Deep Learning Models for Image Classification using CIFAR-10 Dataset
Neural Networks - Deep Learning Course

Computer Science Department,Aristotle University of Thessaloniki

I. Introduction

This project addresses the foundational problem of image classification using one of the most widely adopted benchmark datasets, CIFAR-10. Consisting of 60,000 tiny 32x32 color images across 10 distinct object classes, CIFAR-10 serves as an ideal testbed for evaluating the performance of simple, yet conceptually powerful, classification algorithms and deep learning models.

In contrast to the previous project which focused on the non-parametric, distance-based classifiers k-Nearest Neighbors (kNN) and the Nearest Centroid Classifier (NCC), this section establishes the core methodology and rationale behind utilizing two prominent deep learning classifiers: the **Multilayer Perceptron (MLP)** and the **Convolutional Neural Network (CNN)**. Our primary goal is to highlight their fundamental architectural differences and their respective strengths and weaknesses when applied to image data.

Multilayer Perceptron (MLP)

The Multilayer Perceptron is a class of **feedforward neural networks** consisting of an input layer, one or more hidden layers, and an output layer. In the context of image classification, the MLP operates on the principle of a fully-connected architecture:

- **Flattened Input:** The input image (e.g., a $32 \times 32 \times 3$ image for CIFAR-10) must first be **flattened** into a single, long vector (e.g., 3072 features). This process **destroys the image's spatial structure and local pixel relationships**.
- **Fully-Connected Layers:** Each neuron in one layer is connected to every neuron in the subsequent layer.
- **Drawbacks for Images:** The full connectivity leads to a **massive number of parameters** (weights and biases), making the model highly susceptible to **overfitting** and computationally expensive to train, especially with large images. Furthermore, the loss of spatial information means the MLP cannot inherently learn features that are **translationally invariant** meaning it struggles to recognize the same object if its position in the image changes.

Convolutional Neural Network (CNN)

In contrast, the Convolutional Neural Network is a specialized architecture explicitly designed for processing data with a **grid-like topology**, such as images. It overcomes the limitations of the MLP by incorporating two key concepts:

- **Feature Learning with Convolution:** CNNs introduce **Convolutional Layers**, which utilize **filters (or kernels)** that scan over the input image. These filters learn local patterns (edges, textures, etc.) while preserving the 2D or 3D structure of the data.
  - **Local Connectivity:** Each neuron in a convolutional layer is only connected to a small, local region of the previous layer (the receptive field).
- **Weight Sharing and Translational Invariance:** All neurons in a specific feature map share the same weights. This principle of **weight sharing** drastically **reduces the number of parameters** and enables the network to detect a specific feature regardless of where it appears in the image, granting **translational equivariance** unlike the MLP.
- **Pooling Layers:** CNNs typically include **Pooling Layers** (e.g., Max Pooling) to downsample the feature maps, which helps the model become more robust to small variations in the input position and further reduces the computational load.

By comparing the performance and efficiency of MLP and CNN on image datasets, we highlight the trade-offs between architectural simplicity and generality (MLP) versus structural specialization and high performance (CNN).

II. The CIFAR-10 Dataset and Vector Preparation

The CIFAR-10 dataset is a widely adopted benchmark in computer vision, consisting of $60,000$ color images, each of size 32×32 pixels. These images are equally distributed across 10 distinct object classes (e.g., airplane, automobile, bird, cat, dog), with $6,000$ images per class. The dataset is conventionally split into two subsets:
Training Set: $50,000$ images (N_{train}) used to learn the structure of the data.
Test Set: $10,000$ images (N_{test}) used for final evaluation.

A key challenge of CIFAR-10 is the low resolution (32×32), which forces NNs to rely on fine details, and the high intra-class variance (e.g., different angles and lighting for the same object), making generalization difficult for shallow models. Vector Preparation for Multi-Layer Perceptron and Convolu-

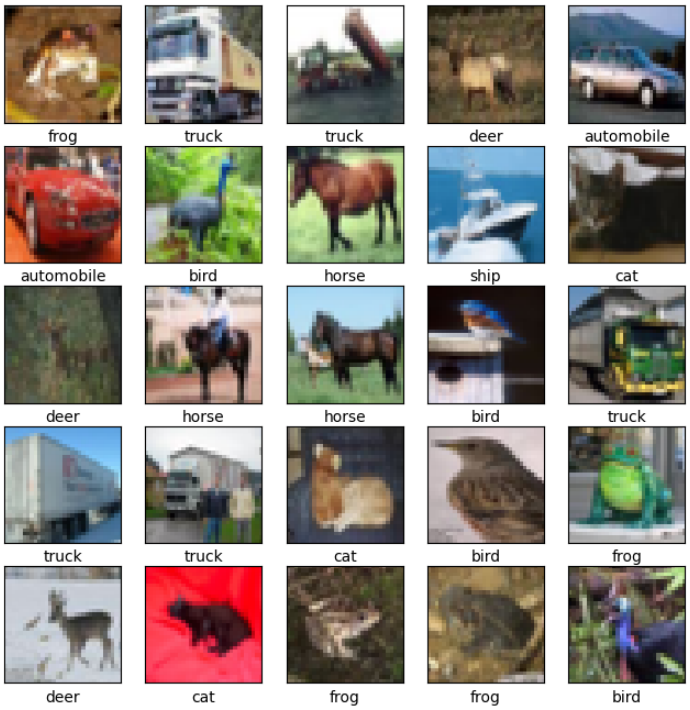


Fig. 1. Samples from every class of CIFAR-10.

tional Neural Network:

Flattening: It is required only for the Multi-Layer Perceptron (MLP).Each 32×32 color image has three color channels (Red, Green, Blue). To convert the image into a single feature vector $\mathbf{x}$, the dimensions are multiplied:

$$D = 32 \times 32 \times 3 = 3072 \text{ features}$$

Every image is thus represented by a 3072-dimensional vector. This process is necessary because an MLP's input layer expects a single vector for each sample.

A Convolutional Neural Network (CNN) input layer is designed to accept data in its raw, multi-dimensional form, which is crucial for processing images,unlike a Multi-Layer Perceptron (MLP) which requires a flattened $1D$ vector input.

Normalization: The raw pixel intensity values range from 0 to 255. To prevent features with inherently larger magnitude (such as those in the blue channel, if not properly scaled) from producing larger numerical gradients during the backpropagation process, this preprocessing step is required for both MLP and CNN. The large pixel values lead to larger gradients. This results in the optimizer taking huge, erratic steps across the loss landscape, causing the training process to oscillate and take many more epochs to converge. Thus, normalization is a critical step as it helps accelerate convergence during training and prevents larger input values from dominating the gradient calculations, leading to better model performance. This is achieved by converting the data type to float and dividing all pixel values by 255.0, resulting in feature vectors where all components lie in the normalized range $[0.0, 1.0]$. This step is crucial for ensuring stable and efficient training of the neural network.

One-hot Encoding: Transforms a single categorical integer label into a binary vector where only one element is "hot" (set to 1), and all other elements are "cold" (set to 0). The length of this vector is equal to the total number of classes on CIFAR-10.One-hot encoding is crucial in classification tasks because it solves the problem of misinterpreting class labels as ordinal data and is required for the standard loss function used in multi-class classification.Therefore, it is the necessary transformation to make the integer class labels compatible with the network's output structure and the loss function's mathematical requirements.

Training Algorithm for MLP and CNN

Both Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN) are trained using the Backpropagation algorithm, which is the foundational technique for teaching deep networks. Backpropagation's core purpose is to efficiently calculate the gradient (the partial derivative of the loss function with respect to every weight and bias) and then use these gradients to iteratively adjust the parameters, minimizing the overall error via a chosen optimizer. While the high-level process is universal, the specific mathematical implementation of gradient computation is tailored to the layer type. Backpropagation in an MLP relies on the simple, layer-by-layer application of the Chain Rule to compute the gradient of the final loss function with respect to every weight. Since MLP layers are Fully Connected—meaning every neuron in one layer connects to every neuron in the next—the primary mathematical operation is matrix multiplication. In the forward pass, input is multiplied by the weight matrix $\mathbf{W}^{(l)}$ and passed through an activation function. In the backward pass, the error signal ($\delta^{(l+1)}$) from the subsequent layer is propagated backward using the transpose of the weight matrix $(\mathbf{W}^{(l+1)})^T$ and multiplied by the derivative of the activation function, $f'(\mathbf{z}^{(l)})$. This results in an error signal $\delta^{(l)}$ that tells each neuron exactly how much it contributed to the total loss, enabling a simple matrix update for the weight gradients: $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^T$. This process is repeated until the first layer is reached. Conversely, in a CNN, the calculation is adapted for image structure: the error in a Convolutional layer is propagated backward using a form of Transposed Convolution to distribute the blame back to the relevant input regions. Instead of standard matrix operations, the backpropagation path must use convolution operations to correctly accumulate gradients. When calculating the gradient for the shared filter weights ($\frac{\partial L}{\partial \mathbf{K}}$), the error signal ($\delta^{(l)}$) from the next layer is convolved with the original input feature map $\mathbf{I}$. This convolution operation ensures that the gradient contributions from all spatial locations where the filter was used are summed up and applied to the single set of shared filter weights.Thus,the error signal $\delta^{(l)}$ is distributed back to the relevant local input region. Pooling layers further simplify the backward pass by only passing the error to the neuron that contributed the most (Max-Pooling). This adaptability allows Backpropagation to effectively train complex architectures while maintaining high computational competence.

III. Multi - Layer Perceptron Architecture

The final Multi - Layer Perceptron Model is featured with a total of four fully connected (**Dense**) layers.

Model Architecture:

- **Dense Layers**: Three hidden layers with $128, 256,$ and 256 neurons, respectively, all utilizing the ReLU activation function to introduce non-linearity, which is essential for the network to learn complex patterns. Because ReLU is nearly linear, preserves many of the properties that make linear models easy to optimize withgradient-based methods. Also it preserves many of the properties that make linear models generalize well.
- **Batch Normalization**: Following each hidden layer, Batch Normalization normalizes the output activations of the preceding Dense layer. It scales the outputs to have a mean of 0 and a standard deviation of 1.This stabilizes the inputs to the next layer, allowing the network to converge significantly faster, thereby accelerates training, allowing for higher learning rates, and also acts as a mild regularizer: Introduces a slight degree of noise during training because the mean and variance are calculated based only on the current mini-batch,therefore helps prevent overfitting.
- **Dropout**: It is a powerful regularization technique used to prevent overfitting. During training, it randomly sets a percentage of the neurons from the previous layer inactive in each training step (0.15 in the first layer, 0.35 and 0.3 in the subsequent ones). It ensures the network does not rely too heavily on specific neurons, improving generalization.
- **Output Layer**: The final output layer has 10 neurons (for the 10 CIFAR-10 classes) and uses the Softmax function that converts the raw outputs of the 10 neurons into a probability distribution over the different classes of CIFAR-10, where the sum of all outputs equals 1. The output with the highest probability is the network's prediction.

Model Compilation:

- **Adam Optimizer**: The model's training process is configured with the Adam optimizer using a starting learning rate of 0.01. Adam is an optimization algorithm that computes an individual, adaptive learning rate for every single parameter (weight) in the neural network, leading to faster and more stable training.
- **Categorical Cross-Entropy**: This is the standard loss function for multi-class classification problems when the target labels are one-hot encoded (which was done in the preprocessing step). It measures the difference between the predicted probability distribution (Softmax output) and the true one-hot distribution (target label).
- **Accuracy**: This is the metric used to monitor the model's performance during training and evaluation. It represents the percentage of correct predictions.

The final stage executes the training of the defined Multi-Layer Perceptron (MLP) model, focusing on robust optimization and performance assessment. The training uses the preprocessed data, running for a maximum of 50 epochs with a batch size of 128, and reserving 10% of the training data as a validation set for monitoring. Two key callbacks are employed to manage the process efficiently:

- **Early Stopping**:This callback is a form of regularization that prevents overfitting and saves computation time. It monitors the validation accuracy and halts training if no improvement is seen after a patience of 15 epochs, ensuring that when training stops, the model's weights will be reset to the epoch where the highest validation accuracy was achieved to prevent overfitting.
- **Learning Rate Scheduler**:This defines a dynamic strategy for adjusting the learning rate (LR) during training. A high LR is often used early for fast progress, and a low LR later for fine-tuning. It maintains the initial learning rate for the first 5 epochs before initiating an exponential decay to facilitate stable, fine-tuned convergence. After training completes (either by reaching 50 epochs or through early stopping), the model's true generalization ability is measured using the completely unseen test dataset (xtest and ytestonehot) via model.evaluate(), resulting in a final score for the Test Accuracy.

IV. Training Analysis and Evaluation on Multi - Layer Perceptron

The model's performance was rigorously evaluated using Categorical Cross-Entropy Loss, a standard metric for multi-class classification, monitored across the entire training duration for 50 Epochs. The resulting learning curves exhibited a critical issue identified as severe underfitting (high bias). Specifically, both the Training Loss and the Validation Loss converged prematurely to a high and virtually identical value of approximately 1.3.

This stabilization at 1.3 is significant for two reasons. First, while it is better than the random baseline loss for a 10-class problem (which is ≈ 2.30), it remains far from an acceptable level for modern image classification, indicating the model is only marginally learning the task. Second, the minimal divergence between the two curves is the definitive signature of underfitting; the model fails to adequately capture the complexity of the training data itself, resulting in poor generalization to the validation set.

This poor convergence is directly reflected in the final Test Accuracy, which achieved only 54%. While 54% accuracy is significantly better than the random baseline of 10% (since there are 10 classes in CIFAR-10), it is a weak result for this well-studied dataset, where competitive deep learning models regularly exceed 85% accuracy. The high loss (≈ 1.3) confirms that the model is making large, confident errors on many samples, which limits the overall accuracy.

The combination of high, similar losses (≈ 1.3) and low accuracy (54%) definitively points to a lack of model capacity. This deficiency stems primarily from the use of a Multilayer Perceptron (MLP) architecture on a spatially complex dataset like CIFAR-10. By flattening the image input, the MLP destroys the critical local spatial hierarchy and translational invariance necessary for effective visual feature extraction. The current MLP configuration simply lacks the inductive

bias to model the intricate visual patterns required for reliable classification.Addressing this requires structural intervention: either dramatically increasing the depth and width (layers and neurons) of the MLP to force it to learn non-local spatial relationships, or as we will examine in the following sections, transitioning the architecture entirely to a Convolutional Neural Network (CNN), which inherently preserves and exploits the spatial coherence of the image data.

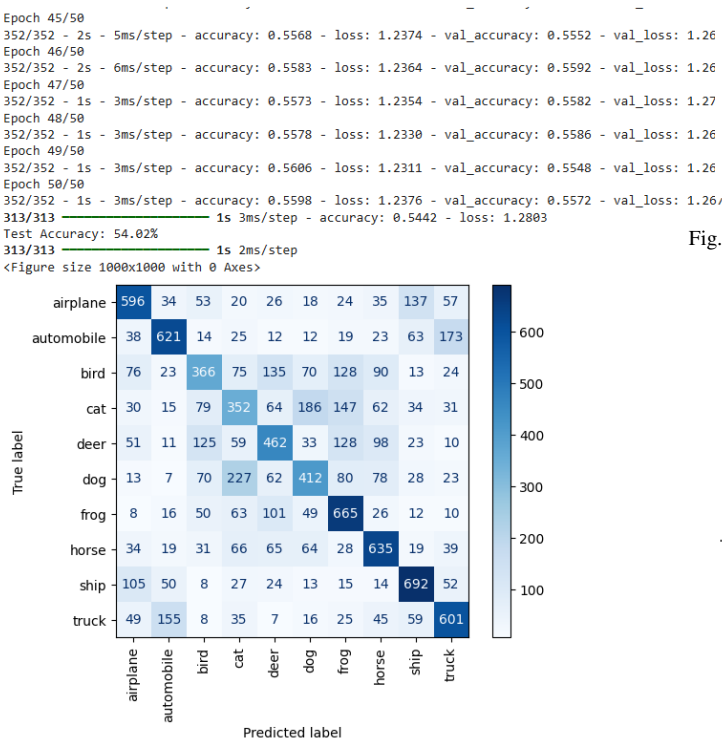


Fig. 2. Test Accuracy 54% and Confusion Matrix for MLP.

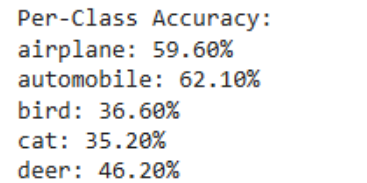


Fig. 3. Per-Class Accurracy for CIFAR-10.

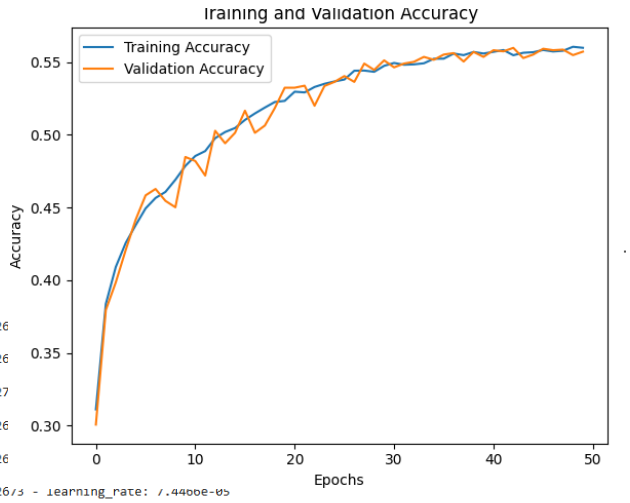


Fig. 4. Training and Validation Accuracy Curve.

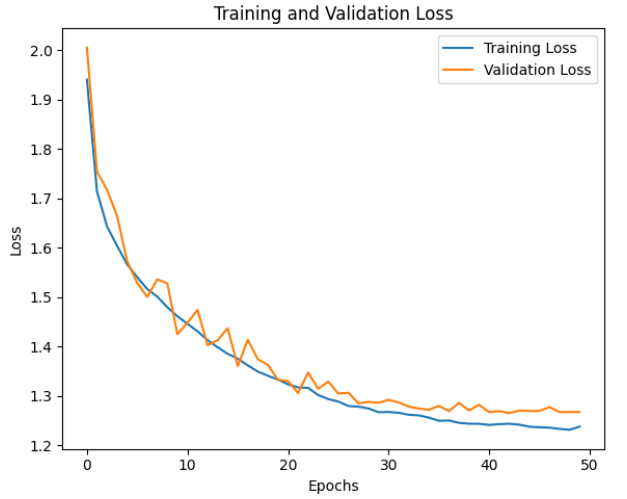


Fig. 5. Training and Validation Loss Curve.

Fig. 6. Correct and Incorrect Class Prediction.

V. CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

The proposed Convolutional Neural Network is described in the following sections, outlining the strategy for data preprocessing, the design of the architectural blocks, detailing the function and rationale behind each layer, and the final compilation configuration.

Model Architecture

Feature Extraction Layers

- **Data Augmentation:** The model utilizes aggressive data augmentation, implemented via the ImageDataGenerator, to enhance the robustness and generalization capabilities of the network, mitigating the risk of overfitting on the training set. The defined transformations include random rotation up to $25°$, horizontal and vertical shifts up to 60% of the image dimensions, a 60% shear intensity, and up to 60% zoom. Crucially, horizontal_flip=True introduces horizontal mirroring, and fill_mode='nearest' ensures that newly created pixels resulting from the transformations are populated using the nearest available pixel value. This comprehensive set of geometric transformations significantly expands the effective training data manifold, encouraging the model to learn features invariant to minor changes in image pose and scale and to expose the CNN to a wider variety of images than it has seen before, teaching it to recognize the core objects and patterns regardless of their position, orientation, scale, or lighting conditions .
- **Convolutional and Pooling**: The feature extraction component is constructed as a deep, three-block hierarchical structure designed to capture features from edge details to complex patterns.The layer takes an input image (or volume of feature maps) and slides a small, weighted matrix—the kernel (or filter)—across its spatial dimensions. At every position, it performs a matrix multiplication between the filter's weights and the corresponding input pixels, producing a single output value.
  Each of the three blocks strictly follows the sequence: Conv2D $\rightarrow$ ReLU $\rightarrow$ MaxPooling2D $\rightarrow$ Dropout.This repeating pattern is a fundamental design choice in CNNs to iteratively extract features.Pooling is applied after each convolutional block to downsample the feature maps. The 2×2 pool size, combined with a stride of 2 (the default for MaxPooling2D), reduces both the height and width of the feature map by half, resulting in a 75% reduction in spatial dimensionality. The Max Pooling function operates by selecting only the maximum activation within the 2×2 local window meaning it reports the maximum output within a rectangular neighborhood (summarizes the responses) and helps to make the representation approximately invariant to small translations of the input.
- **Kernel and Padding**: All layers use a 3×3 kernel size, chosen for its optimal balance between computational efficiency and receptive field size, efficiently capturing local pixel relationships. The kernel convolves across the width and height of the input image in the first layer and performs element-wise multiplication with the corresponding pixels of its size. Then sums the results and produces a **single pixel** in the output. Each kernel learns to detect a specific, low-level feature in the image. The use of 'same' padding is crucial: it ensures that the spatial output dimension (height $\times$ width) of the convolution layer matches its input dimension, thereby preserving information density before downsampling by the pooling layers. Without padding the spatial dimensions (height and width) of the output feature map would **shrink**.
- **Filter Scaling**:The filter count is aggressively scaled across the depth of the network: 65, 130, and 258 filters. This exponential increase allows the network to dedicate a greater capacity (more filters) to capturing the highly abstract and synthesized features in the deeper layers.The shallow layer with 65 filters primarily learn basic, low-level features while the Deep Layers with 130 and 258 filters integrate and synthesize high-level features and complex patterns.
- **Activation and Initialization**: Rectified Linear Unit (ReLU) is used for its effectiveness in introducing non-linearity without suffering from the vanishing gradient problem, which is common in older activations. The use of 'he_normal' kernel initialization is perfectly paired with ReLU, as it initializes weights to maintain activation variance through the layers.Otherwise, if the initial weights of the kernels are too large or too small, either activations will grow rapidly layer after layer, leading to exploding gradients during backpropagation or shrink toward zero, causing the $\max(0, x)$ part of ReLU to output zero for most inputs. This results in dead neurons and vanishing gradients, stalling the learning process.
- **Dropout**: This layer after each pooling step performs local regularization, temporarily nullifying 20% of the feature map activations. This ensures that the subsequent convolutional layer is forced to learn robust feature combinations that do not rely on a few specific preceding neurons.

Classification Layers

- **Global Average Pooling**: This layer serves as the crucial link between the convolutional base and the dense classification layers. It takes the output of the last convolutional layer (which has a shape of $W \times H \times D$, where W is width, H is height, and D is the depth/number of feature maps) and calculates the average value for each $W \times H$ feature map. The output is a $1D$ vector, meaning it averages each 2D feature map into a single scalar, drastically reducing the size to only 258 elements (D), which is the number of feature maps. Significantly reduces the total number of parameters, making the model less prone to overfitting and often allowing for better localization of features learned by the CNN and acts as a superior regularization technique.It doesn't matter exactly where a feature is detected; only that it was detected.This method enhances exlplainability because GAP creates a direct link between the output feature map and the final decision of our network.

- **Dropout**: Even though this regularization would primarily be considered as ineffective after a Global Average Pooling (because GAP has zero trainable parameters itself), through trial and error method we observed that **without** this, our model experienced severe overfitting after 15 epochs.After adding **Dropout**, the overfitting was resolved, meaning that despite the benefits of GAP for reduction of parameters, our model still contained enough learnable capacity to memorize the training data.In this scenario, applying **Dropout(0.3)** we are randomly setting 30% of the feature vector elements to zero and we are essentially forcing the Softmax layer's weights to learn to classify the input correctly using only 70% of the available features at any one time. This process prevents the Softmax layer's weights from becoming overly reliant on any single feature map (channel) or a small group of channels. If the network overfits by creating complex, co-dependent feature maps, Dropout helps to break these dependencies just before the final decision is made.The noise introduced by Dropout ensures that the learned features are more robust and less brittle, leading to better generalization and eliminating the observed overfitting.

```
Test accuracy: 80.47%
Elapsed time: 188.40472197532654 seconds
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

Per-class accuracy:
airplane: 86.80%
automobile: 91.00%
bird: 69.70%
cat: 56.30%
deer: 81.20%
dog: 76.70%
frog: 86.30%
horse: 82.20%
ship: 90.20%
truck: 84.30%
```

Fig. 7. Test accuracy 80%, total elapsed time and per-class accuracy.

- **Final Dense Layer**: The network concludes with a Dense layer of 10 units, precisely matching the number of classes of CIFAR-10. The activation function applied to the output is softmax. It converts the raw scores (logits) into a probability distribution. The sum of the 10 output values will equal 1, representing the probability of the input belonging to each of the 10 classes. It ensures the raw output logits are transformed into a probability distribution, where each of the 10 outputs represents the probability that the input image belongs to the corresponding class.

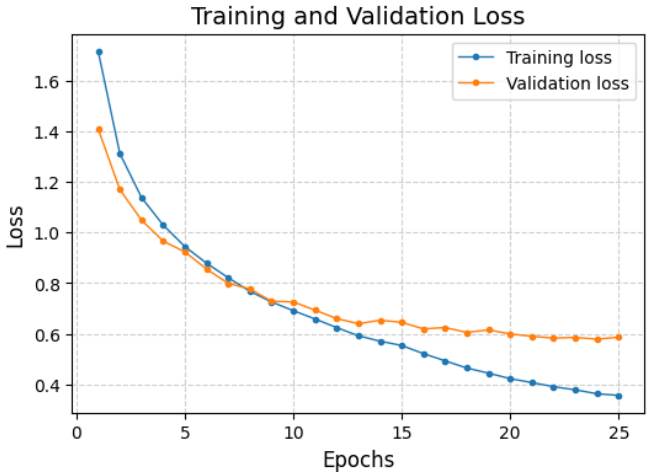


Fig. 8. Training and Validation Loss without Dropout.

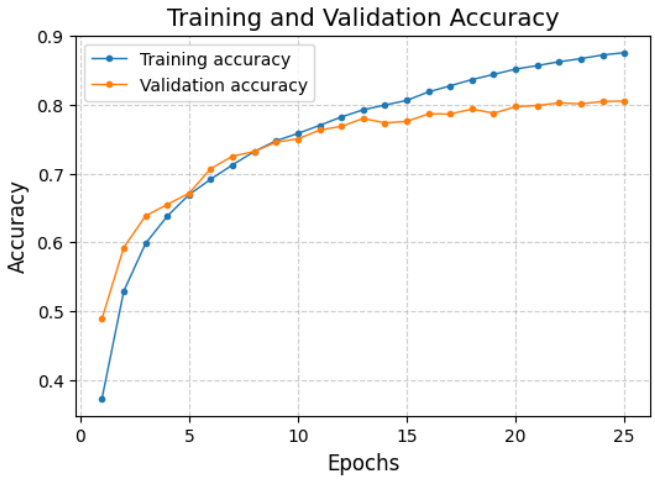


Fig. 9. Training and Validation Accuracy without Dropout.

Effectiveness of Kernel Initializer

The use of the He normal kernel initializer significantly enhanced the performance and efficiency of the Convolutional Neural Network (CNN) compared to using the default initialization. **He_normal** is a method for setting the initial weights in a neural network layer. Its primary goal is to prevent the vanishing or exploding gradient problem during training. Without explicit initialization (default), the model required 50 epochs to reach a final validation accuracy of 80.0%. In contrast, by implementing he_normal, the model achieved a superior accuracy of 81.99% in just 25 epochs. This demonstrates that He initialization, which is specifically designed for networks using the ReLU activation function by maintaining stable activation variance (of the initial random numbers assigned to the weights of a layer), allowed the optimization process to start from a better parameter space. This resulted in faster convergence (halving the required training time) and finding a more optimal minimum in the loss landscape, ultimately yielding a higher final classification accuracy.

```
Test accuracy: 80.81%
Elapsed time: 354.5283422470093 second
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/st

Per-class accuracy:
airplane: 84.90%
automobile: 91.60%
bird: 71.90%
cat: 60.90%
deer: 80.90%
dog: 71.70%
frog: 85.70%
horse: 83.50%
ship: 91.10%
truck: 85.90%
```

Fig. 10. Test Accuracy 80.81% and per-class accuracy without kernel initializer.

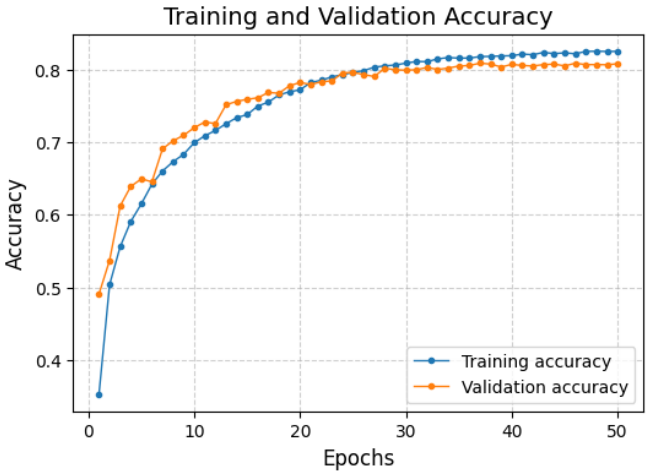


Fig. 11. Training and Validation Accuracy without kernel initializer.

Model Compilation

- **Adam Optimizer**: Optimizer is the most critical component in the training phase of a neural network, as it dictates how the network's weights are adjusted based on the calculated error (loss). Its' primary goal is to find the set of internal model parameters (weights and biases) that minimizes the Loss Function (e.g., Categorical Cross-Entropy). It's preferred because it adapts the learning rate for each individual weight in the network, leading to faster and more stable convergence.
- **Categorical-Cross Entropy**: This is the objective function that quantifies the difference between the model's predictions and the true labels. **Adam optimizer's** goal is to minimize this function. It measures the difference between the predicted probability distribution (Softmax output) and the true one-hot distribution (target label).

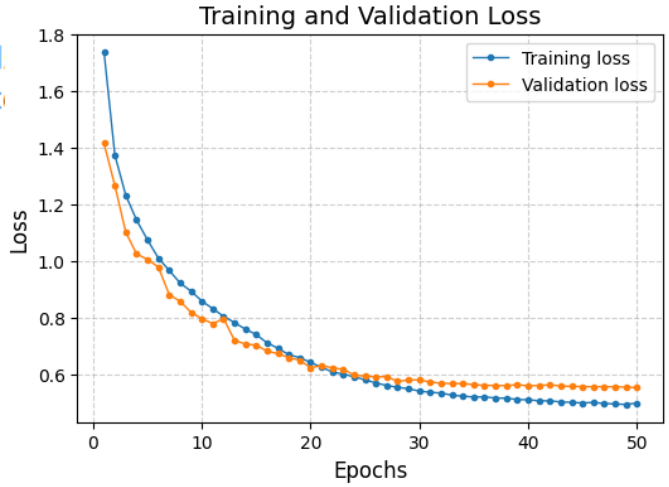


Fig. 12. Training and Validation Loss without kernel initializer.

- **Accuracy**: This is the metric for monitoring the performance of the model. It measures the ratio of correct predictions to the total number of predictions.

The final stage executes the training of the defined Convolutional Neural Network (CNN) model, focusing on optimization and performance assessment. The training uses the preprocessed data, running for a maximum of 25 epochs with a batch size of 64. Two key callbacks are employed to manage the process efficiently:

- **Early Stopping**: This callback is a form of regularization that prevents overfitting and saves computation time. It monitors the validation accuracy and halts training if the validation loss (val_loss) does not improve by at least 0.001 (min_delta) over 10 consecutive epochs and restores the best weights.
- **Learning Rate Scheduler**: This defines a dynamic strategy for adjusting the learning rate (LR) during training. A high LR is often used early for fast progress, and a low LR later for fine-tuning. It maintains the initial learning rate for the first 15 epochs before initiating an exponential decay to facilitate stable, fine-tuned convergence. After training completes (either by reaching 25 epochs or through early stopping), the model's true generalization ability is measured using the completely unseen test dataset (x_test and y_test_onehot) via model.evaluate(), resulting in a final score for the Test Accuracy.

VI. TRAINING ANALYSIS AND EVALUATION ON CONVOLUTIONAL NEURAL NETWORK

The model's performance was monitored across the entire training duration for 25 epochs on both the training and validation datasets and was evaluated using Categorical Cross-Entropy Loss. The resulting learning curves reveal key insights into the model's learning dynamics and generalization ability. The Training Accuracy/Loss learning curves measure the model's performance on the data it is actively using to update its weights. It reflects the model's capacity to learn and memorize the training set patterns. Training Accuracy generally

increases throughout training, demonstrating that the model is successfully minimizing the loss function and mapping inputs to their correct labels. The Validation Accuracy/Loss is measured on a separate, held-out dataset that the model does not use for weight updates. This is the most crucial indicator of the model's ability to generalize to new, unseen samples. By monitoring the trend between these two sets of metrics we observe that both curves track closely, indicating the model learns the underlying patterns without memorizing meaning the model can generalize well.

Training and Validation Accuracy

The Training and Validation Accuracy plot shows a steep initial rise for both datasets, indicating rapid learning. Critically, the Validation Accuracy consistently stays slightly above the Training Accuracy until approximately epoch 18, where they begin to converge and stay close for the remaining epochs, both exceeding 80% accuracy. This high, stable validation accuracy is the primary goal of training, signifying strong generalization.This behavior—where validation performance is higher than training performance in the early stages—is observed because of the **Dropout**. Since Dropout is only active during training, the training process is artificially hindered, while the validation set (which runs without Dropout using all the network's abilities) benefits from the full network capacity, leading to temporarily superior validation results. Moreover, the impact of the **Learning Rate Scheduler** is demonstrated in the curves: A high initial learning rate speeds up the training process by moving quickly across the loss surface. A reduced learning rate in later epochs as the model progresses, helps avoid divergence because smaller steps are needed to fine-tune the weights, and ensures the model finds a deeper, more stable minimum rather than oscillating around it. By taking smaller steps later on, the model achieves better generalization. The accuracy learning curves show well-managed training process where regularization techniques successfully prevented overfitting, leading to a strong, generalized model performance with high validation accuracy.

Training and Validation Loss

During training, the primary objective of the optimizer **Adam** is to continuously minimize this loss. Tracking the loss on both training and validation sets provides the clearest picture of the learning process.The Validation Loss is consistently lower than the Training Loss for almost the entire duration, both exhibiting a monotonic decrease. This indicates that the model is successfully minimizing the loss function on both datasets and is not exhibiting overfitting..This is a very strong sign of excellent generalization and proper regularization (due to **Dropout and Global Average Pooling**), confirming that the model's performance on unseen data is at least as good as its performance on the training data. Since the validation loss never rises (diverges) while the training loss continues to fall, overfitting is successfully avoided. The convergence of both accuracy and loss curves near the end of the 25 epochs, with validation metrics performing slightly better or equal to training metrics, suggests that the model is well-regularized and has achieved a stable, optimal level of performance under these hyperparameters, indicating that the Early Stopping callback (if configured to stop based on minimal change) would likely not have triggered until much later, or not at all, as the loss continues to slightly decrease right up to the final epoch. The training appears to have concluded successfully without instability. By observing the Training loss curve at the final epoch, the curve has stabilized and reached a value below 0.6. This value represents the minimum error the model achieved on the data it was trained on. This low training loss indicates that the **Adam optimizer** was highly effective in minimizing the error during the backpropagation process, resulting in a very good fit to the training set. The final Validation loss is very similar to the Training Loss thus it confirms that the model is generalizing very well.

```
Test accuracy: 81.99%
Elapsed time: 187.6976034641266 seconds
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step

Per-class accuracy:
airplane: 86.20%
automobile: 92.40%
bird: 70.30%
cat: 67.80%
deer: 81.90%
dog: 70.20%
frog: 86.60%
horse: 86.30%
ship: 90.50%
truck: 87.70%
```

Fig. 13. Test Accuracy 81.99%, total elapsed time and per-class accuracy.

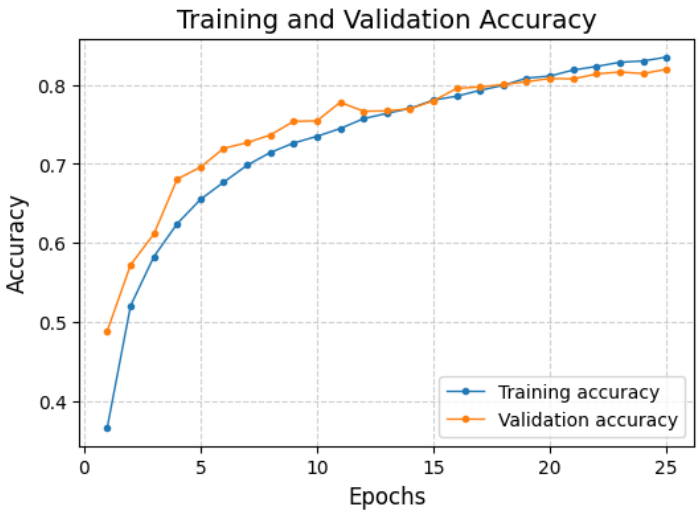


Fig. 14. Learning Curves for Accuracy of the CNN.

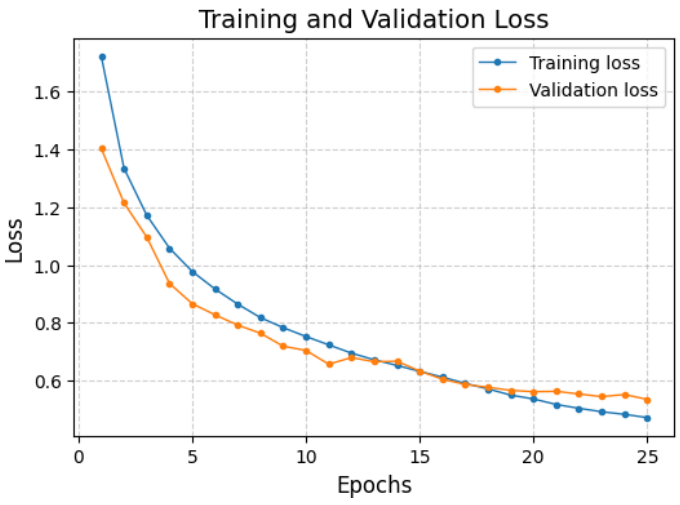
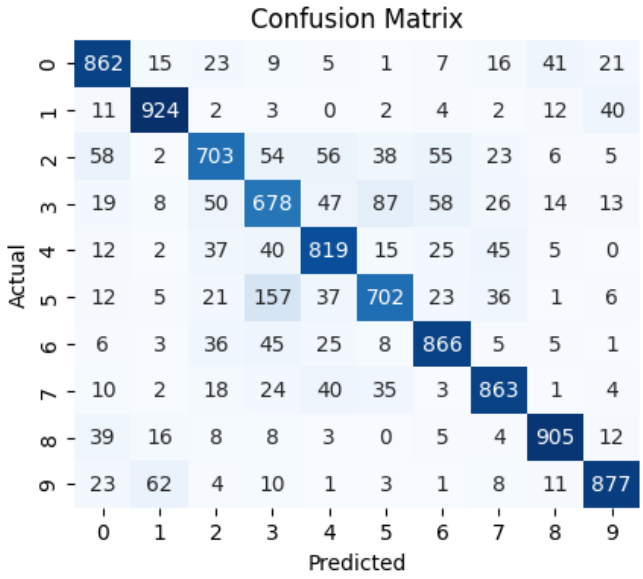


Fig. 15. Learning Curves for Loss of the CNN.



	precision	recall	f1-score	support
airplane	0.82	0.86	0.84	1000
automobile	0.89	0.92	0.91	1000
bird	0.78	0.70	0.74	1000
cat	0.66	0.68	0.67	1000
deer	0.79	0.82	0.81	1000
dog	0.79	0.70	0.74	1000
frog	0.83	0.87	0.85	1000
horse	0.84	0.86	0.85	1000
ship	0.90	0.91	0.90	1000
truck	0.90	0.88	0.89	1000
accuracy			0.82	10000
macro avg	0.82	0.82	0.82	10000
weighted avg	0.82	0.82	0.82	10000

Fig. 16. Confusion Matrix for the CIFAR-10.

Confusion Matrix Analysis

The provided Confusion Matrix and Classification Report detail the CNN model's performance on the CIFAR-10 dataset. The Confusion Matrix (top) visually represents the model's predictions versus the actual labels, with the diagonal elements (darker blue, e.g., 862 for class 0, 924 for class 1) showing the number of correctly classified instances for each class. Off-diagonal elements (e.g., 58, 19, 12 in the first column) represent misclassifications; for example, 58 images actually belonging to class 2 ('bird') were incorrectly predicted as class 0 ('airplane').The Classification Report (bottom) summarizes the performance for each class using:

- Precision: Out of all instances the model predicted as a specific class, what fraction actually belonged to that class. High Precision Means the model is highly reliable when it makes a positive prediction for that class (low false alarms), for instance the 'ship' class has a precision of 0.90. If the model says an image is a ship, there's a 90% chance it's correct.
- Recall: Out of all actual instances of a specific class, what fraction did the model correctly identify. High Recall means the model is good at finding all positive instances of that class (low missed opportunities). The 'automobile' class has a recall of 0.92. The model correctly identified 92% of all actual automobiles in the test set.
- F1-Score: The harmonic mean of Precision and Recall. It provides a single metric that balances both error types. It's the best measure when you need to avoid both False Positives and False Negatives. The metric reveal distinct patterns across the 10 classes:
  - **F1>90:** The classes automobile, ship, truck are the easiest for the CNN to distinguish due to unique, salient features. The diagonal elements are highest here (e.g., 924 for automobile, 877 for truck), showing minimal confusion.
  - **80<F1<90** : The classes airplane, horse, frog, deer show strong performance, but with slightly more confusion among visually similar categories. 'Airplane' is confused with 'bird' (23 $\rightarrow$ 7) and 'ship' (41 $\rightarrow$ 8).
  - **F1<75**: The classes cat, bird, dog, represent the most challenging classification task primarily involving small, non-rigid objects with high within-class variation (different breeds/poses). The class 'cat' has the lowest precision (0.66) and recall (0.68). The matrix shows 56 actual cats were predicted as birds (row 3, col 2), and 55 actual cats were predicted as dogs (row 3, col 6).

The model's strong aggregate metrics (82%) show that the CNN architecture and training parameters (e.g., Dropout, Adam optimizer, Learning Rate Scheduler) were effective, successfully tackling the multi-class image recognition task.

Model Compression

In order to see the impact of the filter size in our training, we reduced the number of filters in every **Dense** layer by half. The number of filters dictates the depth of the feature maps, which in turn determines the number of different features (e.g., edges, textures, corners) the layer can learn. Reducing the filters by half severely limits the network's ability to extract and represent the full complexity of the input data. The original model that achieved 82% accuracy was a better fit for the complexity of the dataset.The thinner model 75% accuracy is too simple to capture all the relevant patterns and distinctions in the data required for high performance.This reduction in model capacity led to underfitting and a drop in generalization performance. The 7% drop represents the value of the complex features that were learned by the 50% of filters we removed.

Fig. 17. Learning Curves for Accuracy of the CNN.

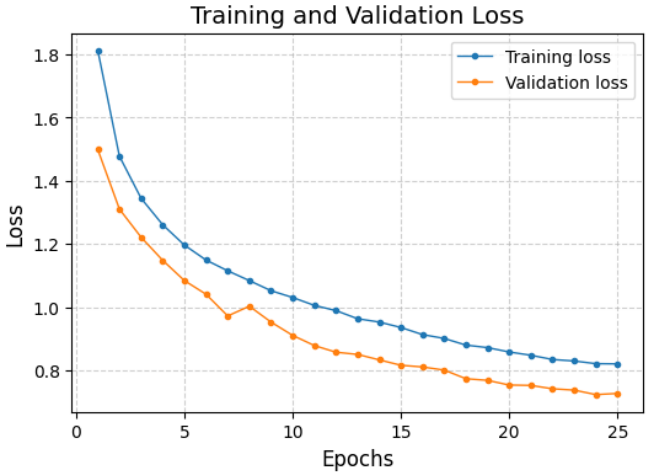


Fig. 18. Learning Curves for Loss of the CNN.

VII. COMPARISON OF OPTIMIZATION ALGORITHMS: ADAM VS STOCHASTIC GRADIENT DESCENT WITH MOMENTUM

When developing Convolutional Neural Networks, the choice of the optimization algorithm is crucial for efficient and effective training.In order to compare the effect of both algorithms on the model's performance, we implemented a second Convolutional Neural Network that optimizes the **Loss Function** with the Stochastic Gradient Descent Algorithm with the use of momentum and weight decay. SGD utilizes a high initial learning rate (0.1), Momentum (0.9), and Weight Decay (5×10^{-4}). While Adam is known for its adaptive learning rates for each parameter, often leading to faster convergence early on, SGD, when properly tuned with a higher initial learning rate (LR) (e.g., 0.1 compared to a typical Adam LR of 0.001) and Momentum (0.9 in this case), often achieves better generalization and can reach a deeper, sharper minimum in the loss landscape, especially in vision tasks. The inclusion of Weight Decay is particularly critical for the SGD variant as it helps prevent overfitting and stabilizes the training process, balancing the inherent fluctuations of the stochastic gradient updates. The learning curves for the SGD optimizer, trained over 75 epochs , display a classic and highly successful pattern for training a Convolutional Neural Network, especially given the use of Momentum and Weight Decay (L2 regularization).

- **For the first 10 epochs** both the Training Loss and, notably, the Validation Loss curves show a very rapid and steep drop. This is characteristic of a model starting with a high initial learning rate (0.1) combined with the accelerating effect of Momentum (0.9). This quick fall indicates the model is efficiently learning the core features right away. Crucially, during the first 4–5 epochs, the Validation Loss is lower than the Training Loss. This is a strong positive sign, indicating that the model generalizes exceptionally well immediately upon starting training. Correspondingly, the Training Accuracy and Validation Accuracy climb steeply from approximately 35% to 60%.
- **Until 40 epochs** improvement continues, but at a slower pace. The curves exhibit slight fluctuations or volatility, which is typical for standard SGD due to the noise introduced by stochastic gradient updates on small batches.The small distance between the curves suggests that the Weight Decay is effectively mitigating severe overfitting.The Validation Accuracy continues to rise steadily, reaching approximately 80% around Epoch 40.
- **The final epochs** show both the Training Loss and Validation Loss stabilize at a very low value (≈ 0.5). It is particularly noteworthy that the two curves virtually merge. This is the strongest evidence of outstanding generalization—the model performs equally well on both seen (training) and unseen (validation) data.The maximum Accuracy is achieved, both the Training Accuracy and Validation Accuracy stabilize at their highest value, approximately $\mathbf{83}\%$. Maintaining this high accuracy without divergence confirms the success of Weight Decay and Momentum in achieving a robust final state.

The start of an exponential learning rate (LR) decay at Epoch 25, specifically for the SGD optimizer, has a profound and visible effect on the learning curves.Before Epoch 25, the learning rate was high (0.1 or whatever the initial value was), allowing the model to take large steps and achieve rapid progress. The decaying LR forces the optimizer to take smaller, more cautious steps, preventing it from overshooting the minimum.After Epoch 25, the loss curves (both Training and Validation) become noticeably smoother and monotonically decreasing **faster than the previous epochs**.The Accuracy curves enter a phase of rising-ascent and experience a final, smooth push toward the highest point, leading to full convergence and the excellent generalization seen in the final phase.That continues until ≈ Epoch 40, after which they stabilize.

Final Model Comparison

The following comparison will explore the trade-offs and performance differences observed between these two optimization strategies and how the two optimizers perform under different training time constraints:

- **Speed and Convergence**: Adam lived up to its reputation for fast initial convergence. It quickly drove the validation accuracy up to about $\mathbf{82}\%$ in just 25 epochs.Thus, we conclude that the Adam optimizer was used for the short, efficient run.SGD, using a higher initial learning rate and Momentum, also converged rapidly initially, but required the full $\approx \mathbf{75}$ epochs to fully stabilize and reach its final performance peak. The curves appear very smooth over the long haul.
- **Accuracy and Generalization**:The Adam model reached a high training accuracy ($\approx 82\%$) and validation accuracy ($\approx 82\%$).The SGD model achieved slightly better training and validation accuracy ($\approx 83\%$). This demonstrates excellent generalization and suggests that the SGD, aided by Momentum and Weight Decay, found a very stable, deep minimum in the loss landscape after extended training.

The Adam optimizer proved to be the most time-efficient choice. It delivered a strong result (82%) in a quarter of the training time required for SGD to fully stabilize.The SGD with Momentum and Weight Decay ultimately delivered the best overall performance by a small margin, achieving the highest peak validation accuracy (83%) and demonstrating the superior generalization ability over the long training period.The results align with the general consensus that Adam is ideal for fast prototyping and convergence, while a well-tuned SGD often achieves slightly better final generalization performance when allowed to train for many epochs.

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8328 - loss: 0.4975
Test accuracy: 82.98%
Elapsed time: 556.0447912216187 seconds
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step

Per-class accuracy:
airplane: 86.50%
automobile: 93.60%
bird: 74.30%
cat: 67.70%
deer: 81.90%
dog: 75.10%
frog: 88.60%
horse: 82.90%
ship: 91.60%
truck: 87.60%
```

Fig. 19. Test Accuracy 82.99%, total elapsed time and per-class accuracy.

Fig. 20. Learning Curves for Training and Validation Accuracy.

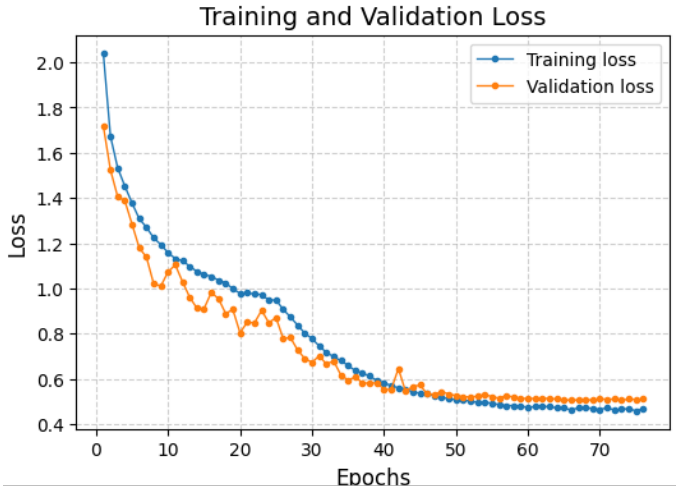
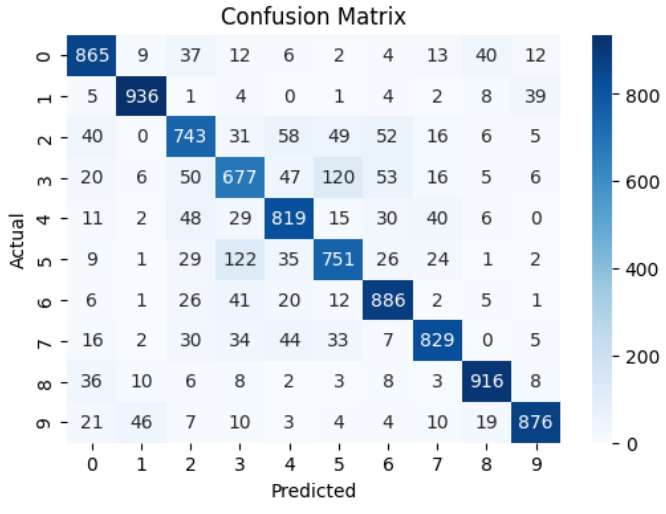


Fig. 21. Learning Curves for Training and Validation Loss.



	precision	recall	f1-score	support
airplane	0.84	0.86	0.85	1000
automobile	0.92	0.94	0.93	1000
bird	0.76	0.74	0.75	1000
cat	0.70	0.68	0.69	1000
deer	0.79	0.82	0.81	1000
dog	0.76	0.75	0.75	1000
frog	0.82	0.89	0.85	1000
horse	0.87	0.83	0.85	1000
ship	0.91	0.92	0.91	1000
truck	0.92	0.88	0.90	1000
accuracy			0.83	10000
macro avg	0.83	0.83	0.83	10000
weighted avg	0.83	0.83	0.83	10000

Fig. 22. Confusion Matrix for the CIFAR-10.

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8424 - loss: 0.461
Test accuracy: 83.88%
Elapsed time: 536.7696177959442 seconds
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

Per-class accuracy:
airplane: 86.00%
automobile: 92.40%
bird: 73.30%
cat: 67.60%
deer: 84.50%
dog: 77.90%
frog: 89.60%
horse: 85.70%
ship: 92.90%
truck: 88.90%
```

Fig. 23. Test accuracy 83.88%, total elapsed time and per-class accuracy.

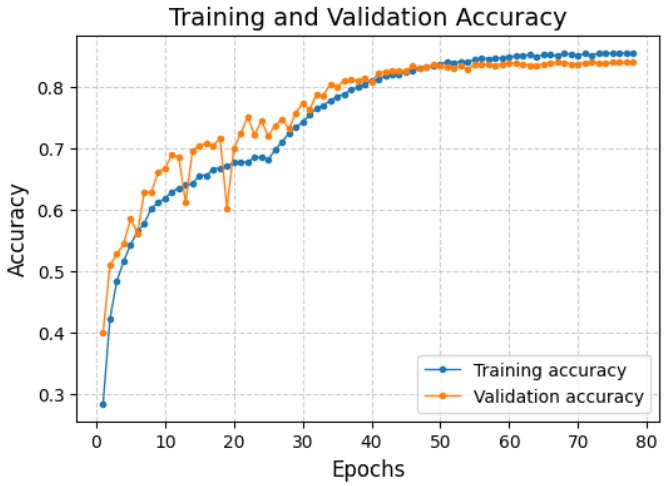


Fig. 24. Learning Curves for Training and Validation Accuracy.

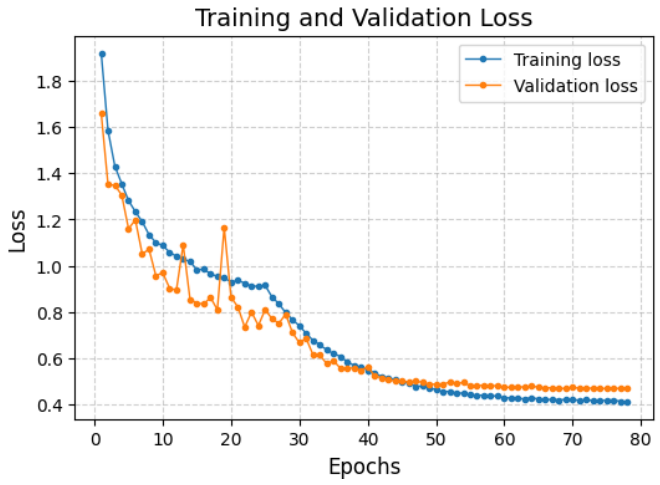
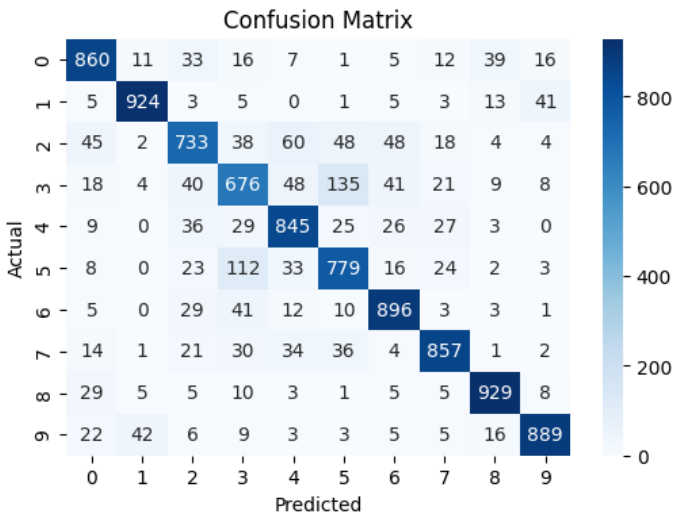


Fig. 25. Learning Curves for Training and Validation Loss.

Weight Decay Tuning

In order to enhance our model's performance and combat the slightest observation of overfitting, we are doubling the Weight Decay coefficient, typically from 5×10^{-4} to 1×10^{-3} (0.0005 to 0.001), as a standard step in the hyperparameter tuning process.Doubling the weight decay coefficient means doubling the penalty applied to the square of weights, thus weights will be smaller and more evenly distributed.The expected outcome is reduced risk of overfitting and better generalization, so a better test accuracy overall. The presented model demonstrates strong overall performance.The final test set accuracy is $\mathbf{83.88}\%$. This is slightly higher than the final validation accuracy of $\approx 83\%$ observed in the previous model.The final test loss is $\mathbf{0.4615}$, 3% lower than the final loss of the previous one, meaning our model has improved.The SGD model is a strong performer with an average test accuracy of nearly 84%. It excels at classifying vehicles (automobile, ship, truck) and frog, which have distinct visual characteristics. The primary area for improvement remains the classification of ambiguous animals, particularly cats, birds, and dogs, whose accuracy falls significantly below the model's average.



	precision	recall	f1-score	support
airplane	0.85	0.86	0.85	1000
automobile	0.93	0.92	0.93	1000
bird	0.79	0.73	0.76	1000
cat	0.70	0.68	0.69	1000
deer	0.81	0.84	0.83	1000
dog	0.75	0.78	0.76	1000
frog	0.85	0.90	0.87	1000
horse	0.88	0.86	0.87	1000
ship	0.91	0.93	0.92	1000
truck	0.91	0.89	0.90	1000
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.84	0.84	0.84	10000

Fig. 26. Confusion Matrix for CIFAR-10.

VIII. DIFFERENT DESIGNS OF CLASSIFICATION HEAD

The design of the classification head in a Convolutional Neural Network is critical, dictating how the spatially arranged features are transformed for final prediction. The two primary methods for this transition are **Global Average Pooling** and the **Flatten** operation, which is followed by **Dense** layers. The conventional method for preparing feature maps for a Dense layer is to use the Flatten operation. This layer's sole purpose is reshaping: it takes a multi-dimensional tensor and simply unrolls all its elements into a single, long vector, while preserving every activation value. With the same hyperparameters as before and featuring a Flatten layer followed by a Dense(512) layer and the softmax which is common for both models. The results from the model trained using the Adam optimizer for 25 epochs reveal a common yet critical issue in CNN architecture: severe overfitting driven by an excessive number of parameters in the classification head. The training curves demonstrate Adam's characteristic efficiency and speed, but also highlight poor generalization. Both the Training and Validation Accuracy curves rise rapidly during the first 10 epochs. The most significant observation is the dramatic divergence starting around Epoch 12. The Validation Accuracy essentially plateaus and stagnates at approximately $78\% - 80\%$, while the Training Accuracy continues its steady climb, reaching almost 90% by Epoch 25.

The Architectural Flaw

The primary cause of this extreme overfitting lies in the structure of the classification head, which introduces a massive number of trainable parameters: This operation takes the multi-dimensional output of the final convolutional layer (e.g., $3 \times 3 \times 258$ feature maps) and converts it into a single, long 1D vector ($2,233$ nodes). Connecting this $2,233$-node vector to a subsequent Dense(512) layer creates an enormous weight matrix. The number of parameters in this single connection is roughly $2,233 \times 512 \approx \mathbf{1.2}$ million parameters.This high parameter count gives the model excessive capacity. The model uses this capacity to capture and memorize minute details and noise specific to the training set, rather than extracting robust features, resulting in excellent training performance but poor generalization on the validation set.

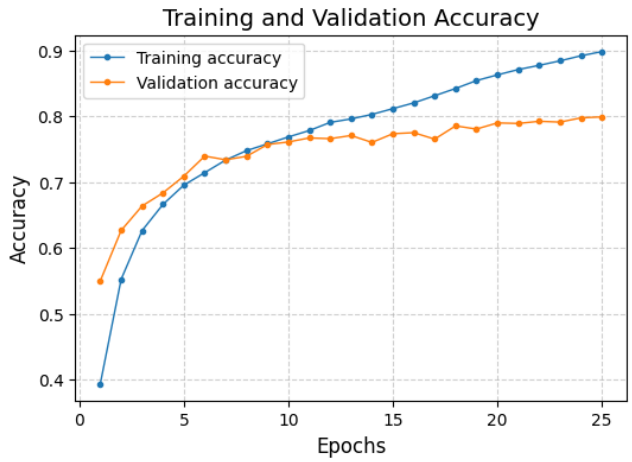


Fig. 27. Learning Curves for Training and Validation Accuracy.

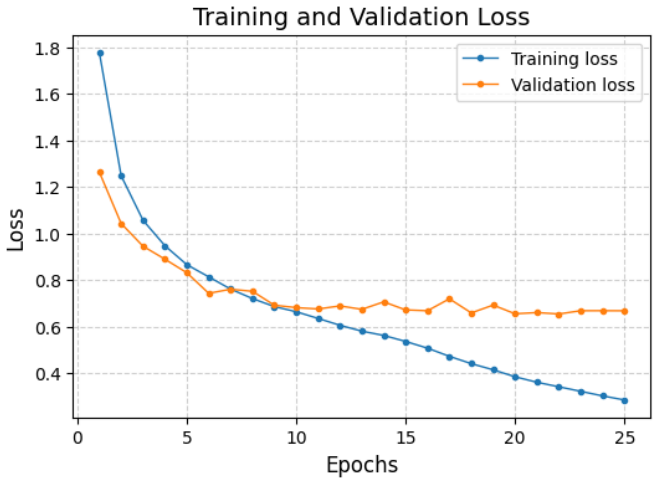


Fig. 28. Learning Curves for Training and Validation Loss.

Fixed Architecture for the overfitting issue

In order to compensate for the high parameter count of our new architecture, we added new external regularization. After each Conv/Pool layer we increase the **Dropout** to 30%.That significantly reduces co-adaptation of features in the convolutional layers. By randomly setting 30% of the feature detectors to zero in each layer, the network is forced to learn redundant and robust features, which is crucial for distinguishing between visually similar objects like cats and dogs.The first fully connected layer has an increased size of neurons **Dense(1024)** because we need the capacity to potentially learn complex, non-linear feature combinations for 10 classes. The expectation is that the heavy Dropout will prevent this high capacity from leading to memorization.The last **Dropout(0.5)** is a very critical fix that provides maximum uncertainty and regularization. It forces the network to ensure that any subset of half the features (the remaining 50% of neurons) can still accurately classify the input. This is a powerful countermeasure against the massive number of parameters introduced by the **Flatten** layer.The overall fix is the implementation of an aggressive and layered regularization strategy to manage the high capacity of the network, particularly the classification head.

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7900 - loss: 0.6071
Test accuracy: 78.90%
Elapsed time: 114.07982516288757 seconds
```

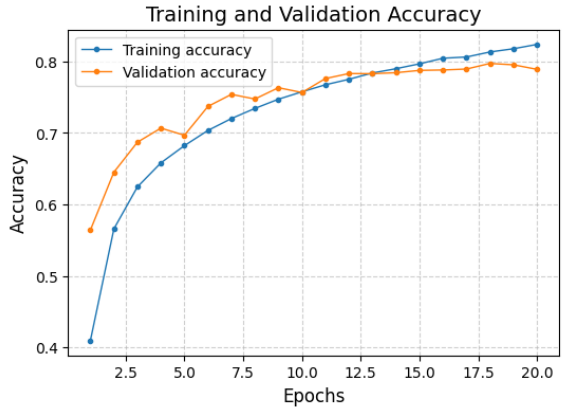


Fig. 29. Learning Curves for Training and Validation Accuracy.

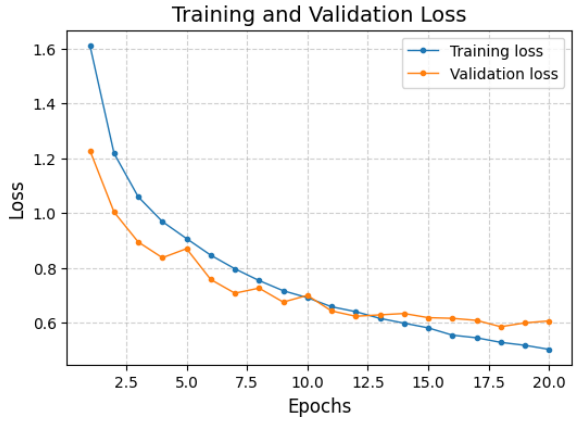


Fig. 30. Learning Curves for Training and Validation Loss.

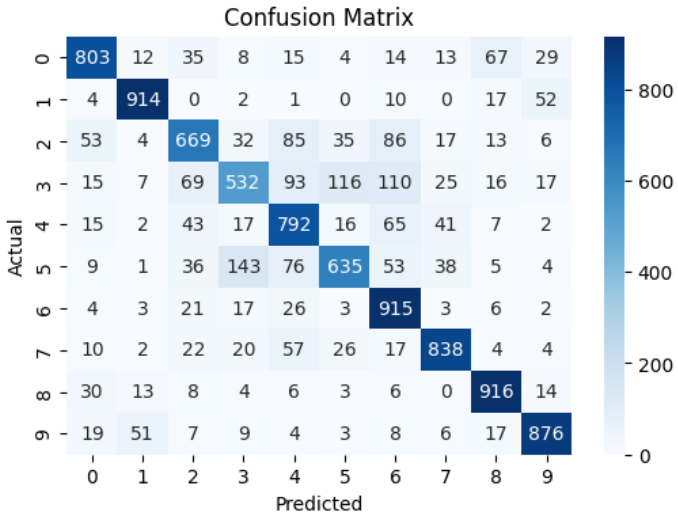


Fig. 31. Confusion Matrix for CIFAR-10.

Learning Curves Analysis

From the learning curves of our new model, we observe that our new regularization strategy has successfully mitigated the severe overfitting mentioned before. The Training Accuracy and Validation Accuracy curves track each other very closely until around Epoch 15. The gap between them is much smaller than in the previous run (where the gap was $\approx 10\% - 12\%$). This confirms that the aggressive Dropout is effectively preventing the model from memorizing the training data.The Validation Loss dips below the Training Loss early on and tracks closely, although the Validation Loss shows more volatility (spikes around Epoch 7 and 10), which is common with heavy regularization. The model is no longer severely overfitting and generalization is much better. The fact that the model achieved a Test Accuracy of 78.90% in only 20 epochs (with a training time of ≈ 114 seconds) demonstrates a significant level of computational efficiency from the Adam optimizer combined with the high capacity of the network.

Doubling the batch size

The effect of doubling the batch size for a flattened and dense classification head from 64 to 128 is primarily related to the optimization process and the properties of the gradient estimates. Our new batch size=128 means the gradient is calculated over more samples. This results in a less noisy or smoother estimate of the true gradient for the entire training dataset.This change successfully utilized the better parallelization capabilities of the hardware, providing a smoother and more stable estimate of the true gradient during backpropagation. Consequently, the model's performance saw a tangible lift in generalization, resulting in an accuracy increase from 78.9% to 79.63%

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7966 - loss: 0.5838
Test accuracy: 79.63%
Elapsed time: 77.24445343017578 seconds
```

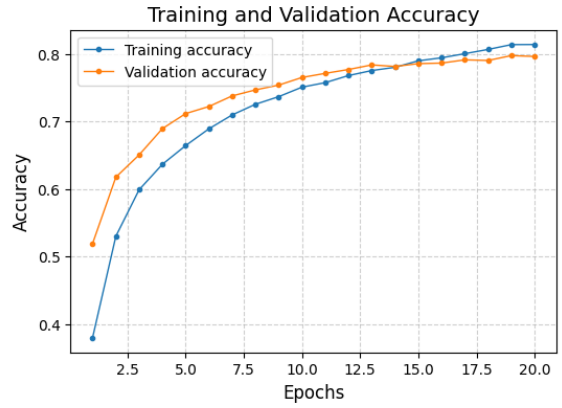


Fig. 32. Learning Curves for Training and Validation Accuracy.

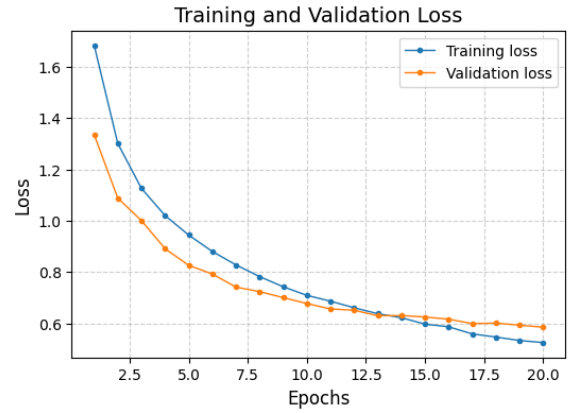


Fig. 33. Learning Curves for Training and Validation Loss.

IX. ENSEMBLE NETWORK OF CNN MODELS

An Ensemble Network (or Ensemble Model) is a machine learning technique where, instead of relying on a single model to make a prediction, multiple, independently trained models are used, and their predictions are combined to produce a final, improved decision.Our ensemble network consists of the three best models mentioned in the previous sections:

1) Convolutional Neural Network with Adam Optimizer: This model achieved 83% test accuracy.
2) Convolutional Neural Network with Stochastic Gradient Descent and Momentum: This model achieved 84% test accuracy.
3) Convolutional Neural Network with Adam Optimizer and Flatten-Dense Classification Head: This model achieved 80% test accuracy.

The primary reason for using an Ensemble Network is to reduce error and improve generalization in the final prediction.Each individual model makes different errors. When their predictions are combined, it is highly likely that the errors made by one model will be canceled out by the correct predictions of the other models. Our Ensemble Network achieved higher prediction accuracy than the best individual model within the set.The prediction from each model is aggregated using **Soft Voting** where the final combination rule is simply averaging the probability distributions.

```
1. Loading and Preprocessing of dataset CIFAR-10...
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-p
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step

2. Model Loading (.keras)...

3. Creation of Ensemble Network (Soft Voting)...
313/313 ━━━━━━━━━━━━━━━━━━━━ 62s 197ms/step
Test Accuracy Model 1: 0.8160
Test Accuracy Model 2: 0.8309
Test Accuracy Model 3: 0.8398
----------------------------------------
Test Accuracy of the FINAL ENSEMBLE: 0.8589
```

Fig. 34. Ensemble Network Accuracy.

The goal of an ensemble is to outperform the best individual component, and our network achieved this:

- **Best Individual Accuracy**:83.98%.
- **Final Ensemble Accuracy**:85.89%

The Soft Voting Ensemble provided an absolute improvement of 1.5% over the best single model. This successful lift confirms the fundamental principle of ensemble learning: By combining the strengths of the three models, we achieved a new state-of-the-art performance for our set of models, reaching a final test accuracy of 85.89% on the CIFAR-10 dataset. This suggests the individual models were diverse enough in their error patterns to make the Soft Voting combination beneficial.

X. COMPARISON OF PERFORMANCE BETWEEN DEEP LEARNING MODELS AND MACHINE LEARNING ALGORITHMS

K Nearest Neighbours and Nearest Centroid Classifiers

In the previous assignment we implemented two specific non-parametric, distance-based classifiers: the k-Nearest Neighbors (kNN) algorithm and the Nearest Centroid Classifier (NCC).The K-Nearest Neighbors algorithm operates on the principle of instance-based learning, classifying a new data point based on the majority class of its k closest neighbors in the training set and NCC calculates the mean image for every class (e.g., the average pixel value for all cat images) so in order to classify a new image, it calculates the distance to each class mean and chooses the closest one.The k-NN algorithm achieved 30% accuracy in the CIFAR-10 dataset,thus k-NN is only picking up on very coarse similarities in color and lighting, but it completely fails to understand concepts like edges, corners, or object structures.On the other hand, NCC's 27% accuracy reflects the fact that a mean image is a horrible representation of a complex class. The average "cat" image is blurry and contains none of the specific features needed to distinguish it from an average "dog" or "deer" image.It throws away all the meaningful variance required for discrimination. The mean vector destroys all variance and crucial distinguishing details, making it almost useless for classification.Even though k-NN is slightly better because it uses individual instances, not the average. A CIFAR-10 image is $32 \times 32 \times 3$ (RGB), resulting in a vector of 3,072 dimensions. In this high-dimensional space, the distance between data points becomes unreliable (known as the "curse of dimensionality").

Multi-Layer Perceptron

The MLP's $\mathbf{54}\%$ accuracy shows it's a significant improvement over the simple classifiers because the MLP is capable of learning non-linear feature combinations, whereas k-NN and NCC are limited to simple, linear distance comparisons in the input space.The MLP transforms the initial raw pixel input vector into a new, internal representation across multiple layers. Each layer combines the features from the previous layer in complex ways. While the MLP still struggles with the loss of spatial structure (due to flattening), its hidden layers are powerful enough to extract some general concepts and correlations from the pixel data, leading to a respectable $\mathbf{54}\%$ accuracy.This ability to learn complex relationships is what enables the MLP to recognize patterns that simple distance methods miss.

Key Difference of Convolutional Neural Networks

The CNN achieves high accuracy because it completely avoids the pitfalls of pixel-based distance metrics.Focusing on feature learning, the convolutional layers automatically learn a hierarchy of abstract features. The first layers find simple features (edges, colors), the middle layers find complex textures and parts (eyes, wheels), and the final layers recognize the full object.Also, by using pooling layers, the CNN achieves spatial invariance. A slight shift or rotation of the object does not significantly change the resulting feature map, allowing the classifier to recognize the object regardless of its exact position.On the other hand, k-NN only cares about the absolute distance in the pixel space, so a shifted image will be treated as very far away from the original. The k-NN algorithm will likely fail to find the original (or similar unshifted) image as a nearest neighbor, leading to a classification error.In conclusion, the performance disparity between the Convolutional Neural Network (CNN) and the simpler classifiers like k-NN and NCC stems from their fundamentally different approaches to handling the high-dimensionality and specific nature of image data.The SGD/Adam optimizer tunes the network's weights to maximize the separation between the output probability distributions of the classes,unlike k-NN that is a non-parametric and lazy learning algorithm and **non trainable** and just uses the train set as **instances**. It does not have any internal mechanism (like learnable kernels or pooling) to detect abstract features. This makes the CNN's output a representation optimized for the task of classification and outperforms all the other models. This fundamental shift from raw pixel-based distance to learned, invariant feature-based distance is the reason our CNN achieves an accuracy of $\mathbf{84}\%$ while MLP, k-NN and NCC remain stuck around 54% and 30% respectively.