

3RD MANDATORY ASSIGNMENT

RBF NEURAL NETWORKS AND CLUSTERING

Georgios Lazaridis

AEM: 4419



Aristotle University of Thessaloniki

January 2025

1 Introduction and Implementation

This project implements a **Radial Basis Function (RBF) Neural Network** to solve the Multi-Class Classification problem on the CIFAR-10 dataset. The primary objective is to construct a hybrid neural network that utilizes **K-Means Clustering** for the hidden layer and **Logistic Regression** for the output layer, and to compare its performance against distance-based baselines (k-NN, Nearest Class Centroid).

1.1 Task Definition and Data Pre-processing

To handle the high dimensionality of CIFAR-10 ($32 \times 32 \times 3$), the following pre-processing pipeline was applied before training:

- **Data Loading:** The full dataset (50,000 training images, 10,000 test images) was utilized.
- **Flattening & Scaling:** Images were flattened into 1D vectors (3072 features) and standardized using `StandardScaler` ($\mu = 0, \sigma = 1$).
- **Dimensionality Reduction (PCA):** Principal Component Analysis was applied to retain 90% of the variance, reducing the feature space significantly to facilitate efficient distance calculations.

1.2 Algorithms Implemented

1. RBF Neural Network (Custom Implementation):

Unlike standard implementations found in libraries, we developed a custom Python class `RBFNetwork` (located in `rbf.py`) to construct a hybrid learning architecture. This approach combines unsupervised clustering with supervised classification.

- **Architecture:** The network consists of three layers:
 - (a) **Input Layer:** Receives the PCA-reduced feature vectors.
 - (b) **Hidden Layer (Radial Basis Functions):** This layer performs a non-linear transformation. Instead of using learnable weights via backpropagation, the neurons (centers) are determined via **K-Means Clustering**.
 - (c) **Output Layer:** A linear classifier (**Logistic Regression**) that maps the hidden layer activations to the final class probabilities.
- **Mathematical Formulation:** The activation $\phi_j(x)$ of the j -th hidden neuron for an input vector x is calculated using the Gaussian Kernel:

$$\phi_j(x) = \exp(-\gamma \|x - c_j\|^2) \quad (1)$$

where c_j is the centroid of the j -th cluster found by K-Means, and γ controls the spread of the neuron's receptive field.

- **Algorithmic Logic (Code Commentary):** The custom class implements the standard Scikit-Learn interface with `fit` and `predict` methods:

- `fit(X, y)`: The training process is two-fold. First, K-Means is run on the input data X to fix the positions of the centers (`self.centers_`). Second, the input data is transformed into a new matrix where each feature represents the distance to a center. Finally, a Logistic Regression classifier (`self.clf_`) is trained on these new features against the labels y .
 - `predict(X)`: New data is projected onto the existing centers using the RBF kernel function, creating the activation map. The pre-trained Logistic Regression then predicts the class based on these activations.
2. **k-Nearest Neighbors (k-NN)**: A non-parametric baseline ($k = 1, k = 3$) using Euclidean distance.
 3. **Nearest Class Centroid (NCC)**: A simple linear classifier that assigns samples to the class of the nearest mean vector.

2 Experimental Results

We conducted experiments to tune the RBF network’s complexity and compared the optimal configuration against the baselines.

2.1 Experiment 1: RBF Network Sensitivity Analysis

We investigated the impact of the number of hidden neurons (RBF Centers) on classification accuracy and training time. The parameter γ was fixed at 0.001.

Table 1: Impact of Number of Centers on RBF Performance

RBF Centers	Training Time (s)	Test Accuracy	Observation
100 Centers	10.26s	38.02%	Fast, Underfitting
200 Centers	11.02s	39.76%	Balanced
500 Centers	24.93s	41.72%	Best Performance

As shown in Table 1, increasing the number of prototypes from 100 to 500 resulted in a strictly positive trend in accuracy (+3.7%), albeit with a doubling of the training time.

2.2 Experiment 2: Comparison with Baselines

The best-performing RBF model (500 centers) was compared against the k-NN and Nearest Centroid baselines on the same PCA-reduced data.

Table 2: Final Comparison: RBF vs Distance-Based Methods

Model Algorithm	Configuration	Accuracy
RBF Network (Ours)	500 Centers, PCA	41.72%
k-NN	$k = 1$, PCA	37.38%
k-NN	$k = 3$, PCA	35.34%
Nearest Class Centroid	PCA	27.75%

2.3 Visualizations

To gain deeper insight into the model’s behavior, we visualized the prediction errors and qualitative performance of the best RBF model (500 centers).

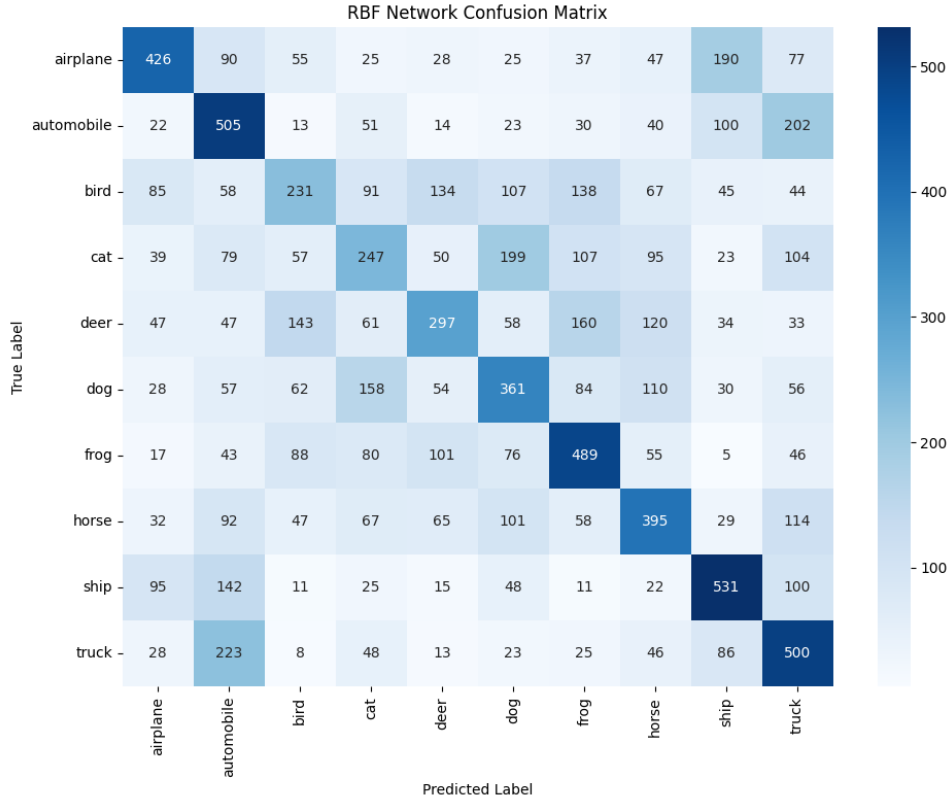


Figure 1: **Confusion Matrix (RBF - 500 Centers):** The diagonal elements represent correct predictions. We observe that the model struggles most with semantically similar animal classes (e.g., Cat vs. Dog, Deer vs. Bird), while vehicle classes (e.g., Ship, Truck) are classified with higher accuracy.

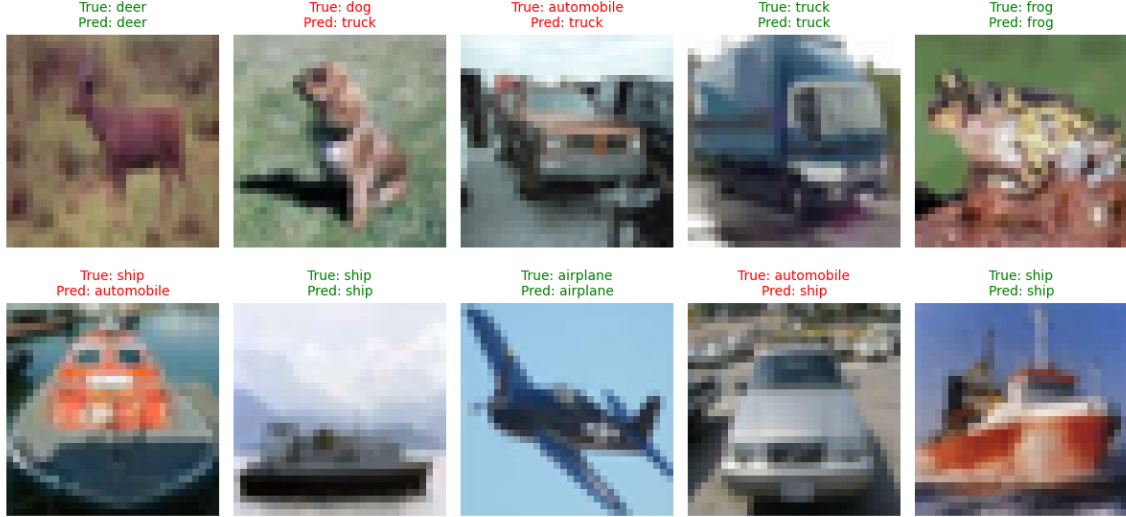


Figure 2: **Qualitative Results:** Randomly selected samples from the Test Set. **Top Row (Green):** Correct classifications demonstrating the model’s ability to recognize clear shapes. **Bottom Row (Red):** Incorrect classifications, highlighting cases where background noise or low resolution confused the RBF network (e.g., a blurry Ship misclassified as an Airplane).

3 Analysis and Discussion

3.1 RBF vs. k-NN: Generalization vs. Memorization

The experimental results demonstrate that the **RBF Network significantly outperforms k-NN** (+4.34% improvement over $k = 1$).

- **k-NN Limitations:** k-NN is a "lazy learner" prone to noise. In high-dimensional spaces like CIFAR-10, pixel-wise distance to a single neighbor ($k = 1$) is often unreliable due to background noise or slight variations.
- **RBF Strengths:** By using K-Means, the RBF network learns 500 "prototypical" features (centroids) that represent the data distribution. The final decision is based on the distance to these stable prototypes rather than noisy individual samples, leading to smoother decision boundaries and better generalization.

As confirmed by the Confusion Matrix in Figure 1, the RBF network effectively separates distinct classes like vehicles but shows expected confusion between similar animal species, a limitation inherent to the low resolution of CIFAR-10

3.2 The Role of PCA

The application of PCA was critical for the feasibility of the experiments. Running K-Means and k-NN on raw 3072-dimensional vectors would be computationally prohibitive (Curse of Dimensionality). PCA compressed the information while retaining the structural variance necessary for the RBF kernel to distinguish between classes.

4 Conclusion

The study confirms that a hybrid RBF Network serves as a powerful intermediate between simple distance-based classifiers and deep neural networks. By aggregating information into 500 centers, the RBF model achieved an accuracy of **41.72%**, successfully surpassing the baseline benchmarks.

5 Code Implementation

The implementation was developed in Python using the following libraries:

- **Scikit-Learn:** For KMeans, LogisticRegression, PCA, and StandardScaler.
- **TensorFlow/Keras:** Exclusively for loading the CIFAR-10 dataset.
- **Matplotlib/Seaborn:** For visualization of Confusion Matrices.

To handle the computational load on the full dataset, a caching mechanism (`joblib`) was implemented to save/load trained models and avoid redundant re-training.

6 GitHub Repository

The complete source code and necessary configuration files for reproducing the experiments are publicly available on the GitHub repository:

7 GitHub Repository

The complete source code and necessary configuration files for reproducing the experiments are publicly available on the GitHub repository:

- <https://github.com/lazoulios/image-classification-rbf-network>
- **Contents:** The repository includes the `rbf.py` (custom RBF Network implementation), `main.py` (training execution and experiments), `visualize.py` (graph generation), and `utils.py` (helper functions) scripts, the `requirements.txt` dependency list, and the `README.md` setup instructions.