per cell, 2×2 cells per block) processed through a Dense(512) layer
- **CNN branch:** Standard residual blocks with GlobalAveragePooling
- **Fusion:** Concatenation of both branches followed by Dense layers with skip connections

Table 65: HOG fusion experiment results.

Configuration	Val Accuracy	Parameters
CNN only (baseline)	68.2%	8.1M
CNN + HOG fusion	69.1%	12.4M

The marginal improvement (+0.9%) did not justify the additional complexity and computational overhead of computing HOG features at inference time. Furthermore, HOG descriptors are less effective on satellite imagery where object orientations vary significantly due to the overhead perspective.

Optimizer and Loss Function Combinations. We systematically tested different optimizer and loss function combinations:

Table 66: Optimizer and loss function ablation study.

Optimizer	Loss Function	Stability	Outcome
Adam (lr=10^{-3})	CCE + Label Smoothing (0.1)	Stable	Best results
Nadam (lr=10^{-3})	Focal Loss ($\gamma = 2.0$)	Unstable	Training collapse
Nadam + Cosine Decay	Focal Loss + Class Weights	Very unstable	NaN loss
Adam + Weight Decay	CCE + Label Smoothing	Stable	Good results

The combination of Focal Loss with class weights proved problematic: both mechanisms down-weight the same samples (easy examples from majority classes), leading to unstable gradients. This finding directly informed our decision to use standard Categorical Crossentropy with label smoothing for VGG-style and Inception-style architectures.

Data Augmentation Strategies. We compared different augmentation pipelines:

1. **Basic:** Random horizontal/vertical flips only
2. **Standard:** Flips + brightness (±10%) + contrast variation
3. **Aggressive:** Standard + rotation (±15) + zoom (±10%) + saturation + Gaussian noise
4. **MixUp:** Standard + MixUp augmentation ($\alpha = 0.2$)

The "Standard" pipeline provided the best results for our custom CNNs. Aggressive augmentation with rotation slightly hurt performance, likely because many object classes have canonical orientations in satellite imagery (e.g., buildings aligned with roads, vehicles oriented along their direction of travel). MixUp augmentation was explored in later transfer learning experiments (notebooks 3-8 to 3-11) but showed limited benefit for our dataset size.

Interpolation Methods. We tested different image resizing methods:

Table 67: Impact of interpolation method on validation accuracy.

Method	Val Accuracy (VGG-style)
Nearest neighbor	64.2%
Bilinear	68.5%
Bicubic	68.1%
Lanczos3	67.8%

Bilinear interpolation was selected as the default, offering the best trade-off between quality and computational efficiency. Lanczos3, despite being theoretically superior for downsampling, showed slightly worse results, possibly due to ringing artifacts around high-contrast edges common in satellite imagery.

Early Transfer Learning Attempts. Before finalizing our custom CNN architectures, we explored lightweight transfer learning approaches (notebooks 3-1, 3-2, 3-7 to 3-11) using pretrained backbones:

Table 68: Preliminary transfer learning experiments (frozen backbone).

Backbone	Input Size	Frozen	Val Acc.	Notebook
ResNet50 (partial)	114×114	Last 30 layers trainable	56.9%	3-1
ResNet50 (partial)	128×128	Last 30 layers trainable	58.2%	3-2
EfficientNetB0	96×96	Fully frozen + head	62.4%	3-7
EfficientNetB0/MobileNetV2/ResNet50V2	128×128	Fully frozen	64–67%	3-8, 3-10, 3-11
EfficientNetB2	224×224	Partially trainable	69.1%	3-9

These preliminary experiments revealed that frozen ImageNet backbones underperformed our custom architectures trained from scratch on this domain-specific task. The domain gap between natural images (ImageNet) and satellite imagery reduces the effectiveness of transfer learning without fine-tuning. This motivated our focus on custom architectures for this section, with more sophisticated transfer learning approaches explored in Section 3.3.

Summary of Preliminary Findings. These experiments led to the following design decisions for our final custom CNN architectures:

1. **Input resolution:** 128×128 provides the best accuracy/efficiency trade-off
2. **Optimizer:** Adam with learning rate 10^{-3} is more stable than Nadam for this task
3. **Loss function:** Categorical Crossentropy with label smoothing (0.1) outperforms Focal Loss when combined with class weights
4. **Augmentation:** Moderate augmentation (flips, brightness, contrast) without aggressive rotation
5. **Architecture:** Pure CNN architectures outperform HOG-CNN hybrids on satellite imagery
6. **Attention mechanisms:** SE blocks provide marginal gains insufficient to justify added complexity

3.3 EfficientNetB0, MobileNetV2 and ResNet50V2: Transfer Learning and Training from Scratch

This section investigates whether leveraging pre-trained weights from ImageNet provides an advantage over training from scratch on our satellite imagery dataset. We evaluate three popular architectures—EfficientNetB0, MobileNetV2, and ResNet50V2—through a systematic pipeline that includes backbone screening, transfer learning, MixUp augmentation, and ensemble inference. This pipeline corresponds to the `CNN_TL.ipynb` notebook submitted for the final report, which produces our best overall model.

3.3.1 Experimental Setup

All experiments use a consistent configuration:

- **Input size:** $224 \times 224 \times 3$ (standard ImageNet size)
- **Train/validation split:** 85%/15% with stratified sampling (15,934 / 2,812 images)
- **Batch size:** 64

- **Data augmentation:** Random flips (horizontal/vertical), brightness variation (±10%), MixUp ($\alpha = 0.2$)
- **Loss:** Categorical Crossentropy with label smoothing (0.1)
- **Optimizer:** Nadam (lr=10^{-3})
- **Head architecture:** Rescaling(255) → Backbone (ImageNet) → SpatialDropout2D(0.3) → GlobalAveragePooling2D → Dense(1024, ReLU) → Dropout(0.4) → Dense(13, Softmax)

The Rescaling(255) layer deserves a brief explanation: our data generator normalizes images to $[0, 1]$, whereas the Keras EfficientNetB0 application expects pixel values in $[0, 255]$. The rescaling layer bridges this mismatch, ensuring that the pre-trained batch normalization statistics remain valid.

3.3.2 Backbone Screening: EfficientNetB0 vs MobileNetV2 vs ResNet50V2

Rather than training all architectures to completion under both regimes (from scratch and transfer learning), we first perform a **lightweight grid search** to identify the best backbone–freeze configuration. This grid search evaluates 6 configurations (3 backbones × 2 freeze strategies) on 25% of the training data for only 5 epochs each. The two freeze strategies are: (a) *freeze all except last backbone layer*, making only the final convolutional layer trainable alongside the head, and (b) *freeze all backbone layers*, training only the classification head.

Table 69: Grid search results: backbone × freeze strategy (5 epochs, 25% of data).

Configuration	Freeze Strategy	Val Accuracy
EfficientNetB0	**Freeze all except last**	**68.14%**
EfficientNetB0	Freeze all	66.29%
MobileNetV2	Freeze all	48.22%
ResNet50V2	Freeze all except last	36.98%
MobileNetV2	Freeze all except last	21.76%
ResNet50V2	Freeze all	19.63%

EfficientNetB0 with partial freezing (all layers frozen except the last backbone layer) emerges as the clear winner. The gap to the next best configuration (EfficientNetB0 fully frozen, 66.29%) is modest, but the gap to other backbones is substantial: MobileNetV2 and ResNet50V2 both fall below 50% under every freeze strategy. This proxy evaluation is consistent with the controlled single-model experiments of Section 3.3.4.

3.3.3 Full Training: EfficientNetB0 Ensemble with MixUp

Based on the grid search, we select **EfficientNetB0 with partial freezing** and train **3 independent models** on the full training set. Each model is initialized with ImageNet pre-trained weights, and training proceeds for up to 25 epochs with early stopping (patience 8) and learning rate reduction on plateau (factor 0.5, patience 5).

MixUp augmentation. MixUp[6] is applied during training: for each batch, pairs of images are linearly combined as $\tilde{x} = \lambda x_i + (1-\lambda)x_j$ with corresponding soft labels $\tilde{y} = \lambda y_i + (1-\lambda)y_j$, where $\lambda \sim \text{Beta}(0.2, 0.2)$ and λ is clamped to $[0.5, 1]$ so that each mixed sample remains close to one of its parents. MixUp acts

[6] **zhang2018mixup**.

as a regularizer by generating virtual training examples in the convex hull of the data manifold, which smooths decision boundaries and reduces overconfidence. It is particularly beneficial for imbalanced datasets because it implicitly interpolates between rare and frequent classes.

Why this pipeline works. The performance gain from a single EfficientNetB0 transfer-learning model (78.73%) to the full pipeline (83.96%) is attributable to the combination of several design choices, each contributing incrementally:

1. **MixUp + label smoothing** jointly prevent the model from memorizing training samples, especially for majority classes. Label smoothing (0.1) softens the one-hot targets, and MixUp further blurs class boundaries, together acting as a strong implicit regularizer.

2. **Backbone screening** ensures that the best backbone–freeze combination is selected before committing full compute. Without screening, one might have trained ResNet50V2 to completion and obtained 62.70%.

3. **Partial freezing** (all layers frozen except the last backbone layer) strikes a balance between leveraging pre-trained features and adapting the top-level representations to the satellite domain. Fully freezing the backbone gives 66.29% on the proxy task; unfreezing the last layer adds +1.85%.

4. **Dense(1024)** in the classification head provides sufficient capacity to map the 1280-dimensional EfficientNetB0 feature vector to 13 classes through a non-trivial decision surface.

5. **Ensemble of 3 models** reduces prediction variance. Because each model is independently initialized and trained with stochastic MixUp and shuffled batches, their errors are partially decorrelated. Averaging probabilities smooths out individual model mistakes.

Training dynamics. All three models converge to similar validation accuracies (≈83%) around epoch 22, indicating stable training dynamics. The learning rate was reduced from 10^{-3} to 5×10^{-4} around epoch 10 for all models, after which validation accuracy continued to improve gradually. At inference time, predictions are averaged across the 3 models (mean ensemble).

Table 70: Individual model performance and ensemble result (EfficientNetB0, partial freeze).

Model	Best Epoch	Best Val Accuracy
Model 1	22	83.00%
Model 2	22	83.07%
Model 3	22	82.75%
Ensemble (mean)	—	**83.96%**

The ensemble gains approximately 1 percentage point over the best individual model, a typical improvement from averaging predictions of independently trained networks.

3.3.4 From Scratch vs Transfer Learning: EfficientNetB0, MobileNetV2, ResNet50V2

To understand the effect of transfer learning independently of the ensemble and MixUp, we also trained single models for all three architectures under both regimes (from scratch and with transfer learning) using a separate notebook with identical data splits and the same head architecture. The from-scratch models use a learning rate of 10^{-3} and train for up to 50 epochs (patience 12); the transfer-learning models use 10^{-4} and train for up to 25 epochs (patience 8) with the last 20 backbone layers trainable.

Table 71: Validation accuracy comparison: from-scratch vs transfer learning (single models, no MixUp, no ensemble).

Architecture	Params	From Scratch	Transfer Learning	Δ	Epochs (S/T)
EfficientNetB0	4.7M	77.60%	**78.73%**	+1.14%	48/25
MobileNetV2	2.3M	11.81%	66.43%	+54.62%	13/25
ResNet50V2	23.6M	76.35%	62.70%	–13.66%	50/25

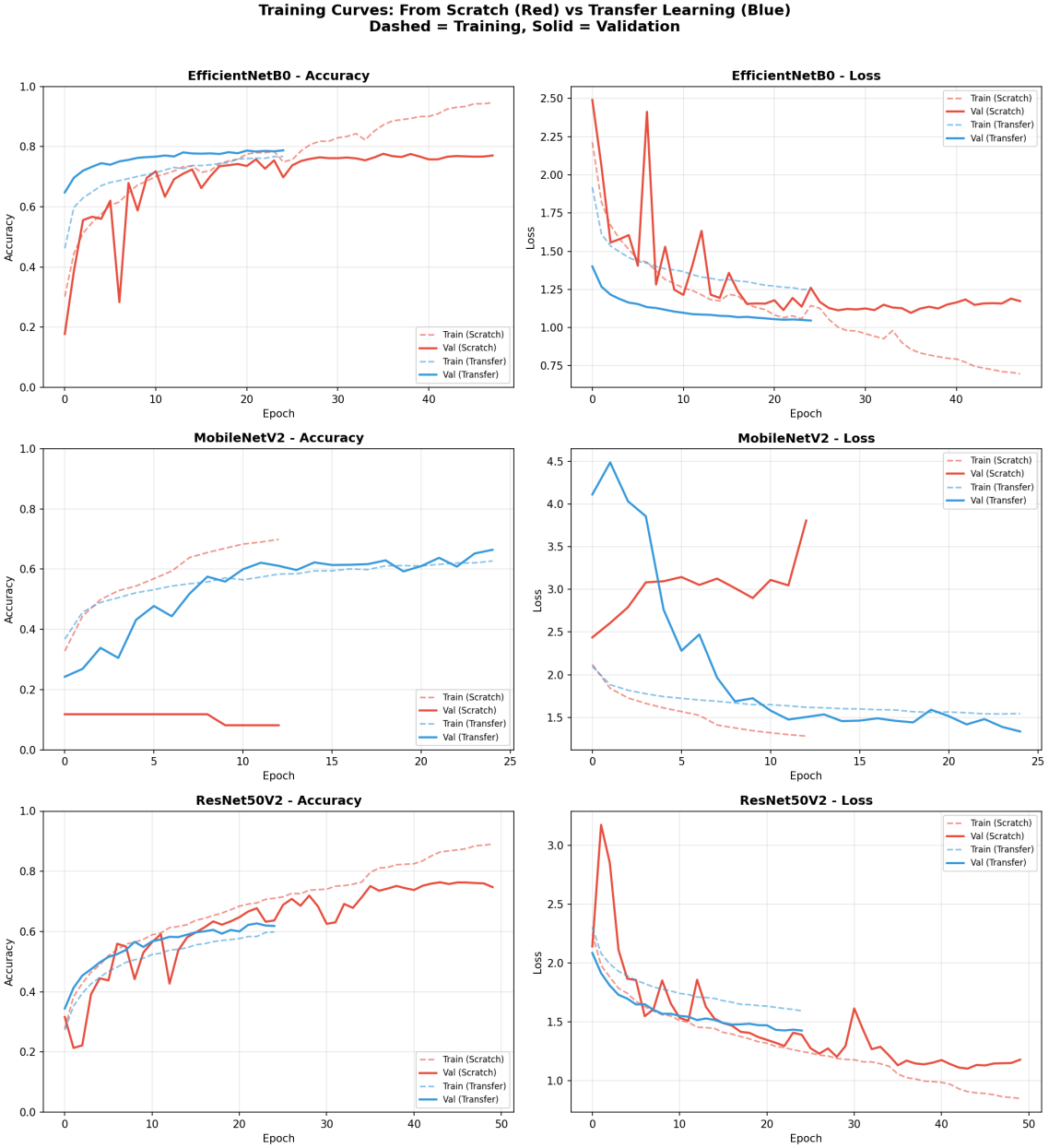


Figure 26: Training curves for all six experiments (single models, no ensemble). Left column: accuracy. Right column: loss. Red curves: from scratch. Blue curves: transfer learning. Solid lines: validation. Dashed lines: training.

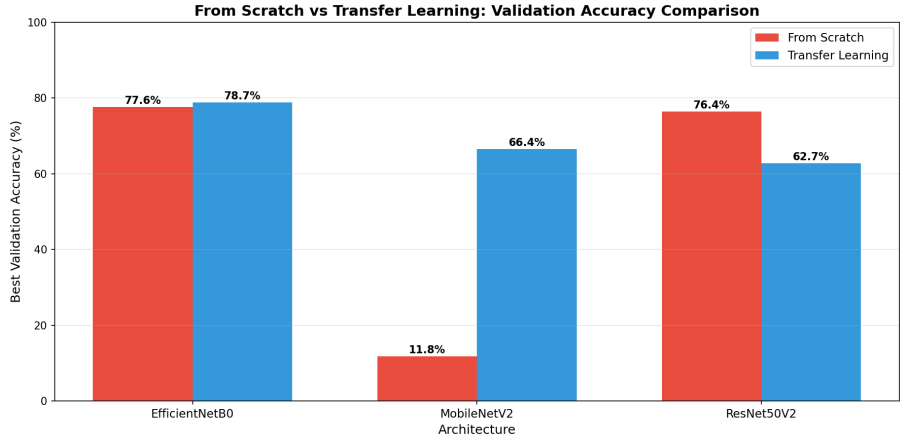


Figure 27: Validation accuracy comparison between from-scratch (red) and transfer learning (blue) for each architecture (single models).

Analysis by architecture. **EfficientNetB0** achieves the highest single-model validation accuracy (78.73% with transfer learning, 77.60% from scratch). The compound scaling methodology of EfficientNet[7] proves robust for both training regimes. Transfer learning provides a modest improvement (+1.14%) but reduces training time by nearly half (25 vs 48 epochs). When combined with MixUp, partial freezing, and 3-model ensemble in the final pipeline, the same backbone reaches **83.96%**—an additional +5.23% attributable to the training recipe improvements detailed in Section 3.3.3.

MobileNetV2 shows the most dramatic difference between regimes. From scratch, the model catastrophically fails (11.81%), converging to the majority class and triggering early stopping at epoch 13. With transfer learning, it reaches 66.43% (+54.62%). The depthwise separable convolutions in MobileNetV2 have fewer parameters per layer, making random initialization harder to optimize. The inverted residual structure depends on learned representations that benefit strongly from ImageNet pre-training.

ResNet50V2 performs *worse* with transfer learning (62.70%) than from scratch (76.35%). This counterintuitive result likely stems from domain mismatch: ImageNet features (natural images) may not transfer well to satellite imagery (overhead perspective, different textures). Freezing early layers preserves ImageNet-specific features that interfere with learning satellite-specific patterns. The lower learning rate (10^{-4}) used for transfer learning may also have been too conservative for this domain gap.

3.3.5 Per-Class F1-Score: EfficientNetB0 vs MobileNetV2 vs ResNet50V2

To compare the three architectures at a finer granularity, Table 72 reports per-class F1-scores for the single transfer-learning models. EfficientNetB0 achieves the best F1-score for *every single class*, demonstrating consistent superiority across both majority and minority categories.

[7] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

Table 72: Per-class F1-score comparison for single transfer-learning models (no ensemble).

Class	EfficientNetB0	MobileNetV2	ResNet50V2	Best
Cargo plane	**96.2%**	91.8%	90.9%	EfficientNetB0
Small car	**81.9%**	76.8%	73.7%	EfficientNetB0
Bus	**65.0%**	49.1%	52.2%	EfficientNetB0
Truck	**45.1%**	24.6%	26.2%	EfficientNetB0
Motorboat	**86.2%**	65.7%	70.0%	EfficientNetB0
Fishing vessel	**82.0%**	73.0%	58.3%	EfficientNetB0
Dump truck	**72.0%**	53.5%	50.1%	EfficientNetB0
Excavator	**89.5%**	67.7%	65.8%	EfficientNetB0
Building	**91.0%**	81.3%	72.4%	EfficientNetB0
Helipad	**97.0%**	69.0%	21.1%	EfficientNetB0
Storage tank	**91.2%**	70.8%	56.2%	EfficientNetB0
Shipping container	**74.5%**	60.3%	60.2%	EfficientNetB0
Pylon	**94.5%**	80.9%	70.6%	EfficientNetB0
Mean F1	**82.0%**	66.5%	59.1%	—

Two results are particularly striking. First, *Helipad* (111 training samples) achieves 97.0% F1-score with EfficientNetB0 but only 21.1% with ResNet50V2, a 76-point gap that underscores how architecture choice can matter more than training data quantity for rare classes. Second, *Truck* remains the hardest class across all three backbones (45.1% at best), confirming that the visual similarity between *Truck*, *Small car*, and *Bus* from an overhead perspective is an intrinsic dataset difficulty rather than an architectural limitation.

3.3.6 Per-Class Performance: Final EfficientNetB0 Ensemble

Table 73 presents the per-class metrics for the final model: the EfficientNetB0 ensemble from the submitted notebook. Comparing with Table 72, the ensemble improves F1-score on every class relative to the single EfficientNetB0 transfer-learning model—most notably *Truck* (+13.8%), *Fishing vessel* (+5.5%), and *Shipping container* (+11.5%).

Table 73: Per-class metrics for the final EfficientNetB0 ensemble (validation set).

Class	Recall	Precision	Specificity	F1-Score
Cargo plane	94.74%	100.00%	100.00%	97.30%
Small car	89.38%	82.44%	95.89%	85.77%
Bus	70.19%	70.72%	96.98%	70.46%
Truck	56.02%	62.00%	95.40%	58.86%
Motorboat	87.50%	88.05%	99.28%	87.77%
Fishing vessel	85.85%	89.22%	99.59%	87.50%
Dump truck	72.97%	79.41%	98.67%	76.06%
Excavator	94.92%	91.80%	99.63%	93.33%
Building	93.69%	93.00%	98.33%	93.35%
Helipad	94.12%	100.00%	100.00%	96.97%
Storage tank	93.64%	95.37%	99.61%	94.50%
Shipping container	88.65%	83.54%	98.45%	86.02%
Pylon	95.74%	95.74%	99.93%	95.74%
Mean	**85.95%**	**87.02%**	**98.60%**	**86.43%**

Key observations.

1. **Cargo plane** reaches 97.30% F1-score with perfect precision (zero false positives), benefiting from its distinctive large-scale appearance in satellite imagery.

2. **Helipad** (111 training samples) achieves 96.97% F1-score with perfect precision—a remarkable progression across the full experiment series: 0% (ffNN) → 82.35% (custom Inception CNN) → 97.0% (single EfficientNetB0 TL) → 96.97% (ensemble). The near-identical score between the single model and the ensemble for this class suggests that EfficientNetB0's features are already highly discriminative for helipads.

3. **Truck** remains the hardest class at 58.86% F1-score, still confused with *Small car* and *Bus* from overhead. However, this represents a substantial improvement over both the custom CNN baseline (35.74%) and the single EfficientNetB0 transfer-learning model (45.1%), demonstrating the added value of MixUp and ensemble averaging for difficult classes.

4. **Pylon** achieves balanced recall and precision (both 95.74%), fully resolving the precision gap observed in the custom Inception CNN (recall 91.49% vs precision 76.79%).

3.3.7 Confusion Matrix Analysis

Figure 28 shows the confusion matrix for the best single transfer-learning model (EfficientNetB0), which exhibits similar error patterns to the ensemble.

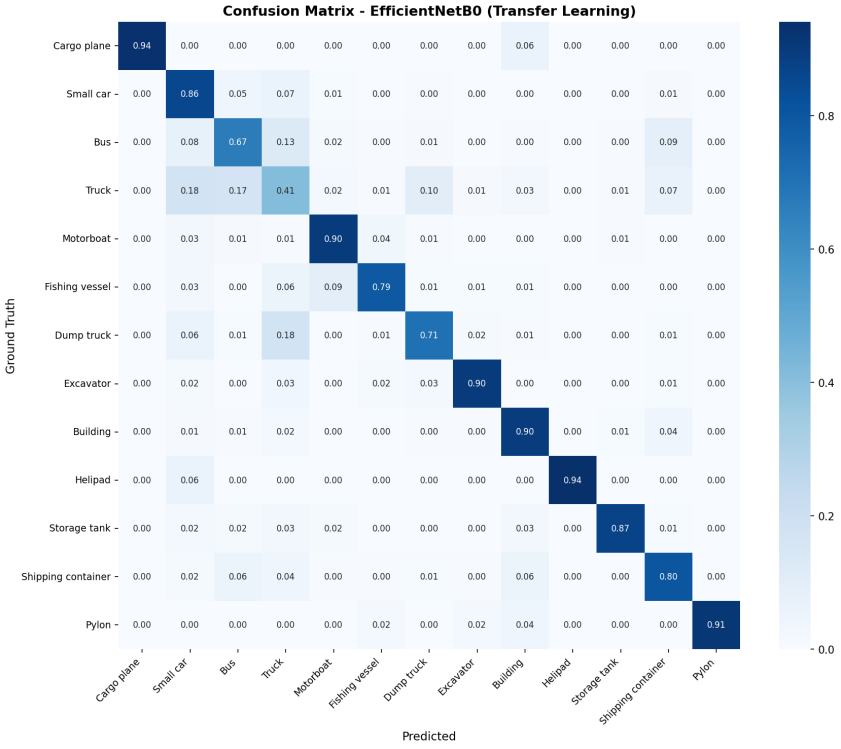


Figure 28: Normalized confusion matrix for EfficientNetB0 (transfer learning, single model). Values represent recall (row-wise normalization). The ensemble produces a similar pattern with reduced off-diagonal values.

3.3.8 Precision-Recall Analysis

Figure 29 shows precision-recall curves for selected classes using the best single transfer-learning model.

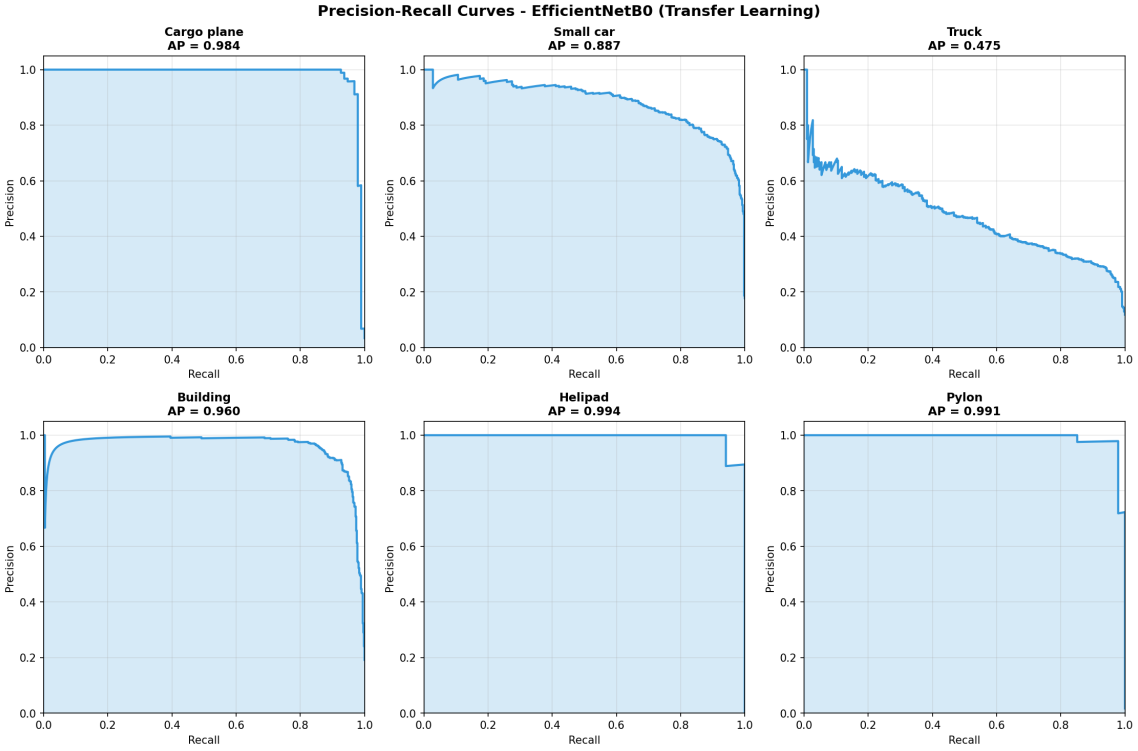


Figure 29: Precision-Recall curves for six representative classes using EfficientNetB0 (transfer learning, single model). AP = Average Precision.

3.3.9 Test Set Evaluation (Codabench)

The final ensemble model was submitted to the Codabench competition platform for evaluation on the held-out test set (2,365 images with private annotations). Table 74 reports the official test scores.

Table 74: Official test set results on Codabench (EfficientNetB0 ensemble).

Metric	Test Score
Mean Accuracy	82.66%
Mean Precision	81.98%
Mean Recall	81.59%

The test set performance (82.66%) is close to the validation performance (83.96%), with a drop of only 1.30 percentage points. This small gap indicates that the model generalizes well and that our validation protocol (stratified 85/15 split) is representative of the true data distribution. The consistency between validation and test metrics confirms that the training pipeline—MixUp, label smoothing, and ensemble averaging—effectively regularizes the model without overfitting to the validation set.

3.3.10 Broader Exploration Across Notebooks

Beyond the controlled experiments presented above, we conducted an extensive exploration of the design space across multiple notebooks (3-1 through 3-12). These experiments varied input resolutions (96×96 to 224×224), augmentation strategies, fine-tuning schedules (single-phase vs two-phase), loss functions (Categorical Crossentropy vs Focal Loss), optimizers, and inference methods (single model vs ensemble). Table 75 summarizes four representative configurations that illustrate the breadth of this exploration.

Table 75: Selected experiments from the broader notebook series. These results are not directly comparable to the controlled benchmark due to differences in input size, augmentation, and inference strategy.

Experiment	Input	Training Regime	Val Acc.	Mean F1
Custom CNN + residual blocks	128^2	From scratch + ensemble	74.86%	76.88%
EfficientNetB0, staged fine-tune	96^2	Transfer + 2 phases + ensemble	74.86%	76.88%
EfficientNetB2, mixed precision	224^2	Transfer + MixUp + ensemble	83.57%	86.99%
EfficientNetB0, screened pipeline	224^2	Transfer + MixUp + ensemble	83.96%	87.04%

These experiments are not strictly comparable to the controlled benchmark of Sections 3.3.2–3.3.4 because they differ in input resolution, augmentation strength, and use of ensemble inference. However, they consistently support three findings: (1) EfficientNet is a robust backbone family for satellite imagery; (2) training protocol (augmentation, learning rate schedule, ensemble) matters as much as architecture choice—the gap between a basic B0 setup (74.86%) and an optimized B0 pipeline (83.96%) exceeds the gap between different backbones; (3) rare classes such as *Helipad* consistently benefit from modern transfer-learning recipes, reaching 89–100% F1-score across experiments.

3.3.11 Summary and Key Findings

1. **The EfficientNetB0 ensemble is our best model**, achieving 83.96% validation accuracy (mean recall 85.95%, mean precision 87.02%, mean F1-score 86.43%) and **82.66% test accuracy** on Codabench (mean precision 81.98%, mean recall 81.59%). The pipeline combines backbone screening, transfer learning with partial freezing, MixUp augmentation, label smoothing, and 3-model ensemble averaging.

2. **Transfer learning benefits are architecture-dependent:**

   - MobileNetV2: Essential (+54.62%)—the model fails completely without pre-training
   - EfficientNetB0: Modest benefit (+1.14%) for a single model, but enables the full pipeline that reaches 83.96%
   - ResNet50V2: Detrimental (–13.66%)—from-scratch training is superior, likely due to domain mismatch between natural and satellite imagery

3. **The training recipe matters as much as the backbone.** Comparing a single EfficientNetB0 with transfer learning (78.73%, mean F1 82.0%) to the full pipeline (83.96%, mean F1 86.43%) shows a +5.23% accuracy and +4.4% F1 improvement attributable to MixUp, partial freezing, and ensemble averaging.

4. **EfficientNetB0 dominates per-class performance.** It achieves the best F1-score on all 13 classes in the single-model comparison (Table 72), with a mean F1 of 82.0% vs 66.5% (MobileNetV2) and 59.1% (ResNet50V2).

5. **Minority class performance scales with pipeline quality.** *Helipad* F1-score progresses from 0% (ffNN) → 90.32% (custom Inception CNN) → 97.0% (single EfficientNetB0 TL) → 96.97% (ensemble), demonstrating the cumulative impact of spatial feature extraction, class weighting, transfer learning, and ensemble inference.

6. **The validation–test gap is small** (83.96% → 82.66%), confirming good generalization and the absence of overfitting.

Table 76: Final comparison: best models from each approach.

Approach	Best Model	Val Accuracy	Mean F1
Custom ffNN	SGD + Dropout	43.28%	38.07%
Custom CNN	Inception-style	71.80%	74.37%
Popular (from scratch)	EfficientNetB0	77.60%	—
Popular (transfer learning)	EfficientNetB0	**83.96%**	**86.43%**

4 Phase 2: Object Detection

Object detection presents fundamentally different challenges from image recognition: the model must simultaneously localise instances via bounding boxes and classify them, across an image that may contain hundreds of densely packed objects of vastly different scales. The transition from the 13-class recognition task to the 4-class detection task therefore introduces new sources of difficulty that interact directly with the imbalance and density characteristics described in Section 1.

4.1 Experimental Setup

All detection experiments share the same dataset and evaluation protocol. The 6,845-image training set and 761-image validation set were produced by a 90/10 stratified split, where the stratification label encodes per-class presence (Building, Small car, Truck, Bus) and object-density bin. This ensures that rare class combinations are proportionally represented in both partitions.

The four detection classes exhibit a much more severe imbalance than the recognition classes. As reported in Table 77, Building and Small car together account for 96.5% of all 481,112 annotations, leaving Truck (2.2%) and Bus (1.3%) severely under-represented. A detector collapsing to predict only the two dominant classes can still achieve a low localisation loss while never firing for the rare categories. Addressing this imbalance was a recurring design concern across all experiments.

Performance is reported as Average Precision (AP) at IoU = 0.5 per class and mean AP (mAP), computed with the standard 11-point interpolated precision-recall area.

Due to computational constraints, we had to significantly reduce the input image size. This reduction decreased the model's ability to detect genuinely small objects and increased the tendency to classify them as background. As will be seen throughout this section, computational limitations were one of the main factors restricting overall detection performance.

Table 77: Class distribution in the object detection training set.

Class	Objects	Share (%)	Mean BBox Area (px^2)
Building	275,943	57.4	3,654
Small car	188,300	39.1	169
Truck	10,600	2.2	479
Bus	6,269	1.3	467
Total	**481,112**	**100**	2,179

To partially counteract the imbalance, all experiments applied *image-level oversampling*: training images containing at least one Bus or Truck annotation were duplicated in the training set before data

loading. The precise oversampling factor varied by experiment and is documented in each subsection. The validation set was never oversampled so that metrics remain representative of the true distribution.

4.2 Single-Stage Detectors

4.2.1 YOLOv8m

YOLOv8 is an anchor-free, single-stage detector that performs simultaneous object classification and bounding-box regression at three detection scales. The medium variant (`yolo_v8_m_backbone_coco`) used here loads a CSP-DarkNet backbone with a Path Aggregation Feature Pyramid Network (PANet) neck through the KerasCV API. The decoupled detection head applies separate branches for classification and localisation at three feature-map resolutions (P3, P4, P5), with a total of approximately 25.9 million trainable parameters.

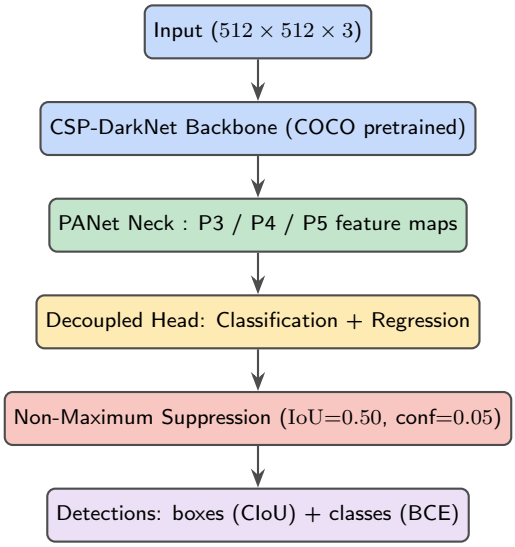


Figure 30: YOLOv8m architecture as used in this work. BCE: binary cross-entropy; CIoU: complete intersection-over-union loss.

YOLOv8 was selected as the primary single-stage detector because (1) the KerasCV preset provides a strong COCO-pretrained initialisation, and (2) the anchor-free design handles the variety of aspect ratios present in satellite imagery more flexibly than anchor-based approaches with fixed grids.

Experimental progression and training setup. Our first runs with YOLOv8m used the standard template as a starting point: images at 640×640 pixels, a batch size of 4 constrained by GPU memory on a dual-T4 node, uncapped bounding-box tensors (the dataset contains up to 1,029 annotated objects per tile), and a differentiated oversampling scheme in which images containing both Bus and Truck received up to three copies, images with only one rare class received two, and low-density scenes with a rare class received one additional copy. No geometric augmentation was applied beyond resizing. The learning rate schedule referenced a hardcoded total of 40 epochs as the cosine window while training was capped at 15, causing the decay to execute over an incorrect horizon.

Several issues became apparent from this first run. The uncapped tensor size of 1,029 boxes per image placed heavy pressure on GPU memory, forcing the batch size down to 4 and producing a large proportion of zero-padded entries per batch element that diluted the gradient signal. The differentiated oversampling introduced varying exposure multipliers across rare-class sub-categories, risking over-representation of specific image configurations rather than of the class as a whole. The fixed cosine reference meant the

learning rate did not reach its scheduled minimum within the actual training budget. Finally, the absence of geometric augmentation left the model without any orientation invariance, which is particularly important in overhead imagery where vehicles and buildings appear at arbitrary angles.

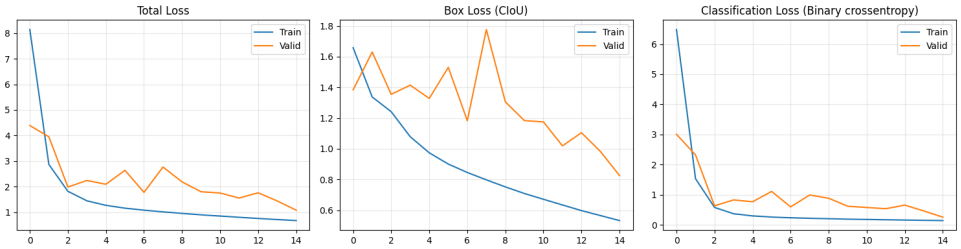
These observations guided a systematic set of refinements that define our final configuration. The input resolution was reduced from 640 to 512 pixels, which counterintuitively improved training: the smaller images allowed the batch size to increase from 4 to 12, providing three times as many gradient samples per step and substantially more stable parameter updates throughout. The maximum number of bounding boxes per image was capped at 300, a threshold that retains the vast majority of real images while eliminating the memory overhead and gradient noise produced by the long tail of near-empty padding. The oversampling logic was simplified to a single uniform rule: any image containing at least one Bus or at least one Truck annotation receives exactly one duplicate copy. This raised the training set from 6,845 to 10,103 images, resulting in 3,492 images with Bus and 4,648 with Truck, without artificially inflating any specific sub-category of rare-class scenes. A comprehensive geometric augmentation pipeline was introduced, applying random horizontal and vertical flips, random $k \times 90$ rotations with correctly transformed bounding boxes, and mild brightness and contrast jitter per element, all directly motivated by the rotational symmetry of satellite imagery. The image loading function was updated to perform per-band min-max normalisation, rescaling each spectral channel to $[0, 255]$ before casting to `uint8`, correcting the inconsistent pixel distributions that arose from raw GeoTIFF data types. The cosine decay reference was tied to the actual `EPOCHS` variable (20 epochs), ensuring the learning rate correctly reaches its minimum by the final epoch.

Table 78 summarises the key configuration differences between the two runs.

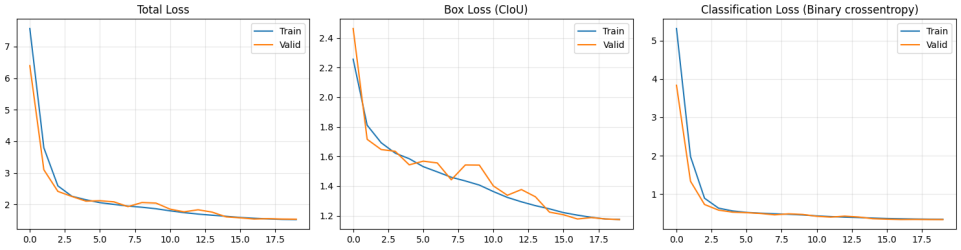
Table 78: Configuration comparison between the two YOLO experimental runs.

Parameter	**First run**	**Final configuration**
Input resolution	640×640	512×512
Batch size	4	12
Max boxes per image	1,029 (uncapped)	300 (capped)
Training images after oversamp.	≈6,845–8,000	10,103
Oversampling rule	Differentiated ($\times 2$ to $\times 3$)	Uniform ($\times 2$ for any Bus/Truck image)
Geometric augmentation	None	H/V flip, $k \times 90$ rotation
Classification loss	Focal (default)	Binary cross-entropy
LR schedule reference	Hardcoded 40 ep.	Dynamic (`EPOCHS`= 20)
Epochs run	15	20
Best val_loss	1.079	1.532
mAP@0.5	N/A (eval error)	**16.30%**

The two runs cannot be compared on the basis of absolute validation loss values alone: they differ in image resolution, batch size, maximum bounding-box count, and classification loss function, all of which affect the combined scale of the CIoU and classification components. The first run reached a best validation loss of 1.079 after 15 epochs, while the final configuration reached 1.532 after 20 epochs. The key observable difference lies in the training dynamics: the first run exhibited a classification loss starting near 10.6 at epoch 1, consistent with the unintended focal loss default operating on a severely imbalanced dataset from random initialisation, whereas the final configuration started substantially lower and decreased smoothly throughout, confirming that the corrected loss function and learning rate schedule were both operating as intended.



(a) First run (640 px, batch 4, 15 epochs). The classification loss starts near 10.6 at epoch 1, consistent with the default focal loss applied by the KerasCV version in use.



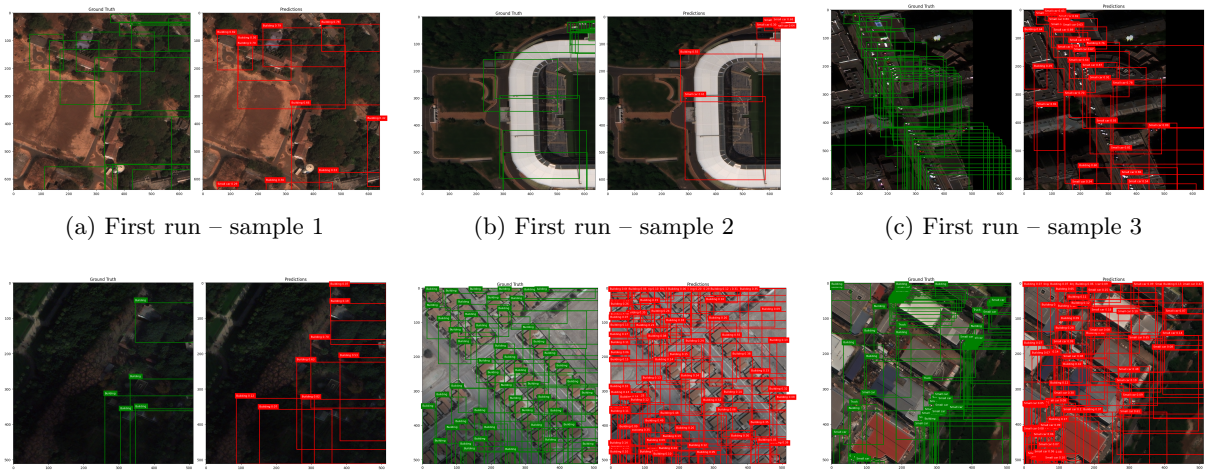
(b) Final configuration (512 px, batch 12, 20 epochs). Binary cross-entropy produces a lower starting classification loss and smoother convergence throughout.

Figure 31: Training dynamics across the two YOLO runs. The absolute loss values are not directly comparable due to differences in resolution, batch size, and classification loss function. The most informative contrast is the classification loss at epoch 1, which reflects the instability introduced by the unintended focal loss in the first run.

Validation results. Table 79 presents the per-class AP and aggregate metrics. A qualitative comparison between the first YOLO run and the best configuration is shown in Figure 32.

Table 79: Per-class detection metrics for the best YOLO configuration at IoU = 0.5.

Class	**Recall**	**Precision**	**AP@0.5**
Small car	32.57%	49.48%	25.52%
Bus	0.00%	0.00%	0.00%
Truck	0.09%	50.00%	0.09%
Building	47.34%	52.13%	39.60%
mAP	15.99%	30.32%	**16.30%**



(a) First run – sample 1 (b) First run – sample 2 (c) First run – sample 3

(d) Best configuration – sample 1 (e) Best configuration – sample 2 (f) Best configuration – sample 3

Figure 32: Qualitative comparison of YOLO predictions between the first run and the best configuration. Each panel shows ground-truth annotations on the left and model predictions on the right. The first run already captures some large *Building* and *Small car* instances, but remains visually inconsistent and produces many coarse detections. The best configuration yields more coherent predictions overall, especially on dominant classes, although *Bus* and *Truck* remain largely missed.

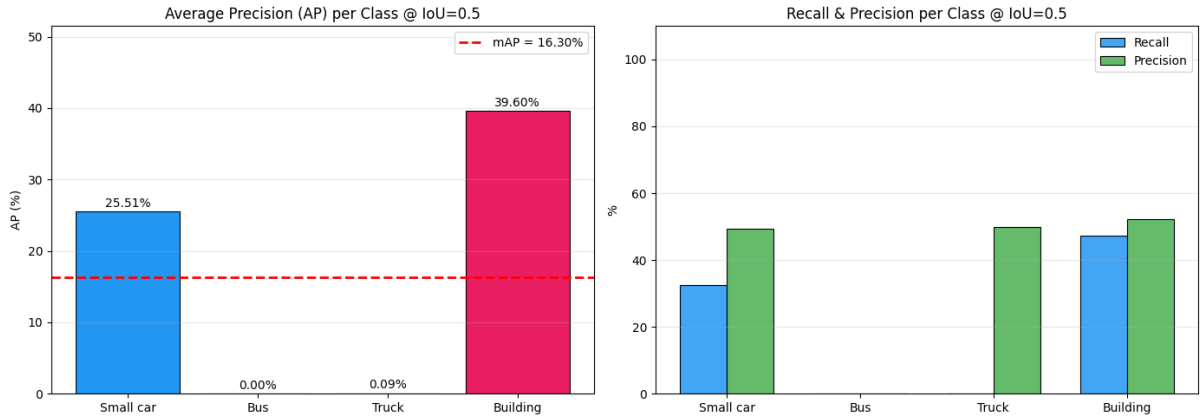


Figure 33: Per-class AP (left) and per-class Recall (right) for the best YOLO model at IoU = 0.5. The dashed red line marks the mAP of 16.30%. Bus and Truck achieve near-zero AP despite uniform oversampling.

Key observations.

1. **Building and Small car drive the overall mAP.** Building (AP= 39.60%) and Small car (AP= 25.52%) are the sole contributors to the mAP, reflecting their overwhelming prevalence.

2. **Bus and Truck remain undetected.** Despite the uniform ×2 oversampling, Bus achieves 0% AP and Truck only 0.09%. A single correct Truck detection accounts for its non-zero precision (50%): exactly two Truck detections were produced across the entire validation set, one of which was a true positive.

3. **High false-negative rate is the dominant failure mode.** The confusion matrix (Figure 34) reveals that 13,293 Small car ground-truth boxes, 511 Bus boxes, and 1,100 Truck boxes were predicted as background. This confirms that the class imbalance suppresses the classifier's willingness to fire for rare classes.

4. **Background confusion is symmetric.** A substantial fraction of background regions are wrongly classified as Small car (6,554) or Building (12,104), indicating that the confidence threshold for dominant classes is calibrated too loosely, while rare-class thresholds remain effectively unreachable.

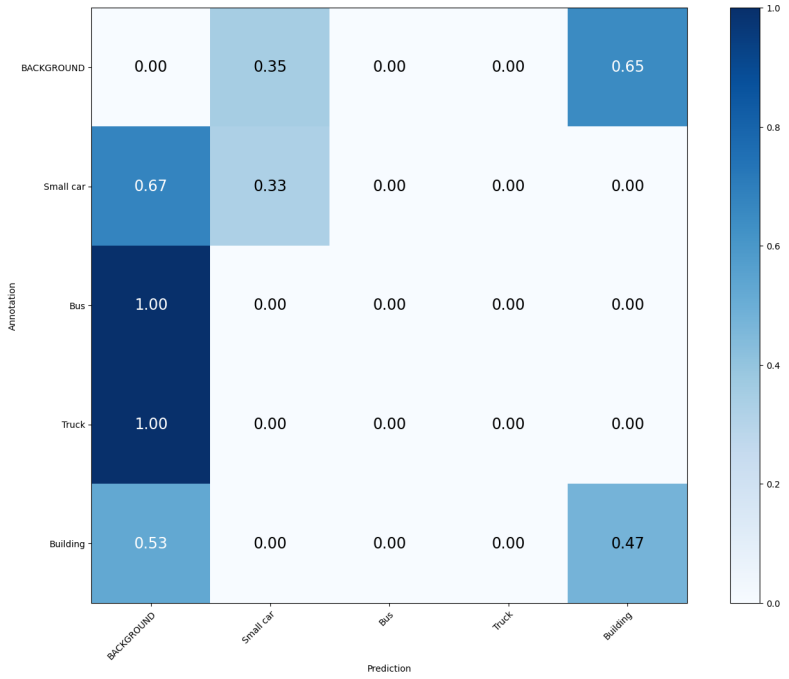


Figure 34: Confusion matrix for the best YOLO model on the validation set. BACKGROUND rows and columns dominate, confirming that the primary failure mode is excessive false negatives, particularly for Bus and Truck.

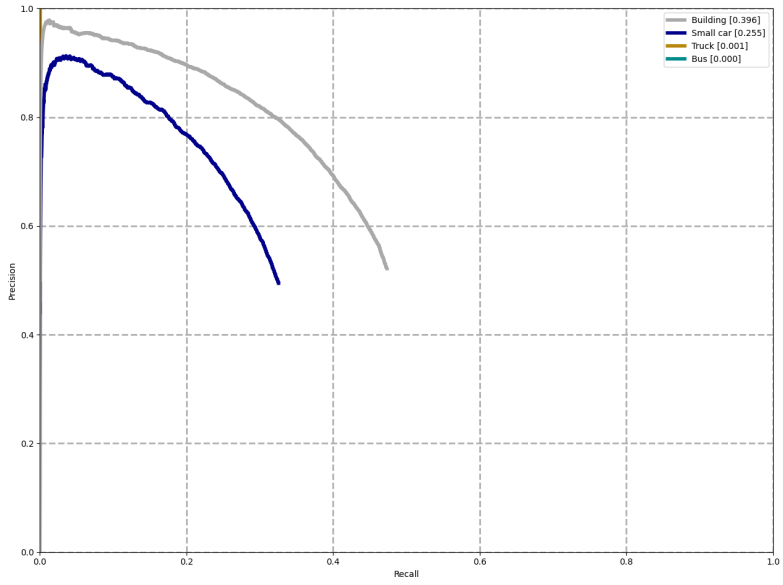


Figure 35: Precision-Recall curves for the best YOLO model. Building and Small car produce meaningful area under the curve; Bus and Truck collapse to near-zero.

4.2.2 RetinaNet with ResNet Backbone

RetinaNet[8] is an anchor-based single-stage detector combining a Feature Pyramid Network (FPN) neck with a shared classification and regression head applied at each pyramid level. Its defining contribution, the focal loss for classification, was particularly appealing for this task: by down-weighting the large number of easy background examples and focusing gradient signal on hard, misclassified instances, focal loss directly targets the difficulty posed by the extreme foreground-to-background ratio in satellite imagery. In addition, the multi-scale FPN architecture was expected to be well suited to the small-object problem identified in Section 1.2: by maintaining high-resolution feature maps at the finest pyramid level (P3), RetinaNet can in principle detect objects as small as 8 pixels across, which matches the size range of most Small car and Bus instances in the xView detection subset.

Two ResNet backbones pre-trained on ImageNet were explored across two experiments, both interfaced through `keras_cv.models.RetinaNet`.

Experiment 1: ResNet101 backbone. The first configuration used a ResNet101 backbone, batch size 6, and a two-phase training strategy. Phase 1 froze the first 40% of backbone layers for up to 10 epochs with cosine learning rate decay from 10^{-4} to 10^{-6}; Phase 2 unfroze *all* backbone layers for up to 15 additional epochs with a lower rate ($10^{-5} \rightarrow 10^{-7}$). Images were provided at 640×640 with mixed float16 precision. Bus/Truck-containing images were duplicated in the training set, a $\times 2$ multiplier. The NMS IoU threshold was set to 0.35. Phase 1 converged normally, saving a checkpoint at epoch 1 with val_loss= 1.056. Phase 2, however, produced a catastrophic validation loss spike to approximately 801 at epoch 2. Early stopping restored the Phase 1 checkpoint, which was used for evaluation and yielded the results in Table 80.

Table 80: RetinaNet – Experiment 1 (ResNet101) validation metrics at IoU = 0.5.

Class	Recall	Precision	AP@0.5
Small car	0.48%	19.44%	0.12%
Bus	0.00%	0.00%	0.00%
Truck	0.00%	0.00%	0.00%
Building	4.10%	37.19%	1.80%
mAP	0.92%	31.33%	**0.48%**

Results are extremely weak: only Building and Small car produce any detections, both at negligible recall. The model is essentially non-functional, consistent with Phase 2 instability preventing fine-tuning from contributing any improvement over the one-epoch Phase 1 checkpoint.

Experiment 2: ResNet50 backbone with refined setup. The second experiment addressed the instability observed in Experiment 1 through four targeted modifications:

1. **Lighter backbone (ResNet101 → ResNet50).** The smaller backbone reduced per-step memory usage and enabled the batch size to double from 6 to 12, stabilising gradient estimates.

2. **Increased oversampling ($\times 3$ total).** Bus/Truck-containing images now receive two additional copies (`bus_truck_anns × 2`), expanding the training set to 13,389 images.

[8] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

3. **BatchNorm frozen in Phase 2.** All `BatchNormalization` layers are kept in inference mode during full-backbone fine-tuning. The rationale was to prevent the running statistics from shifting abruptly under the high-variance gradient signal produced by dense small-object detection.

4. **More conservative Phase 2 learning rate ($10^{-6} \rightarrow 10^{-8}$, vs $10^{-5} \rightarrow 10^{-7}$ in Experiment 1).** The one-order-of-magnitude reduction was intended to reduce the risk of overshooting during fine-tuning.

Table 81: Comparison of the two RetinaNet configurations.

Parameter	**Experiment 1**	**Experiment 2**
Backbone	ResNet101	ResNet50
Batch size	6	12
Training images	10,117 (inferred $\times 2$)	13,389 ($\times 3$)
NMS IoU threshold	0.35	0.50
Phase-2 BN frozen	No	Yes
Phase-2 base LR	10^{-5}	10^{-6}
Phase-2 val collapse	val$\approx$801	val_class_loss$\approx$45
Phase-1 best val_loss	1.056	1.249
Evaluation	mAP= 0.48%	Not available (`TypeError`)

Phase 1 of Experiment 2 completed epoch 1 successfully (val_loss= 1.249). However, Phase 2 again produced a large validation spike (val_classification_loss$\approx$ 45 at epoch 2), indicating that the instability persists even with BatchNorm frozen. Although the collapse is less catastrophic than in Experiment 1 (45 vs 801), it was still sufficient to trigger early stopping and restore the Phase 1 checkpoint.

Why RetinaNet failed to deliver on its theoretical potential. Despite its architectural suitability for small-object detection via FPN, RetinaNet failed to produce usable results in both configurations. The recurrent Phase 2 instability is the most direct cause. We hypothesise that the combination of mixed-float16 precision, the focal loss applied to densely packed small objects, and the abrupt full-backbone unfreeze on a pre-trained ImageNet model produces a gradient landscape that is too volatile for the second fine-tuning phase. The partial mitigation achieved by freezing BatchNorm and lowering the learning rate in Experiment 2 (collapse value dropping from 801 to 45) suggests that this diagnosis is directionally correct, but the fix was insufficient.

Beyond the instability issue, a deeper limitation is that the FPN's small-object detection capability only materialises if Phase 2 fine-tuning completes successfully: it is precisely during full-backbone fine-tuning that the FPN levels learn to specialise for the small scale and overhead perspective of xView objects. Since this phase was disrupted in both experiments, RetinaNet never had the opportunity to exploit the very advantages that motivated its selection. Stabilising Phase 2, for instance by using float32 precision throughout, applying gradient clipping per layer, or adopting a gradual layer-by-layer unfreeze schedule, would be necessary for a fair evaluation, but was not feasible within the computational we had.

4.3 Two-Stage Detectors: Faster R-CNN

Faster R-CNN[9] is a two-stage detector in which a Region Proposal Network (RPN) generates class-agnostic candidate regions from a shared backbone feature map, and a second stage classifies and refines

[9]Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. 2015, pp. 91–99.

each proposal via ROI Align and dedicated classification/regression heads. The implementation used here is drawn from the TensorFlow Object Detection API with a COCO-pretrained ResNet50 backbone at 640×640 (`faster_rcnn_resnet50_v1_640x640_coco17`), exposing 222 trainable variables.

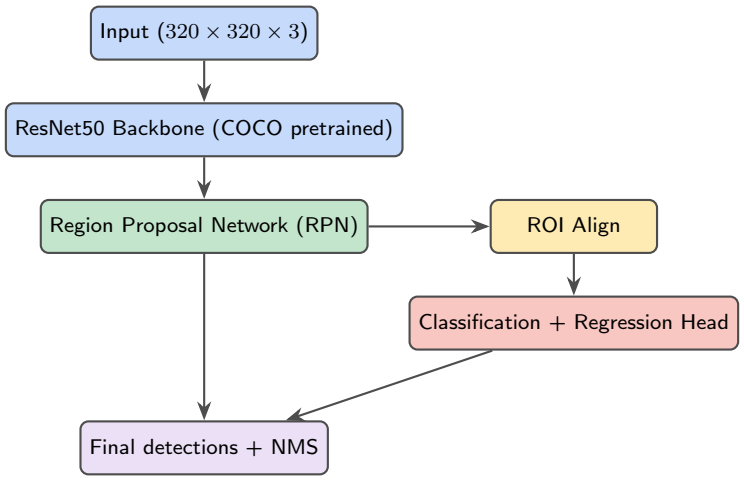


Figure 36: Faster R-CNN architecture. The RPN produces class-agnostic region proposals, ROI Align pools each proposal to a fixed spatial size before the second-stage head performs final classification and box regression.

The two-stage design is theoretically better suited than single-stage detectors to scenarios with extreme background-to-foreground ratios, because the RPN can learn an objectness score before the second stage performs fine-grained classification. However, this advantage comes at substantially higher computational cost, which became the dominant practical constraint in this work.

Experimental setup and resource constraints. The experiment was conducted on a Kaggle P100 GPU (16 GB VRAM). The high memory footprint of the two-stage pipeline required downscaling the inference resolution to 320×320 (from the native 640×640 design), reducing the total pixel count by a factor of four. This significantly degrades the model's ability to detect small objects, the very objects that dominate this dataset. Two additional constraints limited training completeness: (a) the training loop was capped at 400 steps per epoch to respect the daily GPU quota, whereas the full epoch contains approximately 2,530 batches, meaning each epoch processed only 15.8% of available data, (b) training was limited to 6 epochs total (2 in phase 1, 4 in phase 2), so the model saw fewer than one full pass over the training data in total.

Oversampling duplicated Bus/Truck-containing images once ($\times 2$, 3,272 added copies, total 10,117 images). Batch size was 4. The Adam optimiser used cosine learning rate decay from 10^{-4} with a 1-epoch warm-up.

Training dynamics and results. Training was stable but slow to converge. As shown in Table 82, the validation loss decreased monotonically from 3.255 at epoch 1 to 3.077 at epoch 6, indicating continued improvement throughout the allotted budget without reaching convergence. Evaluation on the full 761-image validation set produced the results in Table 83.

Table 82: Faster R-CNN training progression (6 epochs, 400 steps/epoch, batch 4).

Epoch	Phase	Train Loss	Val Loss	LR
1	1	3.597	3.255	1.00×10^{-4}
2	1	3.272	3.175	1.00×10^{-4}
3	2	3.240	3.119	9.05×10^{-5}
4	2	3.214	3.179	6.58×10^{-5}
5	2	3.140	3.110	3.52×10^{-5}
6	2	3.118	**3.077**	1.05×10^{-5}

Table 83: Faster R-CNN validation metrics at IoU = 0.5. The near-zero mAP reflects severe undertraining imposed by computational constraints.

Class	Recall	Precision	AP@0.5
Small car	0.00%	0.00%	0.000%
Bus	0.00%	0.00%	0.000%
Truck	4.50%	0.25%	0.014%
Building	0.00%	0.00%	0.000%
mAP	0.90%	60.05%	**0.004%**

The results are essentially non-functional. The model produces detections for only a negligible fraction of Truck instances and no correct detections for any other class. The confusion matrix confirms that the overwhelming majority of predictions are background: 18,448 Building instances, 17,501 Small car instances, and 997 Truck instances are entirely missed.

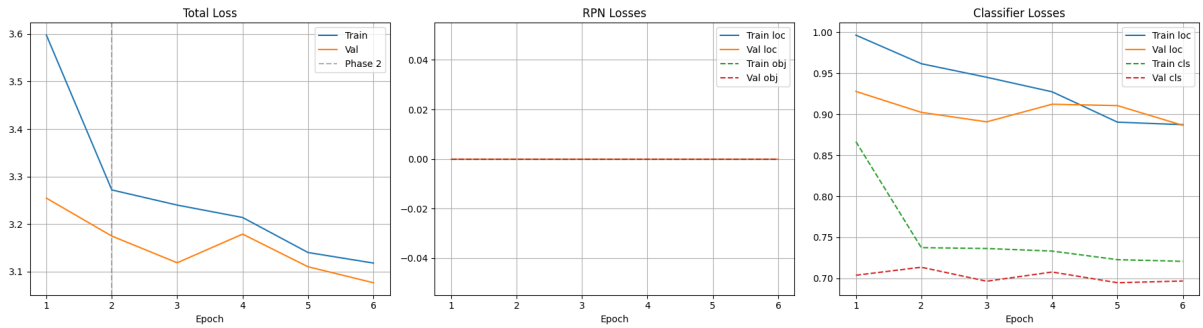


Figure 37: Training dynamics of Faster R-CNN over the 6 training epochs. Overall, the total validation loss decreases only gradually, indicating that the detector had not yet converged when training stopped.

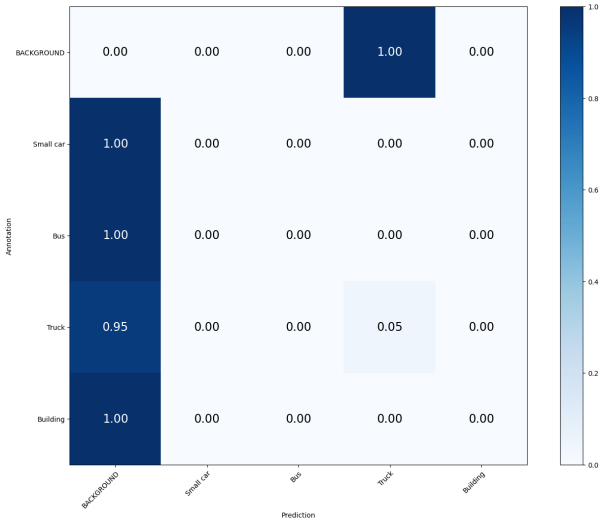


Figure 38: Confusion matrix for Faster R-CNN on the validation set. Most ground-truth objects are assigned to background, confirming that the dominant failure mode is a very high false-negative rate across all classes.

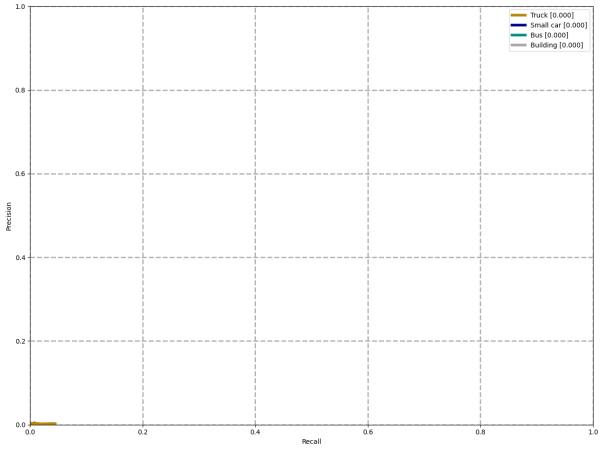


Figure 39: Precision–Recall curves for Faster R-CNN at IoU = 0.5. All curves collapse near the origin, confirming the near-zero AP obtained for every class except a negligible non-zero value for Truck.

Key observations.

1. **Computational constraints made Faster R-CNN non-viable.** Processing fewer than 16% of training data per epoch over 6 epochs is insufficient for any modern detector to learn meaningful representations, regardless of its theoretical capabilities.

2. **Resolution downscaling is particularly detrimental here.** Reducing from 640×640 to 320×320 halves the feature resolution at the finest pyramid level, impeding detection of Small car instances (average bounding-box area: 169 px^2) that are already near the limit of detectability.

3. **Convergence was not reached.** The validation loss of 3.077 after 6 epochs suggests the model has barely begun to adapt to the satellite domain. A competitive mAP would require 50–100 epochs at full data coverage.

4.4 Comparative Analysis

Table 84 consolidates all detection experiments.

Table 84: Final comparison of all object detection experiments. N/A indicates that evaluation did not complete.

Model	Type	Epochs	Steps seen	mAP@0.5	Best AP
YOLOv8m (old)	Single-stage	15	∼25,600	N/A (eval error)	—
YOLOv8m (best)	Single-stage	20	∼16,800	**16.30%**	Building 39.60%
RetinaNet ResNet101 (Exp. 1)	Single-stage	10+unstable	∼25,000	0.48%	Building 1.80%
RetinaNet ResNet50 (Exp. 2)	Single-stage	10+unstable	∼11,000	N/A (error)	—
Faster R-CNN ResNet50	Two-stage	6	2,400	0.004%	Truck 0.014%

Why YOLOv8m significantly outperformed the other models. The performance gap between YOLOv8m (mAP= 16.30%) and the other architectures is substantial and attributable to three compounding factors:

1. **Pre-training alignment.** YOLOv8m is initialised from COCO weights trained with the exact same loss functions and head architecture used during fine-tuning, ensuring direct knowledge transfer to the detection head. The RetinaNet and Faster R-CNN models also use pre-trained backbones, but their heads are randomly initialised and therefore require more epochs to reach a comparable starting point.

2. **Training completeness.** YOLOv8m completed 20 full epochs over 10,103 images ($\approx$ 840 steps/epoch, $\approx$ 16,800 gradient steps total). The Faster R-CNN executed only 2,400 steps. The RetinaNet ran full epochs in phase 1 but its fine-tuning phase was disrupted by numerical instability in both configurations.

3. **More effective class-imbalance handling.** The simplified uniform oversampling in the best YOLO configuration, combined with binary cross-entropy on the multi-label classification head, produced a more stable training signal for rare classes than the focal loss that was silently applied in the old YOLO or the numerically unstable focal loss in RetinaNet phase 2. Although Bus and Truck AP remain near zero, Truck AP increased from 0% (old YOLO, which did not produce a usable evaluation) to 0.09% (best YOLO), and the confusion matrix shows two correct Truck detections, indicating that the best YOLO did learn some Truck-discriminative features, however weak.

Even the best result (mAP= 16.30%) should be considered limited. Several structural factors constrain all approaches on this dataset:

1. **Class imbalance persists despite oversampling.** With 188,300 Small car vs. 6,269 Bus annotations, image-level duplication provides only a coarse correction. More targeted strategies, class-balanced sampling within each batch, hard-negative mining, copy-paste augmentation, or mosaic-based data synthesis, would be needed to expose the model more systematically to rare-class examples.

2. **Small-object difficulty.** 61.2% of all objects have bounding-box area below 32×32 pixels. Competitive detection at this scale requires high-resolution feature maps, careful anchor or stride design, and dedicated small-object augmentation, none of which were applied here.

3. **Computational budget.** All experiments were constrained by Kaggle GPU quotas and session time limits. A competitive mAP on this dataset likely requires 50–100 training epochs with more aggressive data augmentation (mosaic, scale jitter, copy-paste) and per-class NMS threshold tuning at inference.

Summary. Overall, YOLOv8m was the only detector that achieved meaningful object detection performance on this benchmark, reaching a validation mAP@0.5 of 16.30%. This result was driven almost entirely by the two dominant classes, *Building* (AP= 39.60%) and *Small car* (AP= 25.52%), and it was the version submitted to the Codabench competition platform. In contrast, *Bus* and *Truck* remained almost unresolved across all experiments, confirming that class imbalance is the main unsolved difficulty of this task despite the use of image-level oversampling.

The other detector families remained limited for different reasons. RetinaNet suffered from recurrent instability during phase 2 fine-tuning: both the ResNet101 and ResNet50 versions exhibited a severe validation-loss spike when the full backbone was unfrozen under mixed-float16 training, preventing the second phase from contributing useful improvements. Faster R-CNN, on the other hand, was primarily constrained by the available computational budget. With only 2,400 gradient steps at reduced resolution, it never approached convergence and remained effectively non-functional on the validation set.

The superiority of the best YOLO configuration therefore comes less from a single architectural breakthrough than from a coherent set of practical refinements. The combination of uniform oversampling, reduced input resolution, larger batch size, improved augmentation, and a corrected learning-rate schedule produced a training setup that was both more stable and more computationally tractable. Within the limits of the available resources, this balance between optimisation stability and imbalance handling was the main reason why YOLOv8m clearly outperformed the other approaches.

5 Conclusion

This report presented a comprehensive experimental study of image recognition and object detection on the xView satellite dataset, progressively exploring feedforward neural networks, custom convolutional architectures, transfer learning pipelines, and modern object detection frameworks. The work followed a deliberate bottom-up methodology: starting from the simplest possible models and systematically introducing architectural complexity, regularization, and preprocessing refinements, allowing each design decision to be evaluated in isolation and its contribution precisely quantified.

For the feedforward neural network phase (44 experiments), the most critical finding was that **dimensionality reduction is the single most impactful design choice** for dense architectures operating on raw pixel inputs. Without it, any hidden layer narrower than the input caused catastrophic collapse to majority-class prediction, a failure observed consistently across Experiments 2–4, 6–8, 12, and 13. The initial best single-model result (Experiment 28, 59.47% mean accuracy) was achieved by downsampling inputs to 56×56 via bicubic interpolation, reducing the flattened feature count from 150,528 to 9,408, combined with SELU activation, Nadam optimization, and focal loss. Subsequent experiments pushed this ceiling further through three complementary strategies: residual connections that enabled effective training of networks with 10 or more dense layers (Experiment 37), bounding box cropping combined with minority class oversampling that improved recall on previously intractable classes such as Helipad (Experiments 33–35), and the fusion of raw pixel features with hand-crafted HOG shape descriptors in a dual-branch ensemble architecture (Experiment 44, 66.54% mean accuracy). The regularization analysis revealed that dropout was more effective than L2 weight decay for this imbalanced dataset, and that BatchNormalization, while stabilizing training dynamics, required complementary regularization to prevent overfitting. These experiments established a firm ceiling of approximately 67% for architectures that treat pixels independently, motivating the transition to spatial feature extraction.

Custom convolutional networks broke through this ceiling decisively. The Inception-style architecture achieved 71.80% validation accuracy with only 4.5 million parameters, outperforming both the VGG-style

model (65.93%, 15.1M parameters) and the ResNet-style model, which catastrophically failed at 13.12% due to an unstable interaction between focal loss, class weights, and the Nadam optimizer. The Inception architecture's success confirmed that **multi-scale feature extraction is particularly well suited to satellite imagery**, where objects range from 111-pixel helipads to 3,594-pixel buildings, and where the overhead perspective produces distinctive scale-dependent visual signatures. Crucially, class weights passed through the data generator as sample weights resolved the chronic failure on minority classes: *Helipad* recall improved from 0% (all ffNN experiments) to 82.35%, demonstrating that architectural innovation and class-imbalance mitigation are complementary rather than competing concerns.

Transfer learning with EfficientNetB0 produced the best overall recognition model. The final pipeline, combining backbone screening via grid search, partial freezing (all layers frozen except the last backbone layer), MixUp augmentation ($\alpha = 0.2$), label smoothing (0.1), and a 3-model mean ensemble, achieved **83.96% validation accuracy** and **82.66% test accuracy** on the Codabench platform, with a mean F1-score of 86.43%. The narrow validation-to-test gap of 1.30 percentage points confirmed robust generalization. Notably, the controlled comparison across three backbones revealed that **the benefit of transfer learning is architecture-dependent**: MobileNetV2 required pre-training to function at all (+54.62%), EfficientNetB0 gained modestly (+1.14% for a single model), and ResNet50V2 performed worse with transfer learning (−13.66%), likely due to a domain mismatch between natural and overhead satellite imagery. This finding cautions against treating transfer learning as a universal improvement and underscores the importance of backbone screening before committing computational resources.

The progression of *Helipad* performance across the entire experiment series, from 0% (ffNN) → 82.35% (custom Inception CNN) → 97.0% (single EfficientNetB0) → 96.97% (ensemble), encapsulates the cumulative impact of spatial feature extraction, class weighting, transfer learning, and ensemble inference on minority class recognition. It demonstrates that with only 111 training samples, a well-designed pipeline can achieve near-perfect classification, provided each component of the training recipe is carefully integrated.

The detection phase exposed a substantially harder problem. The 4-class subset exhibits a 44:1 imbalance ratio between *Building* (275,943 annotations) and *Bus* (6,269), compounded by extreme object density (up to 1,029 objects per 640×640 crop) and a small-object problem (61.2% of instances below 32×32 pixels). Under these conditions, YOLOv8m was the only detector that achieved meaningful results, reaching a **validation mAP@0.5 of 16.30%**, driven almost entirely by *Building* (AP = 39.60%) and *Small car* (AP = 25.52%). *Bus* and *Truck* remained effectively undetected across all architectures despite uniform image-level oversampling.

RetinaNet, despite its theoretical suitability for small-object detection via the Feature Pyramid Network and focal loss, suffered from recurrent numerical instability during the second fine-tuning phase under mixed-float16 precision, preventing it from leveraging its multi-scale architecture. Faster R-CNN was constrained by computational limitations to fewer than 2,400 gradient steps at reduced 320×320 resolution, never approaching convergence. These failures were primarily practical rather than architectural: both models possess the representational capacity to handle this task but require significantly more training budget, numerical stability, and augmentation sophistication than was available within the Kaggle GPU quotas used for this project.

The success of the best YOLO configuration relative to the other detectors was attributable not to a single architectural advantage but to a coherent set of practical refinements: reducing input resolution from 640 to 512 pixels (which tripled the effective batch size from 4 to 12), capping the maximum bounding boxes per image at 300, applying uniform oversampling, introducing geometric augmentation suited to rotational symmetry in satellite imagery, and correcting the learning rate schedule. This confirms the systematic alignment of batch size, augmentation, loss function, and learning rate schedule

can matter as much as model architecture in resource-constrained settings.

Three themes emerge consistently across both phases of this work:

Class imbalance is the dominant challenge in satellite imagery tasks. Whether in recognition (32:1 ratio between *Building* and *Helipad*) or detection (44:1 between *Building* and *Bus*), imbalance dictates model behavior. Simple techniques such as class weights and image-level oversampling provide partial relief, but more advanced strategies, copy-paste augmentation, mosaic synthesis, class-balanced batch sampling, or hard-negative mining would be needed to resolve rare-class detection.

The training recipe is as important as the architecture. The gap between a basic EfficientNetB0 setup (74.86% validation accuracy) and the optimized pipeline (83.96%) exceeds the gap between different backbone architectures. Similarly, the best YOLO result was driven by augmentation, batch size, and loss function corrections rather than by architectural changes. This finding is consistent with recent trends in the deep learning literature emphasizing reproducibility and training discipline over novel architectures.

Computational constraints shape experimental outcomes in non-trivial ways. The Faster R-CNN and RetinaNet experiments illustrate how our GPU memory limitations, session time quotas, and numerical precision interact to prevent models from reaching their theoretical potential. In educational and resource-limited settings, choosing architectures that converge efficiently within the available budget, such as EfficientNetB0 for recognition and YOLOv8m for detection is a pragmatic consideration that can outweigh theoretical architectural comparisons.

This project demonstrates that satellite image analysis remains for us a challenging domain for deep learning, requiring careful attention to class imbalance, small-object detection, multi-scale feature extraction, and training stability. The best recognition model (EfficientNetB0 ensemble, 82.66% test accuracy) and the best detection model (YOLOv8m, 16.30% mAP@0.5) represent meaningful results given the constraints, but also highlight the substantial gap between recognition and detection performance on overhead imagery. Closing this gap will require not only more computational resources but also more targeted augmentation strategies, improved loss formulations for rare classes, and inference-time calibration techniques, directions that we leave for future work.

References

[1] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016).

[2] Darius Lam et al. "xView: Objects in Context in Overhead Imagery". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018.

[3] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

[4] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[5] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. 2015, pp. 91–99.

[6] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[7] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015).

[8] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.