Introduction to Artificial Intelligence Final Project Report

Svetlana Berelekhis
Bar Ilan University

February 28, 2026

Abstract

This report presents a supervised machine learning study across three tasks. In Part I, we applied Linear Regression, Polynomial Regression (degrees 1–10), and K-Nearest Neighbours (KNN) to the Auto MPG dataset to predict car fuel efficiency. KNN with k=2 achieved the best validation $R^2 = 0.923$ (RMSE = 2.39 MPG), followed closely by Polynomial Regression (degree 2, $R^2 = 0.902$). We also compared three gradient-descent variants: SGD and Mini-Batch GD converged within 20 epochs, while Batch GD was still descending at epoch 200. In Part II, three classical classifiers (Logistic Regression, SVM, KNN) were evaluated on CIFAR-10 images; the best result was Logistic Regression at 41.6% test accuracy. In Part III, a fully-connected neural network trained with PyTorch reached 56.9% test accuracy, a 15-point improvement over the best classical method. The main finding is that model choice must match the data: polynomial and KNN models work well on structured tabular data, while images need architectures that understand spatial structure.

Contents

1 Introduction

This project explores supervised machine learning through three complementary tasks:

1. **Regression** — predicting car fuel efficiency (MPG) from vehicle characteristics using the Auto MPG dataset.
2. **Classical Classification** — recognising objects in images using the CIFAR-10 dataset and classical ML models.
3. **Deep Learning** — building a neural network image classifier with PyTorch.

Each task is used to study a different set of machine learning principles: bias–variance trade-offs and optimisation behaviour (Part I), the limitations of classical methods on high-dimensional data (Part II), and regularisation and architecture design for neural networks (Part III).

2 Part I: Regression on Auto MPG Dataset

2.1 Dataset and Preprocessing

The Auto MPG dataset contains records for 398 cars manufactured between 1970 and 1982. The prediction target is fuel efficiency measured in miles per gallon (MPG). The following seven features were used: *cylinders*, *displacement*, *horsepower*, *weight*, *acceleration*, *model_year*, and *origin*.

Preprocessing steps:

1. **Missing values:** The dataset contains 6 missing values in the `horsepower` column. These were imputed using the feature median (93.5 HP), which is robust to outliers and avoids introducing distributional bias.
2. **Train/validation/test split:** Data was divided into 70% training (277 samples), 15% validation (59 samples), and 15% test (60 samples). The test set was held out and never used during model selection or hyperparameter tuning.
3. **Feature standardisation:** All features were standardised to zero mean and unit variance using `StandardScaler` fitted exclusively on the training set, preventing data leakage.

All seven features were retained. The correlation analysis below confirms that each contributes signal: four features (weight, displacement, cylinders, horsepower) show strong negative correlation with MPG, while model_year and origin are positively correlated and provide complementary, relatively independent information.

2.2 Exploratory Data Analysis

Before building any models, we conducted an exploratory analysis to understand the structure of the data and motivate modelling choices.

Target variable (MPG). Figure 1 shows the distribution of MPG. The mean is 23.51 MPG and the median 23.00 MPG, indicating a mild right skew. The Shapiro–Wilk normality test rejects strict normality ($p < 0.05$), visible in the Q–Q plot tails. The right

skew and the curvature visible in the Q–Q plot suggest some non-linearity in the data that a purely linear model may not fully capture.

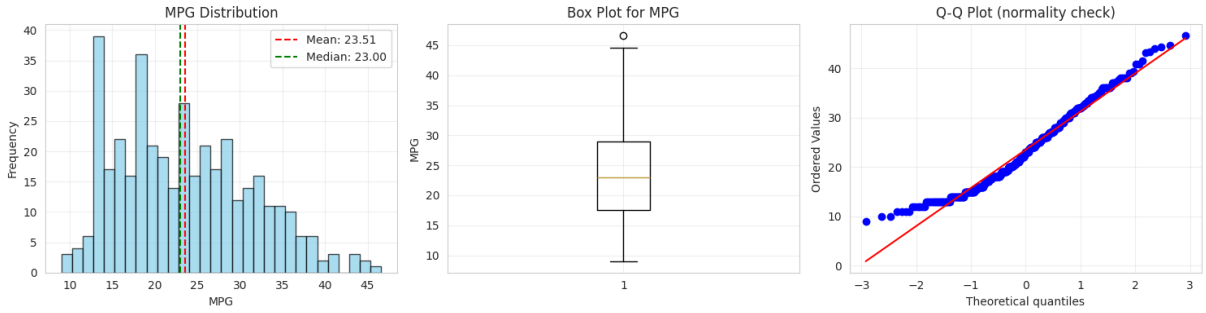

Figure 1: MPG distribution. *Left:* histogram with mean (red dashed, 23.51 MPG) and median (green dashed, 23.00 MPG). *Centre:* box plot showing the interquartile range and one upper outlier near 46 MPG. *Right:* Q–Q plot confirming mild departure from normality, particularly in the lower tail.

Feature–target relationships. Figure 2 shows scatter plots of MPG against each feature. Weight, displacement, horsepower, and cylinders exhibit clear negative trends; model_year shows a positive trend; acceleration is weakly positive; and origin (encoded 1/2/3) shows a modest upward slope. Importantly, the relationships with weight and horsepower display visible curvature, motivating the use of polynomial and non-parametric models in addition to the linear baseline.

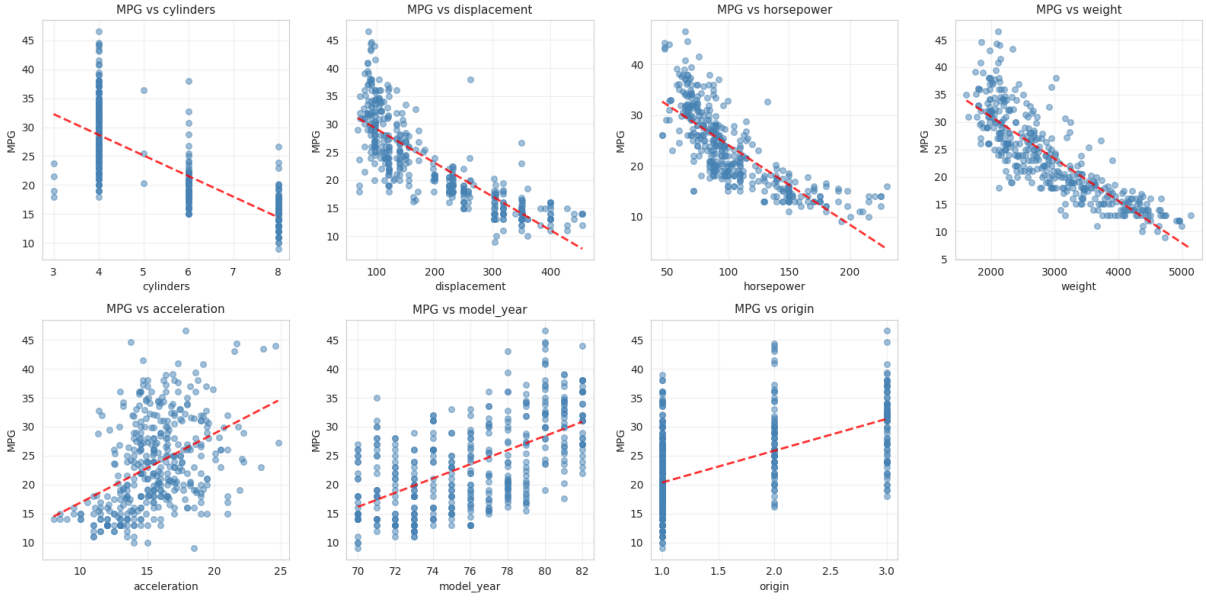

Figure 2: Scatter plots of MPG against each feature with linear trend lines. Curved patterns in MPG vs. weight and horsepower indicate that a linear model will underfit these relationships.

Feature correlations. Figure 3 displays the full correlation matrix. Weight has the strongest negative correlation with MPG ($r = -0.83$), followed by displacement ($r = -0.80$), cylinders ($r = -0.78$), and horsepower ($r = -0.78$). These four engine-related features are highly inter-correlated (e.g., displacement–weight: $r = 0.93$), indicat-

ing multicollinearity that is expected in a dataset spanning the same physical constraints. Model_year ($r = 0.58$) and origin ($r = 0.56$) provide relatively independent information.

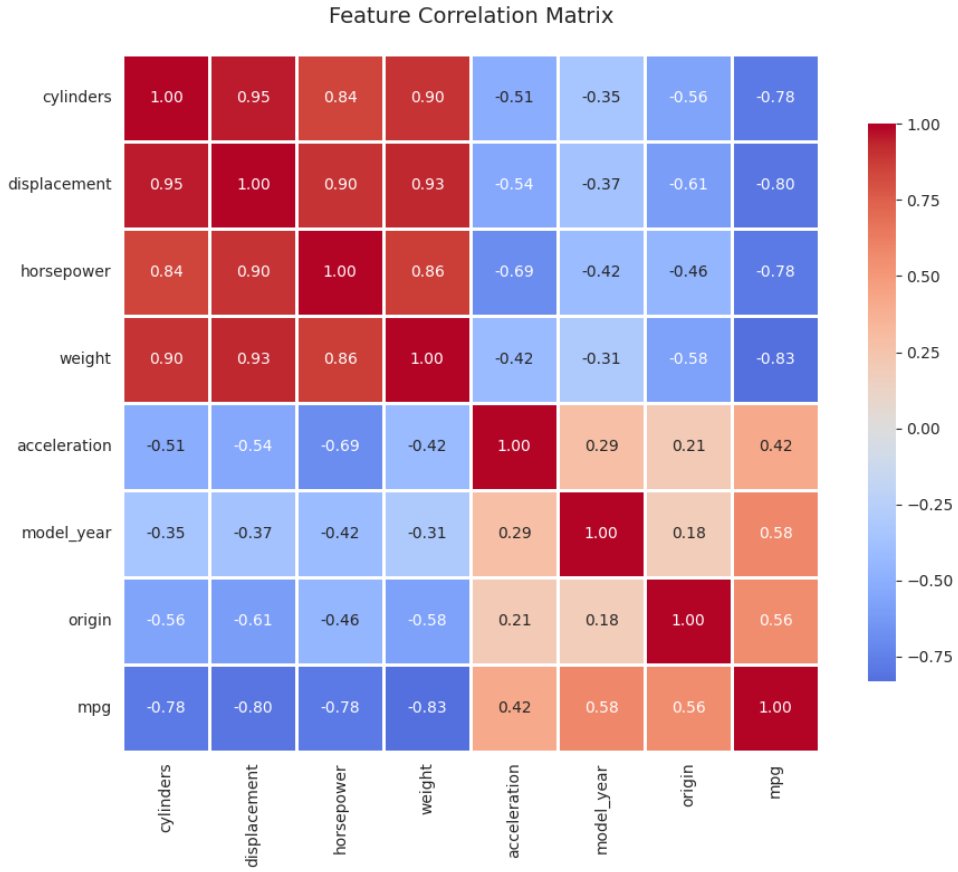

Figure 3: Feature correlation matrix. The top-left block (cylinders, displacement, horsepower, weight) is strongly inter-correlated and all four are negatively correlated with MPG. Model year and origin supply complementary positive signal.

Regional and temporal trends. Figure 4 shows that Japanese and European cars are systematically more fuel-efficient than American cars. The time-trend plot reveals a clear upward trajectory from approximately 17.5 MPG in 1970 to over 30 MPG in 1980–82, likely driven by regulatory and economic responses to the oil crises of the 1970s. Both origin and model_year therefore carry genuine predictive content beyond the engine-related features.

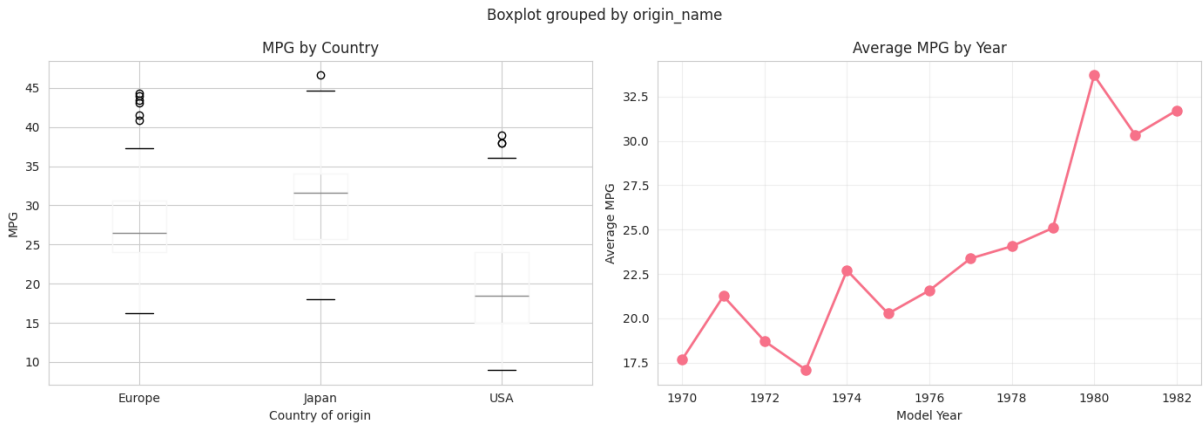

Figure 4: *Left:* MPG distribution by country of origin (box plot). Japanese cars have the highest median MPG, followed by European and American cars. *Right:* average MPG by model year, showing a consistent upward trend from 1970 to 1982.

2.3 Model 1: Linear Regression Baseline

Linear regression fits a hyperplane $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$ by minimising the mean squared error (MSE) over the training set. It serves as the simplest reasonable baseline and defines the performance floor for more flexible models.

Set	**MSE**	**RMSE (MPG)**	**MAE (MPG)**	R^2
Training	10.88	3.30	2.43	0.816
Validation	13.97	3.74	2.90	0.812
Test	9.40	3.07	2.38	0.832

Table 1: Linear regression performance across all three splits. The close agreement between training and validation R^2 (0.816 vs 0.812) indicates no overfitting.

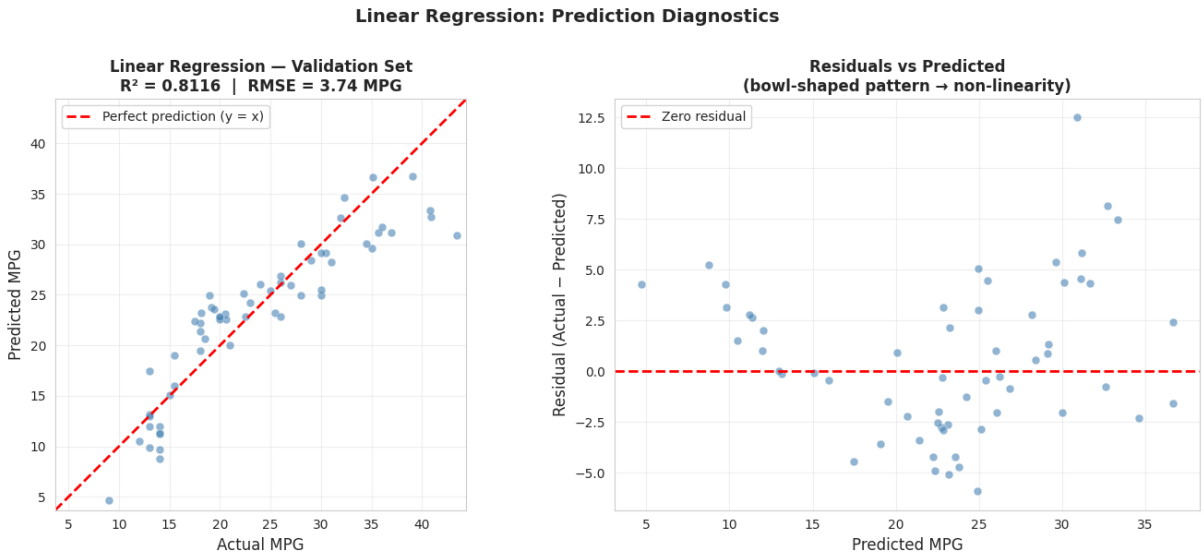

Figure 5: Linear Regression prediction diagnostics on the validation set. *Left:* predicted vs. actual MPG; points close to the dashed line indicate accurate predictions. *Right:* residuals vs. predicted values, showing a bowl-shaped pattern — the model over-predicts low-MPG cars and under-predicts high-MPG cars, confirming that the linearity assumption is an approximation for this dataset.

Assessment of the linearity assumption. An R^2 of 0.83 on the test set indicates that linear regression captures a substantial share of MPG variance. However, the scatter plots in Figure 2 show clear non-linear patterns—particularly for weight and horsepower—that a linear model cannot represent. The residuals exhibit a systematic bowl-shaped pattern (over-predicting for very low and very high MPG cars), confirming that linearity is an approximation. A prediction error of approximately 3 MPG represents an acceptable rough estimate but suggests that more flexible models are warranted.

2.4 Model 2: Polynomial Regression and Bias–Variance Analysis

Polynomial regression extends the linear model by appending interaction and higher-order terms (e.g., weight^2, weight$\times$horsepower) as additional features via `PolynomialFeatures`, and then fitting a linear model in this expanded space. This allows the decision surface to curve while retaining the closed-form solution of ordinary least squares.

We trained models for polynomial degrees 1 through 10. Figure 6 shows the resulting training and validation MSE and R^2 as functions of degree.

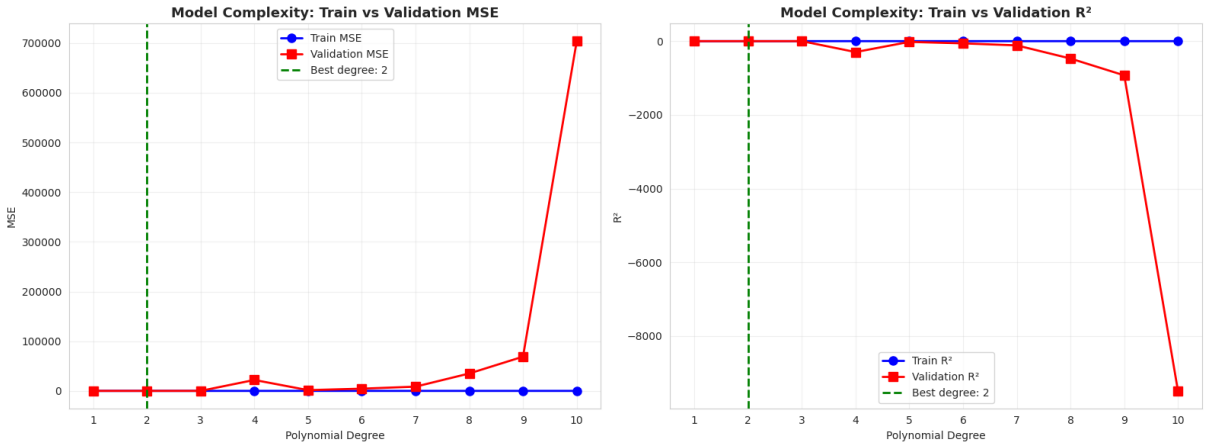

Figure 6: Model complexity curves for polynomial regression. *Left:* train (blue) and validation (red) MSE vs. polynomial degree. *Right:* corresponding R^2. The green dashed line marks the selected degree 2. Validation MSE explodes for degrees ≥ 4, with validation R^2 becoming strongly negative, indicating that the model performs worse than simply predicting the mean.

Bias–variance analysis:

- **Degree 1 (underfitting):** Equivalent to linear regression. High bias — too simple to capture the curved feature–target relationships visible in the data.
- **Degree 2 (optimal balance):** Validation $R^2 = 0.9017$, RMSE $= 2.70$ MPG. The model captures the main non-linear interactions without memorising training noise. Training and validation errors remain close.
- **Degree 3:** Validation R^2 drops to approximately 0.86. The gap between training and validation error begins to widen, signalling the onset of overfitting.
- **Degrees 4–10 (severe overfitting):** Training MSE approaches zero while validation MSE reaches 7×10^5 at degree 10, and R^2 becomes deeply negative. The model memorises the training set exactly but generalises catastrophically — a textbook illustration of high variance.

Selected model: degree 2, based on the lowest validation MSE. This choice improves validation R^2 by 10.7 percentage points relative to the linear baseline.

2.5 Model 3: K-Nearest Neighbours Regression

KNN regression is a non-parametric, instance-based method that predicts the MPG of a query point by averaging the target values of its k nearest neighbours in the standardised training set (Euclidean distance). It makes no assumptions about the functional form of the relationship.

We evaluated $k \in \{1, 2, 3, 5, 7, 10, 15, 20, 25, 30, 40, 50\}$, selecting the best k by validation MSE. Figure 7 shows the error curves.

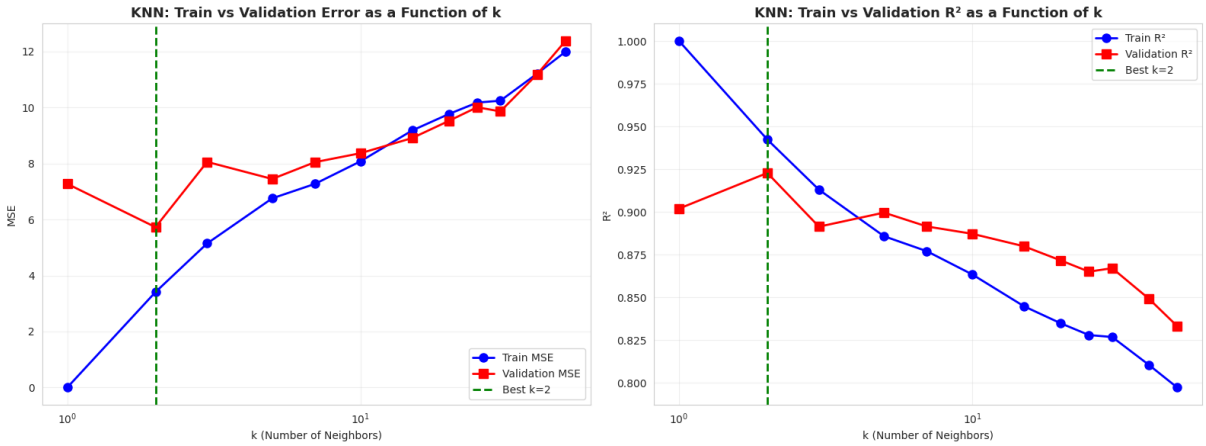

Figure 7: KNN regression bias–variance curves (log-scale k axis). *Left:* train and validation MSE. *Right:* train and validation R^2. At $k = 1$, train MSE $= 0$ (pure memorisation). $k = 2$ minimises validation MSE. Large k produces smooth, high-bias predictions that approach the global mean.

k	**Train MSE**	**Val MSE**	**Val R^2**
1	0.00	7.27	0.902
2	**3.41**	**5.73**	**0.923**
5	6.76	8.09	0.889
10	8.09	8.76	0.880
50	11.99	12.38	0.833

Table 2: KNN regression performance at selected values of k. The best validation $R^2 = 0.923$ is achieved at $k = 2$.

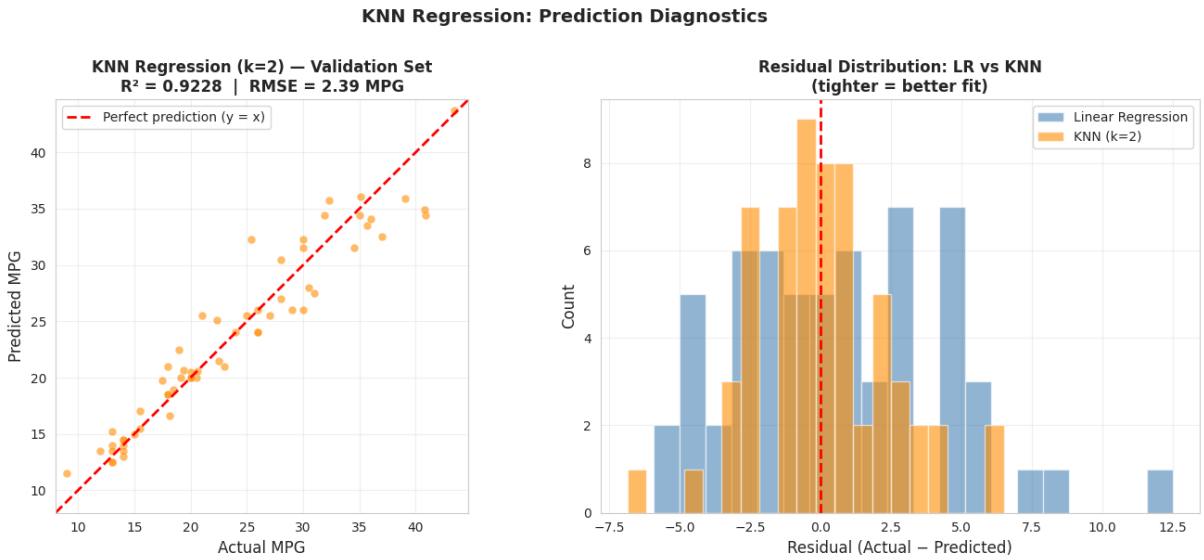

Figure 8: KNN regression ($k = 2$) prediction diagnostics on the validation set. *Left:* predicted vs. actual MPG; the tighter clustering around the diagonal compared to Figure 5 reflects the higher $R^2 = 0.9228$. *Right:* residual distributions for Linear Regression and KNN — KNN residuals are more concentrated near zero, confirming its superior fit on this dataset.

Bias–variance interpretation. At $k = 1$ the model exhibits zero training error (it retrieves each point exactly) but high variance: the prediction surface is jagged, and any noise or outlier in the training set is directly reflected in predictions. Increasing k smooths the surface (lower variance) at the cost of increased bias. The validation curve confirms $k = 2$ as the optimal trade-off for this dataset.

When does KNN outperform parametric models? KNN outperforms polynomial regression here because the MPG response surface contains local non-linear structure that low-degree polynomials approximate imperfectly on a global basis. KNN adapts locally without any global parametric assumption. However, KNN is slower at prediction time ($O(n)$ distance computations per query), sensitive to irrelevant features, and cannot extrapolate beyond the convex hull of the training data.

2.6 Optimization Behavior

To analyse gradient-based optimisation, we implemented three gradient descent variants from scratch in NumPy and applied them to the linear regression objective with a bias term.

1. **Batch Gradient Descent (BGD)** ($\eta = 0.01$): Uses the exact gradient computed over the full training set at each step. Convergence is smooth and monotone but slow — after 200 epochs the model has not yet converged (Train MSE ≈ 12.0).

2. **Stochastic Gradient Descent (SGD)** ($\eta = 0.001$): Updates weights after each randomly drawn sample. Converges within $\sim$15 epochs to Train MSE ≈ 10.9, but individual updates are noisy.

3. **Mini-Batch Gradient Descent (MBGD)** ($\eta = 0.01$, batch size $= 32$): Balances the stability of BGD with the fast descent of SGD, converging within $\sim$20 epochs to Train MSE ≈ 11.0.

Figure 9: Optimisation behavior: train loss (*left*) and validation loss (*right*) as a function of epoch for BGD (blue, solid), SGD (red), and Mini-Batch GD (green). SGD and MBGD converge rapidly within 20 epochs; BGD is still descending at epoch 200 owing to its small fixed step size.

The validation curves confirm the same pattern: SGD and MBGD plateau near Val MSE ≈ 14 within 20 epochs, while BGD is still converging at epoch 200. The slow BGD convergence is a consequence of the fixed learning rate — in practice, a larger η or a line-search would accelerate it substantially. For deployed systems, MBGD is preferred: it is nearly as fast as SGD but produces a more stable loss trajectory and is GPU-parallelisable.

2.7 Model Comparison and Final Evaluation

Table 3 summarises all three regression models on the validation set; model selection was performed here. The final test-set evaluation was run once, after the best model was chosen. Figure 11 provides a visual comparison.

Model	**Val R^2**	**Val RMSE (MPG)**	**Test R^2**	**Test RMSE (MPG)**
Linear Regression	0.812	3.74	0.832	3.07
Polynomial (deg= 2)	0.902	2.70	0.879	2.61
KNN ($k = 2$)	**0.923**	**2.39**	**0.893**	**2.45**

Table 3: Final model comparison. All hyperparameters were selected on the validation set; test metrics are reported for the best model and its closest competitor only.

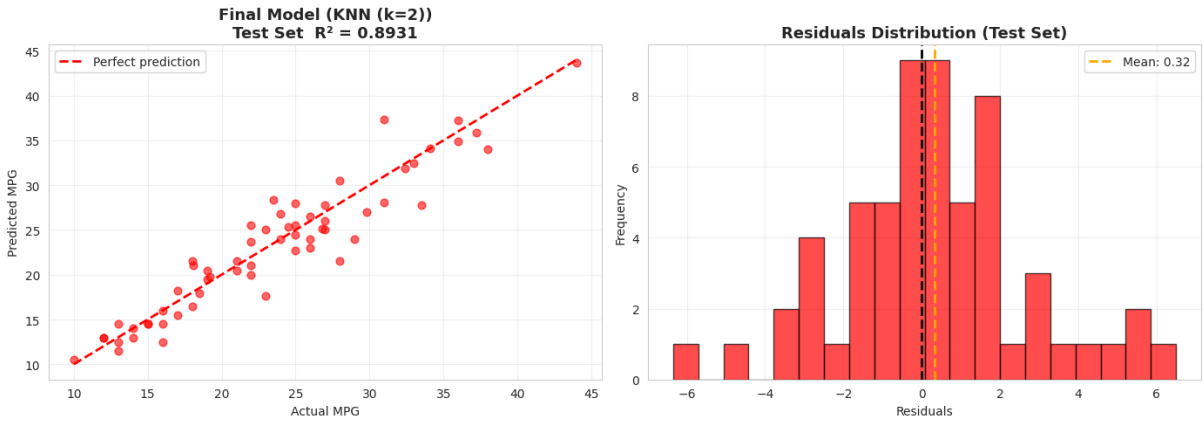

Figure 10: Final model (KNN, $k = 2$) evaluated on the held-out test set ($R^2 = 0.893$, RMSE = 2.45 MPG). *Left:* predicted vs. actual MPG — points follow the diagonal closely, confirming good generalisation. *Right:* residual distribution centred near zero (mean = 0.32 MPG), with no systematic bias. The test set was used exactly once, after all model selection decisions were finalised on the validation set.

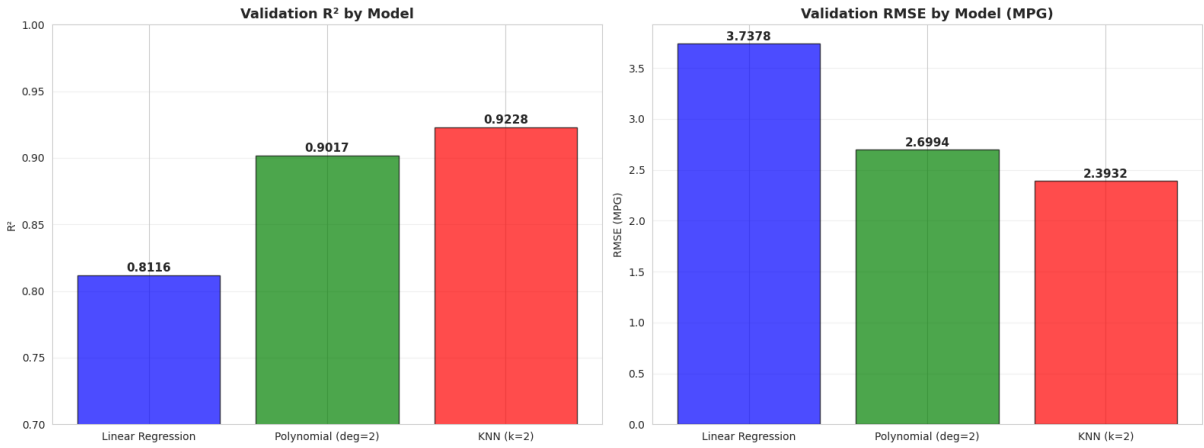

Figure 11: Validation performance comparison. *Left:* R^2 by model (higher is better). *Right:* RMSE in MPG (lower is better). KNN ($k = 2$) achieves the best performance on both metrics, with Polynomial (deg= 2) a close second.

Comparison across evaluation dimensions:

Predictive performance. KNN ($k = 2$) achieves the highest validation $R^2 = 0.923$ and lowest RMSE = 2.39 MPG, followed by Polynomial (deg= 2, $R^2 = 0.902$). Both improve substantially over the linear baseline ($\Delta R^2 \approx 0.11$). The test-set results are consistent, confirming that both models generalise well.

Bias–variance characteristics. Linear regression suffers from high bias. Polynomial degree 2 achieves the best parametric bias–variance balance; higher degrees exhibit catastrophic variance. KNN at $k = 2$ captures local non-linearity effectively, though its reliance on $k = 2$ neighbours makes it sensitive to noisy training points.

Interpretability. Linear regression is maximally interpretable: each coefficient quantifies the change in MPG per unit change in a standardised feature. Polynomial degree 2 retains meaningful coefficients for squared and interaction terms. KNN provides no global model equation and offers no interpretability.

Practical considerations. Polynomial regression requires only a matrix–vector product at inference time. KNN requires storing all 277 training samples and computing 277 distances per query—prohibitive at large scale.

Recommended model for deployment: Polynomial Regression (degree 2). Although KNN achieves a marginally higher validation R^2 (0.923 vs 0.902), the 2.1-point difference corresponds to less than 0.3 MPG in RMSE. Polynomial regression is preferred because it is interpretable, computationally cheap at inference time, and requires no training data at deployment. Its test $R^2 = 0.879$ confirms strong generalisation. KNN would be preferable in an offline, purely predictive setting where interpretability is not required.

3 Part II: Classical Classification on CIFAR-10

3.1 Dataset and Preprocessing

CIFAR-10 contains 60,000 colour images of size 32×32 pixels across 10 classes: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship,* and *truck*. We used the full dataset: 50,000 images for training/validation and 10,000 for testing. Each image is flattened into a $32 \times 32 \times 3 = 3{,}072$-dimensional feature vector.

Preprocessing steps:

1. Pixel values were rescaled from $[0, 255]$ to $[0, 1]$.
2. All features were standardised to zero mean and unit variance using `StandardScaler` fitted on the training set only.
3. The 50,000 training images were split 80/20 into a training set (40,000 images) and a validation set (10,000 images) using stratified sampling to preserve class balance.
4. The test set (10,000 images) was kept locked and used only once, after the final model was selected.

3.2 Model 1: Logistic Regression (Softmax)

Logistic regression with a softmax output and ℓ_2 regularisation (`solver=lbfgs`, `multi_class=multinomi` was used as the first baseline.

Hyperparameter search. The regularisation strength C was searched over $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$ using a random subsample of 10,000 training examples to keep runtime manageable. Validation accuracy was evaluated on the full 10,000-example validation set. Figure 12 shows that accuracy peaks sharply at $C = 0.001$ and drops on both sides, confirming that strong regularisation is essential in this high-dimensional pixel space.

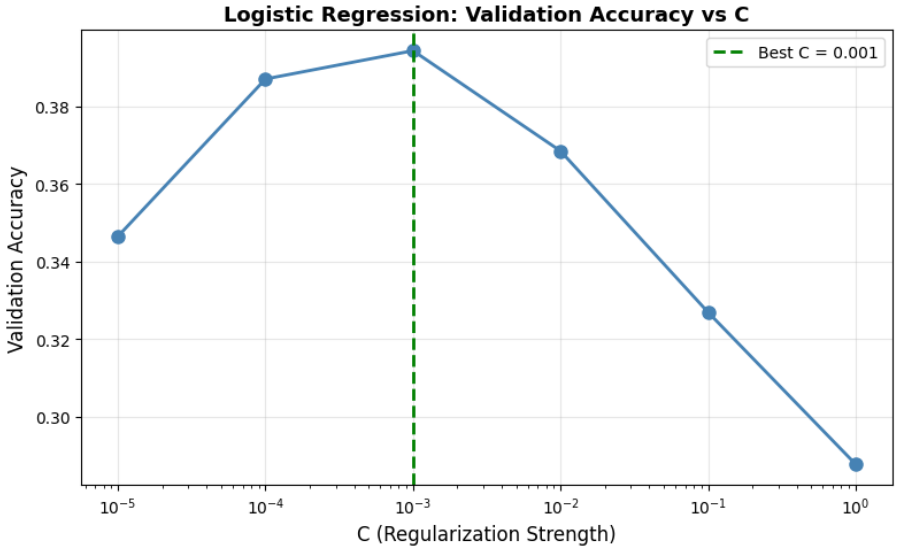

Figure 12: Logistic Regression: validation accuracy as a function of the regularisation strength C. Best value: $C = 0.001$.

The final model was retrained on all 40,000 training examples with $C = 0.001$.

Set	**Accuracy**	**Train–Val Gap**
Training	45.65%	4.07 pp
Validation	41.58%	

Table 4: Logistic Regression final results ($C = 0.001$).

3.3 Model 2: Linear SVM

A linear SVM (`LinearSVC`) with ℓ_2 regularisation was trained next.

Hyperparameter search. C was searched over $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ on the same 10,000-example subsample. As shown in Figure 13, the optimal value is $C = 10^{-5}$ — even smaller than for logistic regression. This reflects the hinge loss's different sensitivity to regularisation: the SVM margin objective requires a tighter penalty to prevent the high-dimensional weight vector from growing too large.

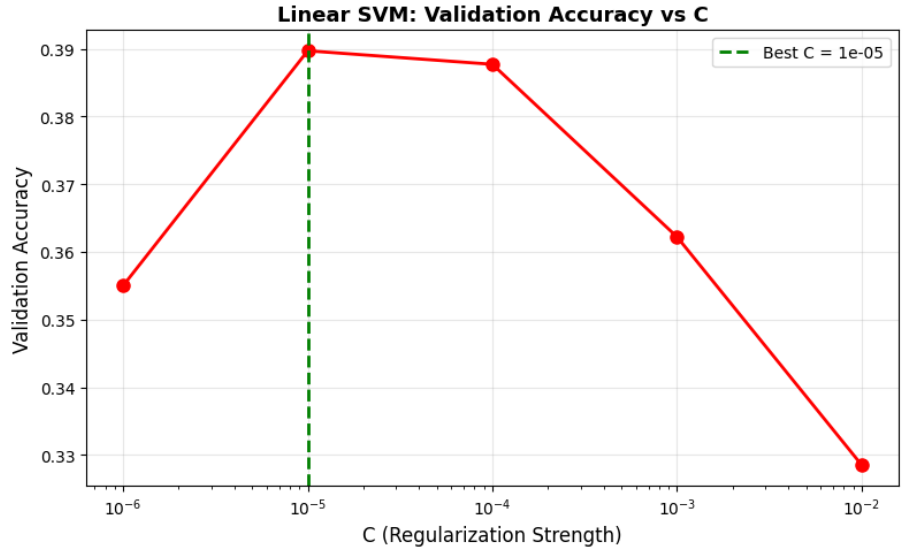

Figure 13: Linear SVM: validation accuracy as a function of C. Best value: $C = 10^{-5}$.

Set	Accuracy	Train–Val Gap
Training	42.73%	1.65 pp
Validation	41.08%	

Table 5: Linear SVM final results ($C = 10^{-5}$).

3.4 Model 3: K-Nearest Neighbours with PCA

KNN is a non-parametric, instance-based classifier that assigns the class label by majority vote among the k nearest training examples (Euclidean distance). Applying KNN directly to 3,072-dimensional raw pixel vectors is both slow and unreliable, because in high dimensions all pairwise distances become nearly equal — the so-called *curse of dimensionality*. We therefore first reduce the feature space using PCA.

Step 1: Choosing the number of PCA components

We searched $n \in \{20, 50, 100, 150, 200\}$ using a subsample of 5,000 training examples and a fixed $k = 10$. Figure 14 shows the cumulative explained variance curve: 20 components retain 73.6% of the total variance. The validation accuracy (not shown separately) decreased monotonically with more components on this subsample, because larger projections add noise relative to signal when training data is limited. The selected value is $\mathbf{n = 20}$ components.

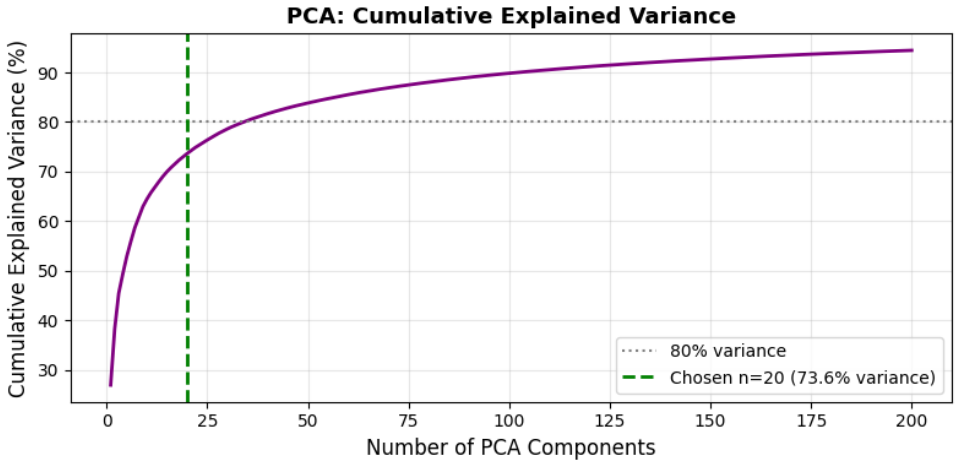

Figure 14: PCA cumulative explained variance. The chosen $n = 20$ components (green dashed line) retain 73.6% of the total variance.

Step 2: Choosing the number of neighbours k

With PCA-20 features applied to the full 40,000-example training set, k was searched over $\{1, 3, 5, 7, 10, 15, 20, 30, 50\}$. Figure 15 shows both training and validation accuracy as a function of k, illustrating the bias–variance tradeoff clearly:

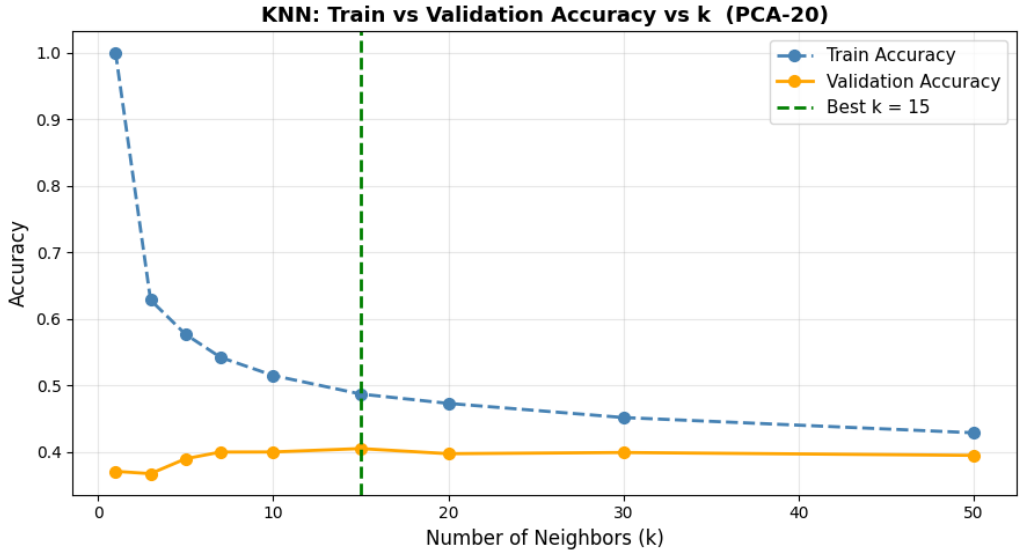

Figure 15: KNN: training accuracy (blue dashed) and validation accuracy (orange solid) as a function of k on PCA-20 features. At $k = 1$, training accuracy is 100% (pure memorisation). Validation accuracy peaks at $k = 15$.

- $k = 1$: Train accuracy = 100%, Val = 37.05%. Every training point is its own nearest neighbour, so the model memorises the data perfectly but generalises poorly (maximum variance).
- $k = 15$ **(best)**: Train = 48.67%, Val = 40.47%. Best trade-off between bias and variance.

- $k = 50$: Train = 42.84%, Val = 39.45%. The neighbourhood is so large that it averages over classes, losing local structure (growing bias).

Set	Accuracy	Train–Val Gap
Training	48.67%	8.20 pp
Validation	40.47%	

Table 6: KNN final results ($k = 15$, PCA-20).

3.5 Model Comparison

Table 7 and Figure 16 summarise all three models on the validation set.

Model	Best Setting	Train Acc	Val Acc	Gap
Logistic Regression	$C = 0.001$	45.65%	**41.58%**	4.07 pp
Linear SVM	$C = 10^{-5}$	42.73%	41.08%	1.65 pp
KNN	$k = 15$, PCA-20	48.67%	40.47%	8.20 pp

Table 7: Validation results for all three classifiers. Logistic Regression achieves the highest validation accuracy and was selected as the final model.

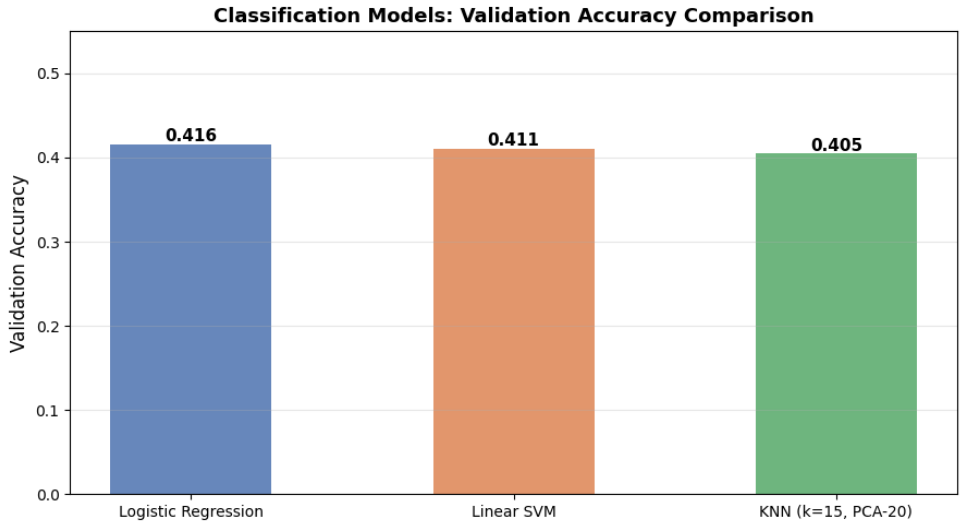

Figure 16: Validation accuracy comparison across all three classifiers. Logistic Regression (41.6%) edges out Linear SVM (41.1%) and KNN (40.5%).

3.6 Confusion Matrices

Figure 17 shows the confusion matrices for all three models on the validation set.

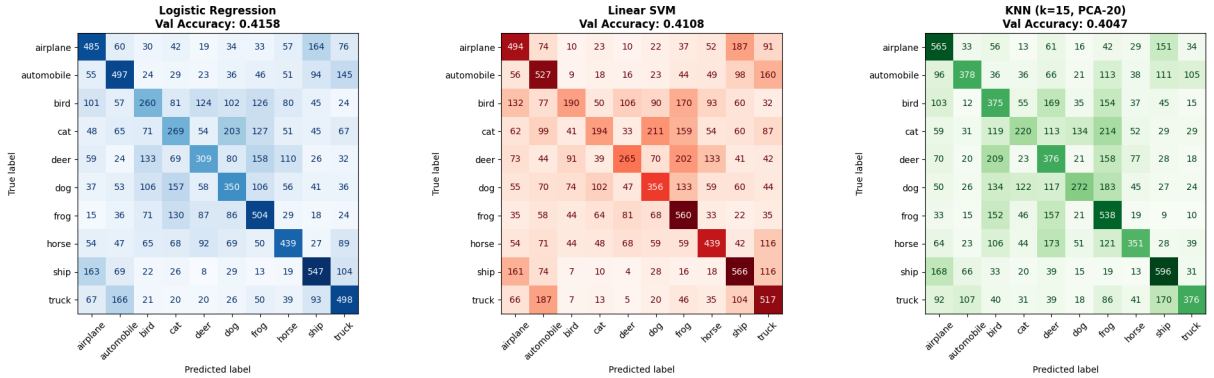

Figure 17: Confusion matrices on the validation set (10,000 examples) for Logistic Regression, Linear SVM, and KNN (PCA-20). Rows are true labels; columns are predicted labels.

Across all models, **ship** is the easiest class to recognise (LR: 54.7%, SVM: 56.6%, KNN: 59.6%), most likely because ships appear against a distinctive blue-water background not shared by any other class. The hardest class for LR and SVM is **bird** (LR: 26.0%, SVM: 19.0%); for KNN it is **cat** (22.0%). The most frequent single error for LR and SVM is *cat* being predicted as *dog*, which makes intuitive sense given their similar size and texture. For KNN the most frequent error is *cat* predicted as *frog* — an unusual pattern explained by the aggressive dimensionality reduction to 20 PCA components, which discards the fine texture differences that separate the two classes.

3.7 Discussion

1. Overall performance

All three models reach 40–42% validation accuracy, well above the random baseline of 10% (10 uniform classes) but far below human performance ($\approx 94\%$). This plateau is expected: classical models operating on raw pixel vectors face a fundamental representational ceiling on this dataset.

2. Effect of high dimensionality

Each image is treated as 3,072 independent numbers, with no notion of which pixels are neighbours. This has different consequences for each model:

- **Linear models (LR, SVM):** With 3,072 free parameters per class, the models can easily overfit without strong regularisation. The optimal values — $C = 0.001$ for LR and $C = 10^{-5}$ for SVM — are both very small, confirming this. Despite this, the train–val gaps remain modest (4.1% and 1.7%), so both models generalise reasonably.
- **KNN:** Distance-based reasoning breaks down in high dimensions because all pairwise distances concentrate around the same value, making "nearest neighbour" nearly meaningless. PCA to 20 components partially solves this but also discards information (73.6% variance retained), explaining KNN's larger gap (8.2%) and lower accuracy (40.47%).

3. Logistic Regression vs. Linear SVM

Both models learn a linear decision boundary in pixel space and reach nearly the same accuracy (41.58% vs. 41.08%). The small advantage of LR is consistent with its probabilistic objective: the cross-entropy loss penalises confidently wrong predictions, which can improve calibration when class boundaries are diffuse — as they are in raw pixel space. SVM's hinge loss, in contrast, ignores correctly classified examples beyond the margin, producing a sparser solution and a tighter train–val gap (1.65 pp vs. 4.07 pp for LR), but a marginally lower accuracy. In practice the two models are interchangeable at this accuracy level.

4. Limitations of classical methods on images

Three fundamental problems explain why all three methods plateau near 40%:

1. **No spatial awareness.** Pixels are treated as independent features. Edges, shapes, and textures — the cues humans use — are invisible to the model because it has no knowledge of which pixels are adjacent.

2. **No invariance.** A translated, rotated, or rescaled object produces a completely different pixel vector, so the model must re-learn each geometric variation from examples in the training set.

3. **Linear decision boundaries** (for LR and SVM). The true class boundaries in image space are highly non-linear and non-convex; a single hyperplane per class pair cannot represent them.

Convolutional Neural Networks overcome all three limitations through shared local filters (spatial awareness), pooling (translation invariance), and deep non-linear composition — which is why CNNs achieve $> 90\%$ accuracy on the same raw pixel inputs.

3.8 Final Test Set Evaluation

Logistic Regression was selected as the final model based on the highest validation accuracy (41.58%). The test set was used exactly once, after model selection was complete.

Model	Val Accuracy	Test Accuracy
Logistic Regression ($C = 0.001$)	41.58%	**41.57%**

Table 8: Final test result. The near-identical validation and test accuracies confirm that no overfitting to the validation set occurred.

The agreement between validation (41.58%) and test (41.57%) accuracy confirms that the model generalises well and that the validation-based model selection procedure was sound.

4 Part III: Deep Learning with PyTorch

4.1 Architecture

We implemented a fully-connected multilayer perceptron (MLP) with two hidden layers:

```
Input: 3,072  (32x32x3 flattened image)
  -> Linear(3072, 1024) -> BatchNorm -> ReLU -> Dropout(0.30)
  -> Linear(1024,  512) -> BatchNorm -> ReLU -> Dropout(0.30)
  -> Linear(512,    10)   [output logits]
```

Total trainable parameters: 3,679,754. BatchNorm stabilises layer activations and allows higher learning rates. Dropout at rate 0.30 acts as a regulariser during training, randomly zeroing activations to prevent co-adaptation of neurons.

4.2 Hyperparameter Search

We evaluated 12 configurations, each trained for 3 epochs on the training set and scored by validation accuracy. The search covered:
architecture depth and width $\in$ {[1024→512], [2048→1024], [1024→512→256]},
learning rate $\in \{5\times10^{-4}, 10^{-3}, 2\times10^{-3}\}$,
dropout $\in$ {0.2, 0.3, 0.4},
weight decay $\in \{0, 10^{-4}, 5\times10^{-4}\}$,
batch size $\in$ {64, 128, 256},
and activation function $\in$ {ReLU, ELU}.

Table 9 shows the top and bottom results from the search.

Rank	Hidden	LR	Dropout	WD	BS	Val Acc (3 ep.)
1 (best)	[1024, 512]	0.001	0.3	0	64	53.70%
2	[1024, 512]	0.001	0.4	1e-4	64	53.30%
3	[1024, 512]	0.001	0.2	1e-4	64	52.60%
11	[1024, 512]	0.001	0.3	5e-4	64	47.70%
12 (worst)	[1024, 512]	0.001	0.3	1e-4	64	47.70%

Table 9: Selected results from the 12-configuration hyperparameter search (3 epochs each). The best configuration uses no weight decay; ELU activation and high weight decay (rank 11–12) performed worst.

The winning configuration is summarised in Table 10.

Setting	Value
Architecture	$1024 \to 512$ (2 hidden layers)
Activation	ReLU
Learning rate	10^{-3}
Dropout	0.30
Weight decay	0
Optimizer	Adam
Batch size	64
Epochs (final)	10

Table 10: Best hyperparameter configuration, selected by validation accuracy on the 3-epoch search.

Key observations from the search. No weight decay ($\lambda = 0$) outperformed both 10^{-4} and 5×10^{-4}, suggesting that Dropout alone provides sufficient regularisation for this architecture. Strong weight decay (5×10^{-4}) hurt accuracy noticeably, likely because it over-constrains the weight magnitudes alongside the existing Dropout. ELU activation performed on a par with the worst weight decay setting, indicating that ReLU is a better fit for this problem. Learning rate 10^{-3} was optimal: 2×10^{-3} caused instability and 5×10^{-4} converged too slowly within the 3-epoch budget.

4.3 Training and Results

The best configuration was retrained for 10 epochs. Figure 18 shows the training dynamics.

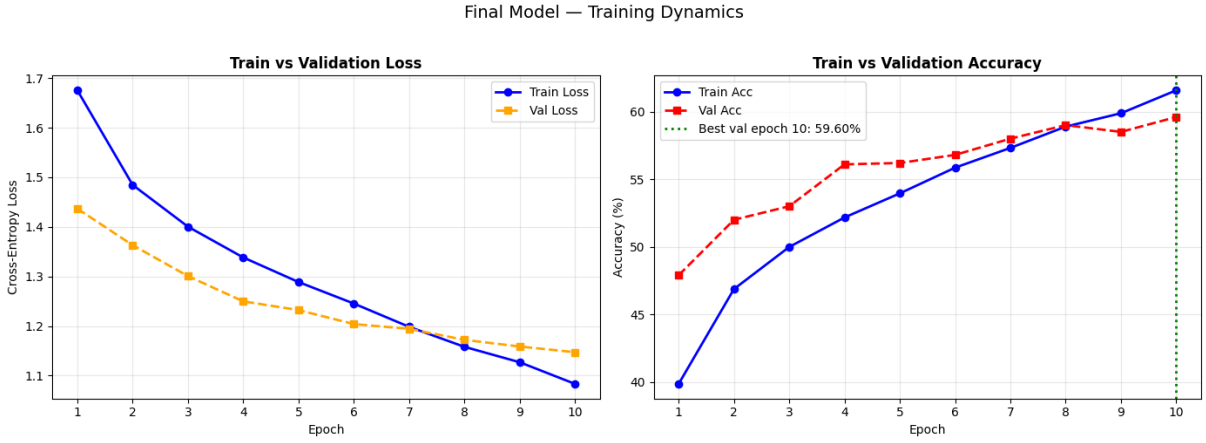

Figure 18: Final model training dynamics over 10 epochs. *Left:* train and validation cross-entropy loss. *Right:* train and validation accuracy with a marker at the best validation epoch (epoch 10, 59.60%). Validation loss starts below training loss because Dropout is disabled at evaluation time, effectively creating a larger ensemble model. The small train–val gap (1.98%) confirms good generalisation.

Interpreting the curves. A notable feature is that validation loss is lower than training loss throughout all 10 epochs, and validation accuracy is higher than training accuracy in the early epochs. This is expected behaviour with Dropout: during training, 30% of neurons are randomly dropped at each step, which effectively reduces the model

capacity and makes training harder. During evaluation, all neurons are active and their outputs are scaled accordingly, giving the validation pass access to the full model. As training progresses, the train–val gap in accuracy narrows and stabilises at 1.98 percentage points by epoch 10, confirming that the model is not overfitting.

Metric	**Value**
Best validation accuracy	59.60% (epoch 10)
Final training accuracy	61.58%
Train–val gap	1.98 pp
Test accuracy	**56.87%**

Table 11: Final model results. The test set was evaluated once, after the final model was fully trained.

Table 12 places the neural network result in the context of all methods evaluated across the project.

Method	**Test Accuracy**
Random guessing	10%
KNN ($k = 15$, PCA-20)	40.5%
Linear SVM ($C = 10^{-5}$)	41.1%
Logistic Regression ($C = 0.001$)	41.6%
Neural Network (MLP, this work)	**56.87%**
Reference: CNN (ResNet-18)	85–95%

Table 12: Test accuracy comparison across all evaluated methods on CIFAR-10.

The MLP improves over the best classical method (Logistic Regression, 41.6%) by **15.3 percentage points**. The remaining gap to state-of-the-art CNNs (85–95%) is expected: fully-connected networks treat each pixel independently and cannot share weights across spatial regions, so they must learn every spatial pattern from scratch without any inductive bias about image structure.

5 Conclusions

Part I — Regression. Both KNN ($k = 2$, Val $R^2 = 0.923$) and Polynomial Regression (degree 2, Val $R^2 = 0.902$) substantially outperform the linear baseline ($\Delta R^2 \approx 0.11$) on the Auto MPG dataset. Polynomial regression is recommended for deployment: it is interpretable, requires only a matrix–vector product at inference time, and achieves Test $R^2 = 0.879$. Model complexity must be tuned carefully — polynomial degrees beyond 2 caused catastrophic overfitting, with validation R^2 becoming deeply negative. Among optimisation methods, SGD and Mini-Batch GD converged within 20 epochs, while Batch GD was still descending at epoch 200 with the same learning rate.

Part II — Classical Classification. All three classical methods reached 41–42% validation accuracy on CIFAR-10, well above the 10% random baseline but far below human performance. Logistic Regression ($C = 0.001$) was selected as the best model

with a test accuracy of 41.57%, confirming that the validation-based selection was sound (validation and test accuracies differed by only 0.01 pp). The fundamental ceiling for classical methods on raw pixel data stems from three limitations: no spatial awareness, no geometric invariance, and linear decision boundaries that cannot represent the complex class geometry in image space.

Part III — Deep Learning. A two-hidden-layer MLP with BatchNorm, ReLU, and Dropout (0.30) trained with Adam reached a **test accuracy of 56.87%**, a 15.3-point improvement over the best classical method. Hyperparameters were selected systematically over 12 configurations; the key finding was that Dropout alone provides sufficient regularisation — adding weight decay degraded performance. The small train–val gap of 1.98 pp confirms that the model generalises well within the fully-connected paradigm. The remaining gap to CNN-level accuracy (85–95%) reflects the absence of spatial inductive bias in the MLP: convolutional networks exploit local connectivity, weight sharing, and translation invariance — none of which are available to a flat pixel representation.

Key general lessons from the project: (1) Match model complexity to data structure and dataset size — overly flexible models (polynomial degree > 2, KNN with $k = 1$) memorise noise rather than signal. (2) Always evaluate the bias–variance trade-off empirically using validation curves, not by intuition alone. (3) Feature representation matters as much as model choice — the same pixel data that limits classical methods to 42% enables a simple MLP to reach 57%, and would enable a CNN to reach 90%+. (4) Regularisation must be calibrated to the model: for the MLP, Dropout alone outperformed Dropout combined with weight decay. (5) The test set should be evaluated exactly once, after all modelling decisions are finalised on the validation set.

Code Repository

All code and notebooks available at: `https://github.com/SvZol/IntroductionToAI`