Matteo Mazzini matteo.mazzini@estudiantat.upc.edu
Noemi Cicala noemi.cicala@estudiantat.upc.edu

Exploring Image Classification Using CNNs and Kernel Methods

Advanced Machine Learning
Semester 2024-2025

1 Introduction

The integration of Convolutional Neural Networks (CNNs) with Support Vector Machines (SVMs) has emerged as a promising approach to enhance classification tasks by leveraging the strengths of both techniques. CNNs excel in extracting deep, hierarchical features from data, while SVMs are known for their robust performance in finding optimal decision boundaries, especially with kernel methods. Recent researches, from which we were inspired, highlight the advantages of this combination.

2 Related and inspiring works

[10] Tang explored an alternative approach to the standard practice of employing the softmax activation function and cross-entropy loss in deep learning models. The study demonstrates a small but consistent performance improvement by replacing the softmax layer with a linear Support Vector Machine (SVM), specifically utilizing L2-SVMs. The model workflow involves initially training a deep convolutional neural network (CNN) using supervised or unsupervised objectives to extract robust and invariant latent representations. These learned hidden variables are subsequently fed as inputs to the linear or kernel SVMs for classification. A significant advantage of this methodology lies in its utilization of the L2-SVM loss function instead of the traditional hinge loss. Since the L2-SVM loss is differentiable, it allows for effective backpropagation of gradients to fine-tune lower-level features in the CNN according to the SVM's objective, ensuring that feature extraction aligns more closely with the classification task. Additionally, the L2-SVM loss penalizes errors more heavily, contributing to better generalization. The proposed approach showcases (Table 1) superior

performance compared to softmax-based networks across widely used datasets (e.g. MNIST, CIFAR-10 ...)

	ConvNet+Softmax	ConvNet+SVM
Test error	14.0%	11.9%

Table 1: Comparisons of the models in terms of % error on test set [CIFAR-10]

[1] CNNs, which are built with layers of neurons possessing learnable parameters, traditionally utilize Softmax in the final layer to perform classification tasks. However, this research challenged the norm by employing an SVM as the final classifier. A visualization of the used CNN is shown by Fig.1

Figure 1: A CNN with two convolutional layers (5×5 kernels, stride 1), max-pooling layers, and a fully connected classifier with 1024 hidden neurons and dropout.

The results, that have been obtained, are defined in Table 2:

Dataset	CNN-Softmax	CNN-SVM
MNIST	99.23%	99.04%
Fashion-MNIST	91.86%	90.72%

Table 2: Comparisons of the models in terms of test accuracy using the two well-known dataset.

These findings indicate that while SVMs offer competitive performance, the slight drop in accuracy may be attributed to the simplicity of the base CNN architecture. Another significant limitation of using SVMs in this context is their inherent design for binary classification: infact extending the approach to multi-class problems requires employing a one-versus-all strategy, which may introduce additional complexity and computational overhead. So the author suggests that improvements in performance could potentially be achieved by adopting more sophisticated CNN architectures and employing advanced data preprocessing techniques.

[7] This paper introduces a hybrid Convolutional Neural Network–Support Vector Machine model designed to combine the strengths of two powerful classifiers for improved pattern recognition. In this framework, CNNs act (Fig.2) as trainable feature extractors, autonomously identifying meaningful and distinguishable characteristics from raw images, while SVMs serve as the final

recognizers, classifying the extracted features with precision.

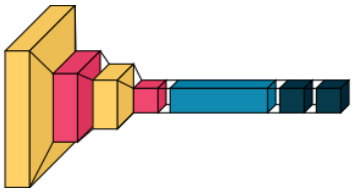


Figure 2: A convolutional neural network with two feature map layers (5×5 kernels, subsampling ratio 2) and a fully connected MLP for classification.

The study emphasizes that while achieving 100% recognition accuracy in handwritten digit classification remains unrealistic, enhancing reliability is a more practical and impactful objective for real-world applications. A key focus of this research lies in feature extraction—ensuring that the derived features capture maximum inter-class distinctions while preserving intra-class consistency. Testing on the MNIST dataset validates the feasibility and effectiveness of the hybrid model, demonstrating its capability to automatically extract high-quality features via CNNs and achieve reliable pattern recognition through SVMs, as shown by Table 3.

	CNN	Hybrid CNN-SVM
Training	0.28%	0.11%
Testing	0.59%	0.19%

Table 3: Comparisons of the models in terms of the error rate on MNIST dataset.

[4] This research proposes an innovative hybrid model that integrates Convolutional Neural Networks with kernel Support Vector Machines for enhanced classification performance. The model introduces a fully differentiable Radial Basis Function (RBF) layer as part of the kernel SVM, seamlessly integrating it into the CNN pipeline. This RBF layer allows the model to exploit the feature extraction power of deep networks while retaining the classification precision of kernel SVMs. A CNN architecture comprising four convolutional layers and one fully connected layer extracts abstract features (Fig.3), which are passed to the RBF layer for classification.

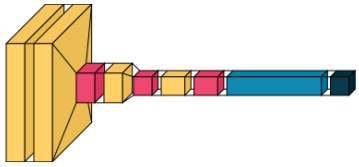


Figure 3: A CNN with four convolutional layers (kernels: 5×5, 3×3, 7×7), ReLU activations, and three max-pooling layers for feature reduction, followed by a fully connected layer for classification.

The classifier layer utilizes a Gaussian RBF, represented by the equation:

$$\hat{y}(x) = \sum_{i=1}^{N} a_i \exp\left(-\frac{\|x - \mu_i\|^2}{\sigma_i^2}\right) + b$$

One of the study's key contributions is demonstrating that this hybrid CNN-kSVM model can be trained end-to-end using gradient descent methods, optimizing both the CNN's feature extraction and the SVM's classification objectives simultaneously. While deeper neural networks enhance feature extraction and lead to linearly separable representations, they also increase computational demands. This model mitigates these challenges by fusing CNNs and kernel SVMs; experimental results (Table 4) validate the architecture's effectiveness, illustrating its potential for applications requiring precise and robust classification performance.

Model	Test Accuracy
Baseline CNN	80.65%
CNN with RBF layer, N=10	80.65%
CNN with RBF layer, N=20	80.95%
CNN with RBF layer, N=50	81.65%

Table 4: Comparisons of the models in terms of test accuracy [CIFAR-10]

3 Towards the improvement of the hybrid model

Compared to previous works and the recommendations provided, we deemed it possible to improve this new proposed hybrid model by using:

- some pre-processing tecniques
- a more sophisticated CNN, the ResNet-18
- the Random Fourier Features, to accelerate the training of kernel machines

Before proceeding with the implementation and analyzing the obtained results, it is essential to shed light on these concepts, highlighting their strengths that led us to consider them.

3.1 Pre-processing tecniques

3.1.1 Data augmentation

[6] Data augmentation is a widely used technique in machine learning, particularly for training deep learning models with limited datasets. It involves artificially increasing the size and diversity of the training data by applying various transformations to the original samples, such as flipping, cropping, rotating, scaling, translation, and noise addition (Fig.4) These methods aim to

extract additional information from the existing dataset, enhance generalization, and reduce the risk of overfitting. Data augmentation techniques can be broadly categorized into geometric transformations, which modify the spatial attributes of an image and photometric transformations, which alter color properties like RGB channels.They are implemented either offline, by pre-generating augmented data, or online, performed dynamically during training.

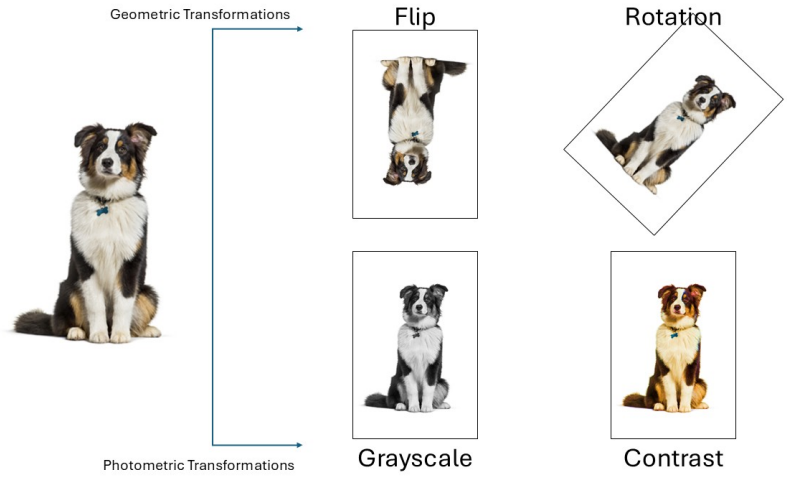


Figure 4: Examples of transformations

In addition to commonly used methods, there are more advanced techniques, such as kernel filtering and image mixing approaches, such as random cropping and patching (Fig.5). Random cropping and patching involve sampling sections from several images to create a new composite image, while random erasing removes a random portion of an image. These excluded techniques provide further variability but are often less commonly utilized due to specific application requirements or the complexity they introduce.
While augmentation is essential for improving model robustness and handling class imbalance, it also requires careful consideration of computational and memory constraints. Choosing appropriate augmentation strategies is critical, as poorly selected methods can lead to undesirable outcomes, such as increased overfitting in cases with extremely small datasets. In general, data augmentation plays a pivotal role in boosting the performance of models in various computer vision tasks, including image classification, object detection, and medical image analysis.

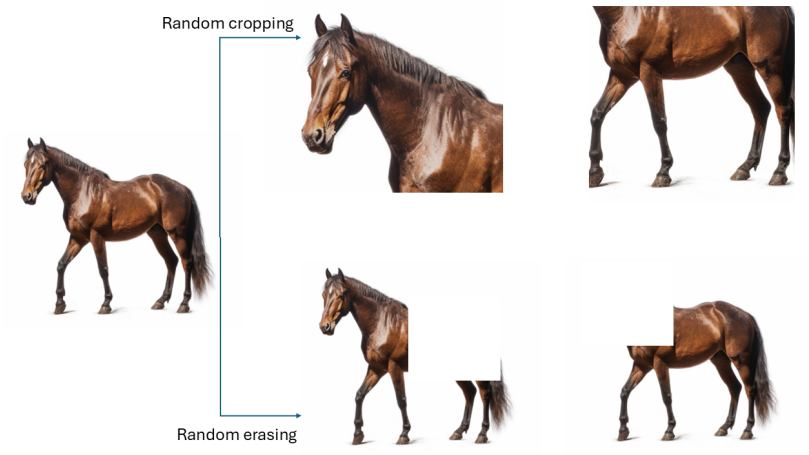


Figure 5: Examples of advanced transformations

3.1.2 Normalizing pixel values per channel

[2] Normalizing pixel values enhances the stability and speed of training, making it an essential step in modern deep learning workflows.

$$x' = \frac{x - \text{mean}}{\text{std}}$$

where *mean* and *std* represent the mean and standard deviation of the RGB channels (red, green, and blue), calculated over an image dataset.
This pre-processing technique is useful because:

- Centers data around zero: By subtracting the mean, pixel values are centered around zero. This ensures that the features passed to the neural network do not have large biases, making optimization more efficient.
- Scales data uniformly: Dividing by the standard deviation scales the pixel values to have a unit variance. This prevents one channel from dominating the others if it has a wider range of values.
- Faster convergence: By standardizing the input, the network starts with inputs that are better aligned with its initialization, allowing the optimizer to converge faster.

3.2 ResNet-18

[3] Deeper neural networks, while promising improved accuracy, are often more challenging to train due to issues such as vanishing or exploding gradients. These problems can hinder convergence, even with advancements like normalized initialization and intermediate normalization layers, which enable convergence for

networks with tens of layers. Another critical challenge, termed the "degradation problem," emerges when increasing network depth: the accuracy saturates and then degrades, even without overfitting. This degradation results in a higher training error for deeper networks.

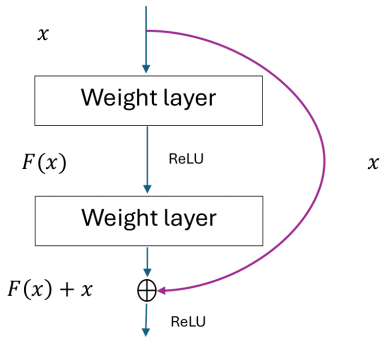


Figure 6: Residual learning (1)

To address these challenges, the residual learning framework was introduced. This framework reformulates the task of learning layers to focus on residual mappings rather than direct mappings. Specifically, given an underlying mapping $H(x)$, the framework encourages the layers to fit a residual mapping $F(x) = H(x) - x$, which is then combined with the input x to produce $F(x) + x$.

This approach hypothesizes that optimizing the residual mapping is easier than optimizing the original, unreferenced mapping. The implementation of residual learning involves shortcut connections (violet line shortcut in Fig.6), which skip one or more layers and perform identity mapping. These connections ensure that the input is directly added to the output of the stacked layers, simplifying the optimization. When input and output dimensions differ (dotted line shortcut in Fig.7), the following strategies are employed:

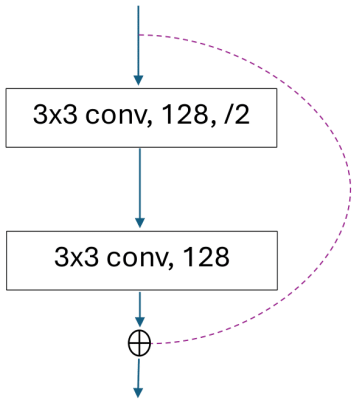


Figure 7: Residual learning (2)

- Option A: Identity mapping with zero-padding to match dimensions, introducing no extra parameters.
- Option B: Projection shortcut using 1 X 1 convolutions to match dimensions.

[9] The architecture of ResNet-18 consists of a total of 18 layers where 17 of them are convolutional layers an the remaining one is a fully connected layer (Fig.8); there is also an additional softmax layer for classification.
The convolutional layers utilize 3×3 filters which follow two core design rules:

- layers with the same output feature map size have the same number of filters.
- when the feature map size is halved, the number of filters is doubled to preserve time complexity per layer.

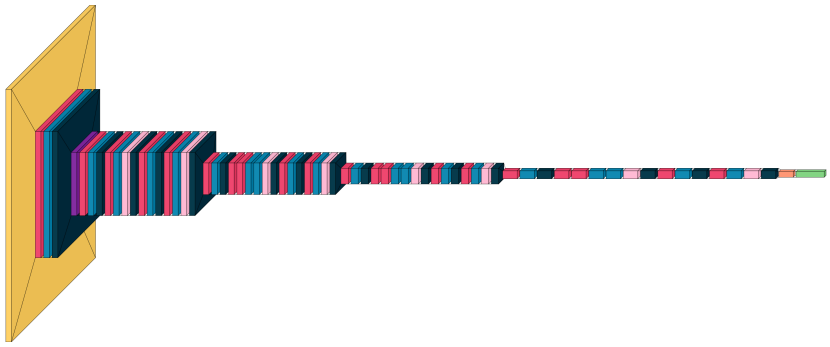


Figure 8: Architecture of a ResNet-18

This elegant architecture enables deep feature extraction while maintaining computational efficiency.

3.3 Random Fourier Features

[8] Rahimi and Recht proposed an efficient approximation of kernel functions to enable scalable machine learning with large datasets.
Consider a learning problem with data and targets

$$\{(x_n, y_n)\}_{n=1}^{N} \quad \text{where} \quad x_n \in X \text{ and } y_n \in Y$$

The traditional kernel methods compute:

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_V$$

where $\phi(\cdot)$ maps data to a high-dimensional feature space V (dimension of V may be high or even infinite).This approach, while powerful, becomes computationally expensive as the dataset grows. To avoid this limitation, their propose was to introduce a randomized feature map :

$$z : \mathbb{R}^D \to \mathbb{R}^R \quad \text{with} \quad R \ll N$$

such that

$$k(x, y) \approx z(x)^\top z(y)$$

Using Bochner's theorem, which states that a shift-invariant kernel $k(x, y) = k(x - y)$ is the Fourier transform of a non-negative measure $p(\omega)$, they defined $z(\cdot)$ as:

$$z(x) = \sqrt{\frac{2}{R}} \begin{pmatrix} \cos(\omega_1^\top x + b_1) \\ \cos(\omega_2^\top x + b_2) \\ \vdots \\ \cos(\omega_R^\top x + b_R) \end{pmatrix}$$

where $\omega \sim p(\omega)$, $b \sim Uniform[0, 2\pi]$ and R is the number of random samples. For Gaussian kernels, $\omega \sim N(0, 2\gamma I)$, and the resulting approximation satisfies:

$$k(x, y) = \exp(-\frac{\|x - y\|^2}{2\sigma^2}) \approx z(x)^\top z(y)$$

This randomized mapping transforms the problem into a linear one in the R-dimensional space, allowing the target function $f_t(x)$ to be expressed as:

$$f_t(x) = \sum_{n=1}^{N} \alpha_n k(x_n, x) \approx \beta^\top z(x), \quad \text{where} \quad \beta \in \mathbb{R}^R$$

reducing the computational cost while retaining the expressiveness of kernel methods.

4 Motivation of the work

The approach in [10] can be implemented as a standard CNN by modifying the loss function to align with the primal SVM problem. Training is performed through backpropagation, simultaneously updating both the network parameters and the hyperplane weights.

While this approach is valid for linear SVMs, extending it to kernel SVMs is not straightforward. Integrating kernel methods within the CNN framework presents significant challenges. Specifically, kernel SVMs require calculating the dot product between all pairs of data points in the feature space, resulting in a kernel matrix that grows quadratically with the number of samples. This pairwise computation is computationally expensive and not well suited for CNN architectures. Additionally, kernel methods are challenging to integrate with the layer-wise training and gradient-based optimization strategies used in CNNs.

[3] proposes a solution to this problem by training a CNN and an SVM separately. In this approach, once the CNN has converged, the feature embeddings produced for each image can be utilized to train a kernel SVM in the dual formulation. This method results in a sparse and convex optimization problem but scales poorly with large datasets since the computational cost of kernel SVMs is determined by the construction of the kernel Gram matrix, which has a time complexity of $\mathcal{O}(n^2)$, where n is the number of samples. Additionally, this approach eliminates the possibility of having a unified model trained simultaneously, which may lead to inefficiencies in scenarios where joint optimization could offer performance benefits.

[4] introduces a novel and interesting method for integrating kernel SVMs directly into a CNN. In this approach, the kernel is computed between the embedding vector and a set of Gaussian centers, instead of using the traditional support vectors derived from the dataset. Unlike standard SVMs, these Gaussian centers are not fixed points from the dataset but rather trainable parameters that are optimized through backpropagation.

This approach offers greater flexibility compared to traditional SVMs because the centers can dynamically adjust to the data distribution, improving the model's ability to learn complex decision boundaries. However, this flexibility comes with trade-offs. Since the Gaussian centers are trainable, the hyperplane constructed by the kernel SVM uses different information than the fixed support vectors in classical SVMs. This change could impact interpretability and requires careful tuning.

Additionally, for complex tasks, the number of Gaussian centers required to accurately model the decision boundaries may be very high, making the approach computationally infeasible for large datasets or tasks with highly intricate feature spaces.

Our proposed method builds upon the idea of integrating an SVM into a CNN, as described in [10], but aims to replicate the behavior of a kernel SVM more faithfully. To achieve this, we utilize Random Fourier Features (RFF). This technique allows us to approximate the embedding vector in the high-dimensional space induced by the kernel function.

Once this high-dimensional representation is obtained, instead of relying on the traditional dual problem, where kernel similarities are computed via dot products, we solve the primal problem. This enables us to construct separating hyperplanes that account for the high-dimensional representation of the vectors, effectively achieving non-linear separation in the original input space.

By adopting this approach, we ensure the ability to replicate the benefits of kernel SVMs while maintaining computational efficiency. Additionally, our method introduces a new hyperparameter, D, which corresponds to the number of Random Fourier Features used in the approximation. This parameter must be tuned carefully, as it directly affects the quality of the high-dimensional representation and the overall performance of the model.

The quality of the kernel approximation improves as D increases, following the relation $O\left(\frac{1}{\sqrt{D}}\right)$. However, increasing D also raises the computational and memory cost, as the transformed features have dimensionality proportional to D.

5 Experimental Setup and Results

The code utilized in this project is available in the GitHub repository. It can be accessed in the directory `<AML_project_Noemi_Matteo>` at the following link: https://github.com/noemicic/AML_project_Noemi_Matteo.

Data Pre-processing For our work, we used the well-known CIFAR-10 dataset. We applied pixel normalization using the standard mean and standard deviation values specific to this dataset: mean $= [0.4914, 0.4822, 0.4465]$ and std $= [0.2023, 0.1994, 0.2010]$. The CIFAR-10 dataset consists of 50,000 images in the training set and 10,000 images in the test set, both evenly distributed across 10 classes. We further split the training dataset into training and validation subsets using an 80%-20% ratio, resulting in 40,000 training images, 10,000 validation images, and 10,000 test images.

Simply standardizing the pixel values was not sufficient, as ResNet is a powerful model capable of capturing intricate image details. Without additional measures, the CNN could achieve near-perfect accuracy in just a few iterations, leading to potential overfitting. To address this, we applied data augmentation exclusively to the training set. While the number of training images remained fixed, we randomly performed the following transformations on each image: horizontal flipping, rotation, and cropping. These augmentations significantly improved the generalization capability of our model. The validation and test images remained unchanged, with only pixel standardization applied.

Novel Architecture In this experiment, we constructed a convolutional neural network (CNN) where, after the CNN extracts features and produces the embedding vector, the vector undergoes random Fourier feature (RFF) transformation before being passed to the fully connected layer. The RFF transformation maps the vector to a higher-dimensional representation $\phi(x)$, defined as follows:

$$\phi(x) = \sqrt{\frac{2}{D}} \cos(\mathbf{W}x + \mathbf{b}),$$

where: $\mathbf{W} \in \mathbb{R}^{D \times d}$ is a matrix of weights sampled from a normal distribution $\mathcal{N}(0, 2\gamma\mathbf{I})$, $\mathbf{b} \in \mathbb{R}^D$ is a bias vector sampled uniformly from $[0, 2\pi]$, D is the number of random Fourier features, d is the dimensionality of the input feature vector x (512 in the ResNet case), γ is related to the bandwidth σ of the radial basis function kernel by the relation:

$$\gamma = \frac{1}{2\sigma^2}.$$

The scaling factor $\sqrt{\frac{2}{D}}$ is applied to normalize the RFF mapping. The weight matrix and bias of the RFF are sampled at the beginning of the training process and remain fixed throughout, unlike the other weights in the CNN, which are updated during training. The resulting higher-dimensional representation $\phi(x) \in \mathbb{R}^D$ captures the relationship with the radial basis function kernel. In this experiment, we focused exclusively on using random Fourier features to approximate the RBF kernel. The transformed vector $\phi(x)$ is then passed to the fully connected layer for classification.

Therefore, the decision boundaries for the multiclass case are defined by:

$$f(x) = \mathbf{W}\phi(x) + \mathbf{b},$$

where $\mathbf{W} \in \mathbb{R}^{K \times D}$ is the weight matrix learned during training, with each row $\mathbf{w}_k$ representing the weights for the hyperplane of class k and $f(x) \in \mathbb{R}^K$ contains the scores for all K classes.

The classification decision is made by selecting the class with the highest score:

$$\text{Class}(x) = \arg\max_k f_k(x),$$

where $f_k(x)$ is the score for class k.

Finally, the L2-SVM loss function for the multiclass case is applied to optimize the model. The total loss consists of a squared hinge loss and an L2 regularization term, defined as:

$$Loss = C \cdot \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \max(0, 1 - \mathbf{y}_{i,k} f_k(x_i))^2 + \frac{1}{N} \|\mathbf{W}\|_F^2$$

where $\mathbf{y}_{i,k}$ is the one-hot encoded target ($+1$ for the correct class, -1 otherwise), $f_k(x_i)$ is the predicted score for class k of sample i, C controls the trade-off between maximizing the margin and minimizing classification errors and $\|\mathbf{W}\|_F^2$ provides the L2 regularization.

Hyperparameter Tuning After constructing our model, we evaluated its effectiveness by comparing it with two other models: a CNN with standard softmax classification and a CNN with soft margin SVM-based classification. To achieve optimal performance for each model, we conducted comprehensive hyperparameter tuning, exploring various parameter values across the different architectures. Each CNN was trained for 50 epochs.

Computational Resources To train the implemented CNNs, we utilized two different machines. The first machine was equipped with an NVIDIA GeForce RTX 2050 GPU featuring 4 GB of memory and 2,048 CUDA cores. The second machine utilized an NVIDIA GeForce GTX 1650 Ti with Max-Q Design, which also featured 4 GB of memory but had 1,024 CUDA cores.

5.1 ResNet + Softmax

In the case of the CNN with softmax classification, the only hyperparameter to tune was the dropout rate. Dropout was introduced in the basic block of the network, specifically after the ReLU activation of the first convolutional layer, and another dropout layer was added before the fully connected layer. For the sake of simplicity, we used the same dropout rate for both layers in the architecture. The values tested for the dropout rate were as follows: [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4] and the results are shown in table 5.

Dropout	Train accuracy	Val accuracy	Test accuracy
0.1	92.55 %	89.05 %	88.04 %
0.15	91.54 %	88.62 %	87.91 %
0.2	90.82 %	88.37 %	87.76 %
0.25	90.17 %	88.73 %	88.22 %
0.3	**89.41 %**	**89.09 %**	**88.63 %**

Table 5: Performance of the ResNet+Softmax model for different values of dropout.

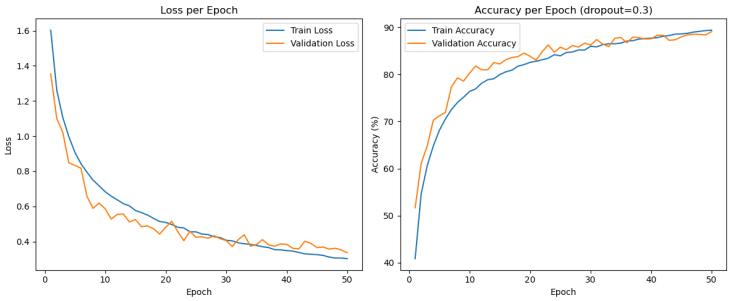


Figure 9: Training and validation performance metrics of the **best model**, the one with dropout rate of 0.3. The left panel shows the training and validation loss per epoch; the right panel illustrates the training and validation accuracy per epoch.

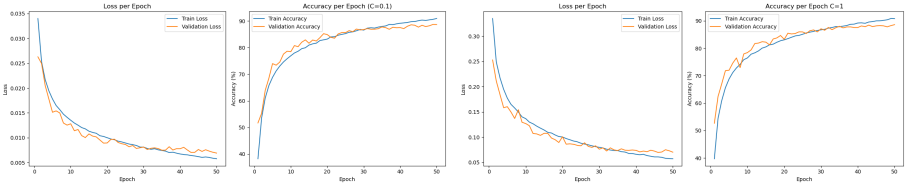
5.2 ResNet + SVM (End-to-End)

For the architecture of this CNN, we applied normalization to the feature embedding vector to ensure compatibility with the input requirements of the SVM. Specifically, normalization helps maintain consistent magnitudes in the feature space, which is crucial for the correct functioning of the SVM classifier.

Regarding the choice of hyperparameter values, due to computational resource constraints and limited time availability, we restricted the dropout rate to two intermediate values, [0.15, 0.3], which had previously shown good performance. For each of these dropout rates, we tested the following values of the regularization parameter C: [0.1, 1, 10, 100] and the results are shown respectfully in tables 6 and 7.

C	Train accuracy	Val accuracy	Test accuracy
0.1	**90.91 %**	**88.63 %**	**88.33 %**
1	**90.78 %**	**88.58 %**	**88.05 %**
10	90.89 %	88.11 %	88.22 %
100	90.86 %	88.36 %	87.85 %

Table 6: Performance of the ResNet+SVM model with a dropout rate of 0.15 for different values of C.



(a) Training and validation performance metrics per epoch with C=0.1.

(b) Training and validation performance metrics per epoch with C=1.

Figure 10: Comparison of training and validation performance metrics for the **best models**

C	Train accuracy	Val accuracy	Test accuracy
0.1	88.59 %	88.75 %	87.60 %
1	88.60 %	88.04 %	87.85 %
10	**88.62 %**	**88.64 %**	**87.91 %**
100	88.64 %	87.97 %	87.47 %

Table 7: Performance of the ResNet+SVM model with a dropout rate of 0.3 for different values of C.

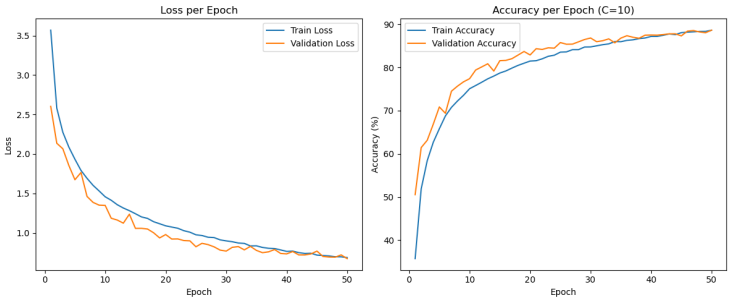


Figure 11: Training and validation performance of the **best model**, the one with dropout rate of 0.3 and C of 10.

5.3 ResNet + RFF Kernel SVM (End-to-End)

For this architecture, normalization of the embedding layer was maintained as in the previous configuration.

Regarding the hyperparameters, the dropout rate was fixed at 0.2 for simplicity and to reduce the complexity introduced by the numerous additional hyperparameters in this implementation. Dropout was applied only after the ReLU activation in the first convolutional layer.

The number of dimensions D for the random Fourier features (RFF) was set to four times the size of the ResNet embedding, resulting in $D = 2048$. For each value of γ in the range $\{0.1, 0.005, 0.003, 0.002, 0.001\}$, we tested the regularization parameter C over the predefined range $\{0.1, 1, 10, 100\}$ as it's shown from tables 8, 9, 10, 11 and 12.

C	Train accuracy	Val accuracy	Test accuracy
0.1	82.06 %	83.47 %	83.10 %
1	75.84 %	78.98 %	78.28 %
10	**84.53 %**	**84.10 %**	**83.55 %**
100	55.02 %	59.68 %	59.98 %

Table 8: Performance of the ResNet+RFF+SVM model with a dropout rate of 0.2, RFF dimensions of 2048 and $\gamma = 0.1$ ($\sigma^2 = 5$) for different values of C.

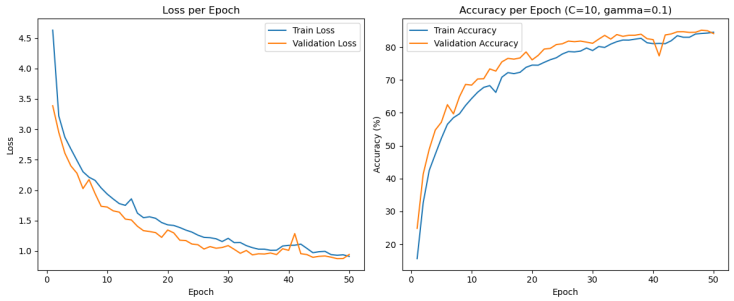


Figure 12: Training and validation performance of the **best model**, the one with C of 10.

C	Train accuracy	Val accuracy	Test accuracy
0.1	88.82 %	87.52 %	87.36 %
1	**89.09 %**	**87.91 %**	**87.86 %**
10	89.55 %	87.74 %	86.99 %
100	90.10 %	87.55 %	87.44 %

Table 9: Performance of the ResNet+RFF+SVM model with a dropout rate of 0.2, RFF dimensions of 2048 and $\gamma = 0.005$ ($\sigma^2 = 100$) for different values of C.

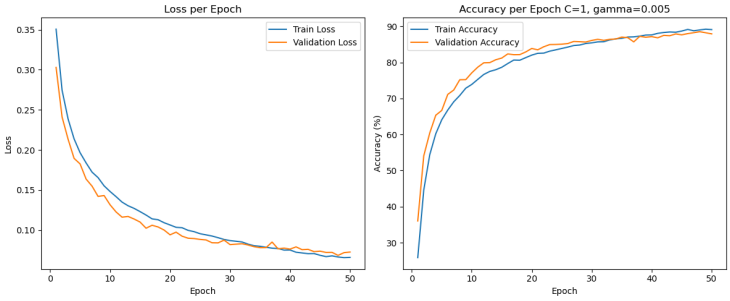


Figure 13: Training and validation performance of the **best model**, the one with C of 1.

C	Train accuracy	Val accuracy	Test accuracy
0.1	89.05 %	87.80 %	86.90 %
1	89.29 %	87.80 %	87.91 %
10	**89.75 %**	**87.82 %**	**87.53 %**
100	90.34 %	87.67 %	87.22 %

Table 10: Performance of the ResNet+RFF+SVM model with a dropout rate of 0.2, RFF dimensions of 2048 and $\gamma = 0.003$ ($\sigma^2 = 166.7$) for different values of C.

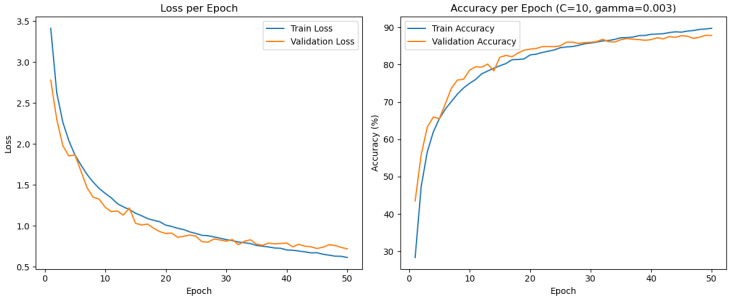


Figure 14: Training and validation performance of the **best model**, the one with C of 10.

C	Train accuracy	Val accuracy	Test accuracy
0.1	89.20 %	87.21 %	86.75 %
1	89.51 %	87.93 %	87.19 %
10	**89.55 %**	**87.92 %**	**87.50 %**
100	89.92 %	87.59 %	87.40 %

Table 11: Performance of the ResNet+RFF+SVM model with a dropout rate of 0.2, RFF dimensions of 2048 and $\gamma = 0.002$ ($\sigma^2 = 250$) for different values of C.

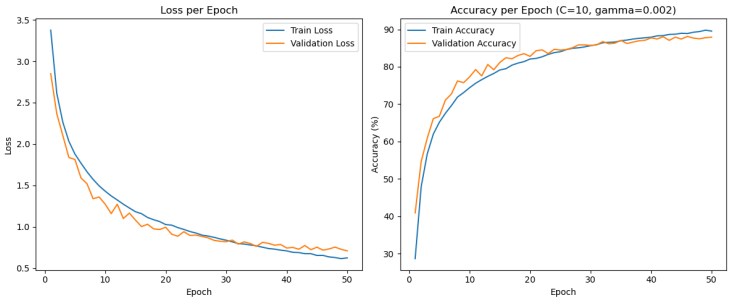


Figure 15: Training and validation performance of the **best model**, the one with C of 10.

C	Train accuracy	Val accuracy	Test accuracy
0.1	89.12 %	87.84 %	87.29 %
1	89.45 %	87.61 %	87.80 %
10	**89.26 %**	**87.93 %**	**87.84 %**
100	89.99 %	87.94 %	87.60 %

Table 12: Performance of the ResNet+RFF+SVM model with a dropout rate of 0.2, RFF dimensions of 2048 and $\gamma = 0.001$ ($\sigma^2 = 500$) for different values of C.

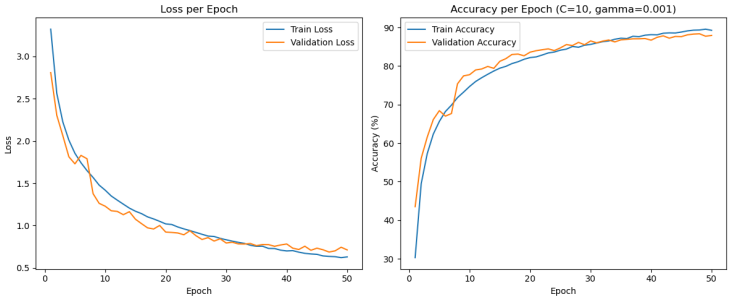
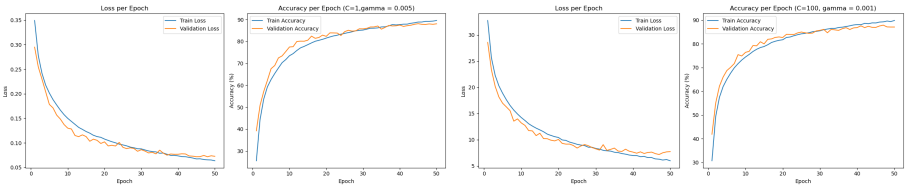


Figure 16: Training and validation performance of the **best model**, the one with C of 10.

Finally, based on the best-performing models obtained from this setup, we increased the number of RFF dimensions to $D = 4096$ to evaluate whether a significant performance improvement could be observed.

C	γ	**Train accuracy**	**Val accuracy**	**Test accuracy**
1	0.005	89.09 %	87.91 %	87.86 %
100	0.001	90.10 %	87.55 %	87.44 %

Table 13: Performance of the ResNet+RFF+SVM model with a dropout rate of 0.2, RFF dimensions of 4096 for different C and γ values



(a) Training and validation performance metrics per epoch (with C=1, γ=0.005).

(b) Training and validation performance metrics per epoch with (C=100, γ=0.001).

Figure 17: Comparison of training and validation performance metrics for the models

6 Discussion

As shown in the previous tables, we can conclude that:

- **ResNet + Softmax** model achieves the best results. The ResNet architecture, with its ability to handle deep networks effectively through residual connections, mitigates issues like vanishing gradients, thereby helping the model generalize better on unseen data.

- **ResNet + SVM** model, although not outperforming or being as competitive as the previous approach, still produces similar results. Moreover, it appears to be a better option because, with different values of the parameter C, the discrepancy between validation and test results is minimized, indicating a better generalization capability.

- Based on the results we obtained, it appears that our implementation does not outperform the previous ones in terms of accuracy. The results are closely comparable to those achieved with the CNN + standard SVM architecture.A further analysis is needed, particularly regarding the impact of the γ (σ^2) parameter in our implementation with **RFF kernel SVM**.

In an RBF kernel, γ controls how much individual data points influence the decision function:

- High γ (low σ^2): The model focuses more on individual points, leading to a highly flexible decision function but increasing the risk of overfitting, as it may capture noise. The RFF transformation reflects this by using random frequencies w with high variance (2γ), resulting in rapidly oscillating basis functions that capture fine details.

- Low γ (high σ^2): The model generalizes better by considering broader regions of the dataset, reducing overfitting but increasing the risk of underfitting. In RFF representations, this corresponds to random frequencies w with lower variance, leading to smoother basis functions.

We have also observed that increasing the number of features should, in theory, improve the kernel approximation. However, when keeping γ at the same level, the model performance deteriorates. This effect can be explained by the fact that, given a smaller number of features, γ should be more global rather than local. With fewer features, γ needs to capture interactions between all variables, while with a larger number of features, this can lead to overfitting, as demonstrated by our results. In other words, with a larger feature space, a fixed value of γ leads to a finer partitioning of the decision space, which may cause the model to overfit smaller regions rather than capturing broader patterns in the data. This suggests that when increasing the feature space, an appropriate tuning of γ is necessary to maintain the right balance between local flexibility and global generalization.

7 Conclusions and future works

The results we obtained by applying random Fourier features to the embedding feature vector did not meet our expectations, and the ResNet + Softmax architecture remains the top-performing model. However, the integration of an SVM classifier for classification showed promising results in terms of generalization. This improvement can likely be attributed to the SVM's focus on maximizing

the margin, which enhances generalization. Furthermore, running higher number of epochs should enhance the accuracy of the models as demonstrated in [5]. Beyond the results of this study, a deeper understanding of each *component* of the model provides a strong foundation for further research in this field. To improve performance and gain more insights, future work could explore several key directions, including:

- Enhancing hyperparameter tuning, potentially through Bayesian optimization or other automated techniques which could help mitigate the high computational cost while improving the model's ability to find optimal configurations without excessive manual intervention.
- Investigating alternative kernels, provided they are translation-invariant (e.g. Laplacian Kernel, Exponential Kernel...), to assess their impact on model performance. These could provide different trade-offs in terms of model flexibility and robustness, potentially improving results in certain tasks.
- Increasing the number of training epochs [5], which may allow the model to better capture complex patterns in the data. However, it is important to consider the computational cost associated with extended training and potential overfitting.
- Exploring other architectures, such as DenseNet, which may offer better balance between complexity and performance.

In conclusion, while the results from this study are not as promising as expected, there are still many avenues to explore, particularly in optimizing model efficiency. This could include better hyperparameter tuning, increasing the number of epochs, or experimenting with different translation-invariant kernels. Addressing these challenges could lead to more promising and improved results.

References

[1] Abien Fred M. Agarap. "An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification". In: *arXiv preprint arXiv:1712.03541* (2017). DOI: 10.48550/arXiv.1712.03541.

[2] Saturn Cloud. *How to Normalize Image Dataset Using PyTorch*. 2024. URL: https://saturncloud.io/blog/how-to-normalize-image-dataset-using-pytorch/.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015). DOI: 10.48550/arXiv.1512.03385.

[4] Shihao Jiang, Richard Hartley, and Basura Fernando. "Kernel Support Vector Machines and Convolutional Neural Networks". In: *Proceedings of the 2018 Digital Image Computing: Techniques and Applications (DICTA)* (2018). DOI: 10.1109/DICTA.2018.8615840.

[5] Kuang Liu. *PyTorch CIFAR models*. https://github.com/kuangliu/pytorch-cifar. Accessed: YYYY-MM-DD. 2017.

[6] Jacob Murel and Eda Kavlakoglu. *What is data augmentation?* 2024. URL: https://www.ibm.com/think/topics/data-augmentation.

[7] Xiao-Xiao Niu and Ching Y. Suen. "A novel hybrid CNN–SVM classifier for recognizing handwritten digits". In: *Pattern Recognition* 45.4 (2012), pp. 1318–1325. DOI: 10.1016/j.patcog.2011.09.021.

[8] Ali Rahimi and Ben Recht. "Random Features for Large-Scale Kernel Machines". In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 20. 2007, pp. 1177–1184. DOI: 10.5555/2981562.2981710.

[9] Farheen Ramzan et al. "A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages Using Resting-State fMRI and Residual Neural Networks". In: *Journal of Medical Systems* 44.2 (2019), pp. 1–12. DOI: 10.1007/s10916-019-1475-2.

[10] Yichuan Tang. "Deep Learning using Linear Support Vector Machines". In: *arXiv preprint arXiv:1306.0239* (2013). DOI: 10.48550/arXiv.1306.0239.