Neural Networks - Deep Learning Second Assignment

Dimitrios Karatis 10775, *Electrical and Computer Engineering, AUTH*

***Abstract*—This document provides an overview of the model training and evaluation process using the CIFAR-10 dataset. The focus is on leveraging Support Vector Machines (SVMs) to classify images into ten categories, such as airplanes, cars, and animals, making this a challenging yet foundational problem in computer vision.**

***Index Terms*—Support Vector Machine, deep learning, CIFAR-10, image classification, computer vision.**

I. Introduction

TO effectively train and evaluate the SVM models presented in this work, the **CIFAR-10 Dataset** was selected as the primary data source (available at: CIFAR-10 Dataset). CIFAR-10 is a popular image classification dataset containing 60,000 32x32 color images divided into ten classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. This dataset serves as a benchmark in computer vision and deep learning for evaluating classification model performance.

II. Dataset Description

The CIFAR-10 dataset consists of 50,000 training images and 10,000 test images, each labeled with one of the ten classes. Due to its diversity, it presents a complex classification problem and provides a comprehensive platform for testing and training CNN architectures in the context of image recognition.

A. Loading and Preparing the Dataset

To load the dataset, I used the ***cifar10.load_data()*** function from ***tensorflow.keras.datasets***. Additionally, a custom function, ***load_cifar***, uses this method to load the CIFAR-10 dataset while applying necessary preprocessing adjustments. The preprocessing steps, including normalization, rotation-based data augmentation, and PCA (Principal Component Analysis), are controlled by a matrix called ***preprocess***. Each element in this matrix determines whether a specific preprocessing step is applied (e.g., normalization is enabled if the first element is set to 1). Finally, the function outputs the preprocessed training, test, and validation datasets, as well as the number of PCA components needed to reach a certain variance.

The ***custom_rotation*** function augments an image dataset by adding randomly rotated versions of the images, enhancing variability to improve model generalization. Each flattened image (shape (3072,)) is reshaped into its original 3D form (32, 32, 3), rotated by a random angle between -20 and 20 degrees (excluding 0), and then flattened back. The function collects these rotated images and their corresponding labels, concatenates them with the original dataset, and returns the augmented dataset. This process increases the dataset size and helps the model handle rotational variations in real-world scenarios, reducing overfitting and improving robustness.

The ***filter_classes*** function is designed to refine a dataset by selecting a specified number of classes and their corresponding data. It allows the user to focus on a subset of the dataset, with the number of classes (num_classes) adjustable between 2 and 10. The function identifies the first num_classes classes in the dataset, filters the images and labels to include only the selected classes, and then remaps the labels to a sequential order (from 0 to num_classes - 1). This ensures the resulting dataset is compact and suitable for tasks requiring specific class subsets. This function was created because the waiting times for training and evaluating the SVM's in CIFAR-10 with all of the 10 classes were too long.

The ***apply_pca*** function reduces the dimensionality of the input data using Principal Component Analysis (PCA) while preserving a specified proportion of the total variance. It takes training and testing datasets along with a target variance ratio as inputs, fits the PCA model to the training data, and applies the resulting transformation to both datasets. The function returns the transformed datasets and the optimal number of principal components selected to achieve the desired variance retention. This approach helps simplify the data, improving computational efficiency and potentially enhancing model performance by focusing on the most relevant features. The code for all the functions used to load and preprocess the dataset can be found at the ***dataset_functions.py***. Also, in the ***paths_config.py*** file, the user can change the necessary paths.

```
# Function to load cifar-10 data
# preprocess is matrxix indicating:
# [using normalize, using roatations, using PCA]
def load_cifar(self, preprocess):
    n_components = 0

    # Load the CIFAR-10 dataset
    (x_train, y_train), (x_test, y_test) =
        cifar10.load_data()

    # Flatten the image data for processing
    # Each image is reshaped from (32, 32, 3) to
        a single vector
    x_train = x_train.reshape(x_train.shape[0],
        -1)
```

This document was prepared by Dimitrios Karatis as part of the first assignment for the Neural Networks course at Aristotle University of Thessaloniki.

```
x_test = x_test.reshape(x_test.shape[0], -1)

# Check if normalization is enabled
# If enabled, scale pixel values to the
    range [0, 1] using MinMaxScaler
if preprocess[0] == 1:
    scaler = MinMaxScaler()
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)

# Flatten label arrays to make them
    compatible with training functions
y_train = y_train.ravel()
y_test = y_test.ravel()

# Check if rotation augmentation is enabled
# If enabled, apply a custom rotation
    function to the training data
if preprocess[1] == 1:
    x_train, y_train = self.custom_rotation(
        x_train, y_train)

# Check if PCA is enabled
# If enabled, apply PCA to reduce
    dimensionality
if preprocess[2] != 0.0:
    x_train, x_test, n_components = self.
        apply_pca(x_train, x_test,
        preprocess[2])

# Return the preprocessed training and test
    datasets
return x_train, y_train, x_test, y_test ,
    n_components
```

B. Training and Evaluating Support Vector Machine (SVM)

To address the challenges posed by the high-dimensional and complex nature of the CIFAR-10 dataset, I implemented a Support Vector Machine (SVM) classifier. SVMs are particularly well-suited for handling high-dimensional data and aim to maximize the margin between classes by finding an optimal hyperplane. To enhance the performance and robustness of my SVM implementation, I conducted a grid search over hyperparameters such as the regularization parameter C and the kernel coefficient gamma. Additionally, I decided to use a combined scoring mechanism to determine the optimal C and gamma values. This combined score incorporates test accuracy, the F1 score, and a penalty for the percentage of support vectors to strike a balance between performance and model complexity while minimizing overfitting. The number of support vectors was also analyzed to further assess the potential for overfitting. The code for my SVM implementation can be found in the ***SVM.py*** file.

This code implements a Support Vector Machine (SVM) to classify the CIFAR-10 dataset using Linear, Polynomial, and Radial Basis Function (RBF) kernels. The dataset is first preprocessed, including normalization, rotation and optional dimensionality reduction via PCA. Due to extremely long training times i decided to make an SVM that classifies only 2 out of 10 classes of the CIFAR-10 dataset. Below are the results for different kernels and various values of C and gamma:

Detailed results, along with examples of both correct and incorrect classifications, can be found in the ***results_txt*** *folder.*

LINEAR KERNEL:

At first i used a preprocess matrix of [1, 0, 0.0] meaning only normalization will be used. Based on the results of the grid search for the linear SVM kernel with varying C values, we can observe a few key trends. As C increases, both the training accuracy and the number of support vectors tend to rise. For example, when C is 0.001, the accuracy is 82.38%, and the percentage of support vectors is 50.72%, while for C = 10, the training accuracy reaches 91.57%, and the percentage of support vectors decreases to 37.04%. However, there is a noticeable trade-off between training accuracy and test accuracy. Although increasing C improves training performance, it does not necessarily lead to better generalization on the test set. For instance, at C = 10, the test accuracy drops to 78.15%, lower than at smaller C values such as 0.001 (82.95%) and 0.01 (82.4%). This suggests that the model might be overfitting to the training data at higher values of C. The F1 score also follows a similar trend, with the highest value being 0.8295 at C = 0.001, and decreasing as C increases. The final best parameter selected is C = 10, indicating that despite the overfitting at this value, it yielded the highest training accuracy. This implies that while the model performs well on the training set, its ability to generalize to unseen data might be compromised at higher C values. Below we can see a diagram of how the C parameter affected the test accuracy of the model.

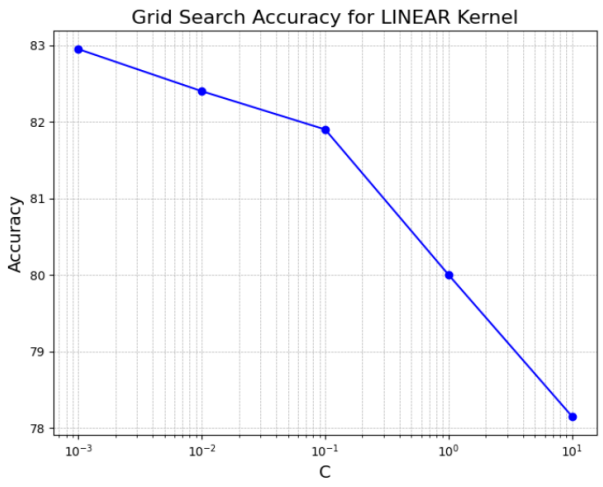

Fig. 1. Test accuracy vs C parameter (NO PCA and NO rotations)

Next, i tried a preprocess matrix of [1, 1, 0.95], meaning that now rotations and PCA are enabled.

The grid search results for the linear SVM kernel with added rotations and PCA show that the performance is relatively stable across the different values of C, with no dramatic improvements or declines. The accuracy on the training set

remains in the range of 81.1% to 82.4%, while the test set accuracy is also consistently high, ranging from 81.6% to 83.2%. Notably, adding PCA and rotations seems to slightly improve the model's ability to generalize, as the test accuracy is higher than it was without these transformations, where test accuracy was lower. The number of support vectors is also lower across all C values compared to the previous results, which suggests that the model is slightly less complex, potentially due to the dimensionality reduction with PCA. For instance, when C = 1, the accuracy is 82.395% on the training set and 82.55% on the test set, with a modest F1 score of 0.8255. The optimal choice of C appears to be 1, as it provides the best balance of accuracy and support vector usage. This suggests that the model is able to generalize well to unseen data while maintaining efficient use of support vectors. However, despite the improvements in generalization, the model still faces challenges at higher values of C, such as C = 10, where the training time significantly increases and the performance on the test set drops slightly. This reflects the trend where increasing C can lead to overfitting and slower convergence. Below we can see a diagram of how the C and gamma parameters affected the test accuracy of the model.

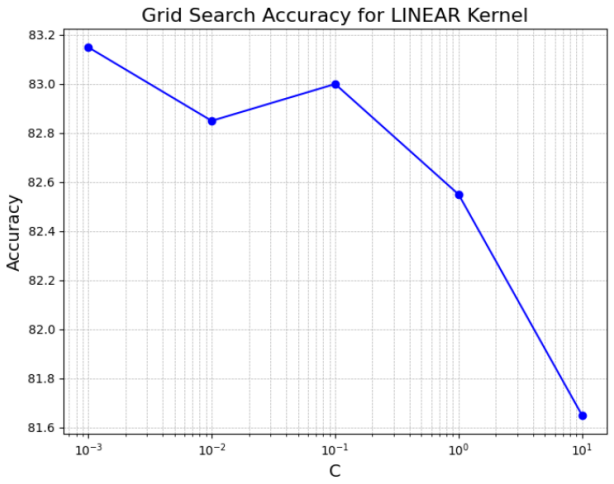

Fig. 2. Test accuracy vs C parameter (PCA and rotations)

POLYNOMIAL KERNEL (degree: 2):

The results from using a polynomial kernel (degree 2) without PCA or rotations show a noticeable variation in model performance as the hyperparameters (C and gamma) change. For smaller values of C and gamma (e.g., C=0.001, gamma=0.001), the model performs reasonably well, with accuracies around 64-66% on the test set. As the gamma value increases, the model starts to overfit, reaching 100% accuracy on the training set with a corresponding decrease in test set performance (around 89%). The number of support vectors decreases as C and gamma increase, reflecting the model's ability to generalize better with higher values of these hyperparameters. The best results occur with moderate values of C and gamma (e.g., C=0.01, gamma=0.1), where the accuracy stabilizes around 89% for both training and test sets, and the F1 scores are high (around 0.89). This suggests that there is a balance between model complexity and overfitting, with optimal generalization occurring at these moderate settings. The percentage of support vectors decreases with higher C values, showing the model's increasing confidence and the reduced number of vectors needed to define the decision boundary. Below we can see a diagram of how the C and gamma parameters affected the test accuracy of the model.

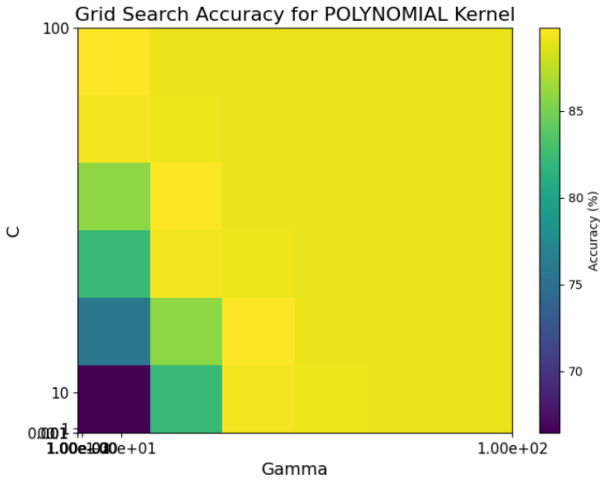

Fig. 3. C vs gamma vs test accuracy (NO PCA and NO rotations)

Next, i tried a preprocess matrix of [1, 1, 0.95], meaning that now rotations and PCA are enabled.

The results from the grid search and SVM classification with PCA and rotation applied show a broad range of performances depending on the hyperparameters chosen. The model with varying values of the regularization parameter C and kernel-specific parameter gamma showed differing accuracies, with the most notable improvement in accuracy observed as gamma increased. For example, when C=0.001 and gamma=0.1, the accuracy was 89.2% on the training set and 87.6% on the test set, yielding a high F1 score of 0.8760, indicating good classification performance. As the value of C and gamma increased, the model's performance on the test set remained

relatively stable, with an accuracy around 86.7%, despite the training accuracy reaching 100%. Notably, the percentage of support vectors varied significantly across the configurations, with values ranging from 25.9% to 100%, reflecting the complexity of the decision boundary. The results suggest that while higher values of C and gamma tend to overfit the training data, optimal configurations allow for balanced accuracy and F1 scores on unseen data. The inclusion of PCA and rotation likely contributed to improved model performance by providing more informative features and preventing overfitting, allowing the classifier to generalize better to the test set. Below we can see a diagram of how the C and gamma parameters affected the test accuracy of the model.

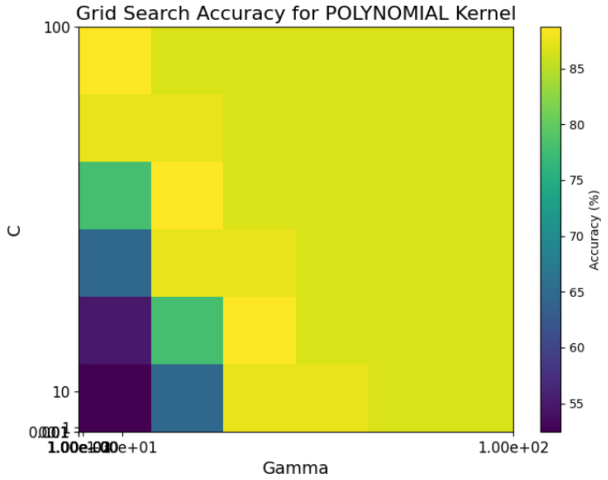

Fig. 4. C vs gamma vs test accuracy (PCA and rotations)

Since, the models with PCA and rotations seemed to perform better (in terms of generalizing at least) i decided to continue by testing different kernels with the augmented dataset.

POLYNOMIAL KERNEL (degree: 5):

Initially, for small values of C (e.g., C=0.001) and low gamma values, the model struggled to achieve meaningful accuracy, with training accuracy at 50.66% and test accuracy barely surpassing random guessing at 50.95%. As gamma increased to moderate values (e.g., gamma=0.01), the performance significantly improved, with test accuracy reaching 68.65% and a noticeable reduction in the percentage of support vectors, suggesting better margin optimization. For higher gamma values and fixed C, the training accuracy reached 100%, but this resulted in overfitting, as the test accuracy plateaued around 79.1%, with a slight increase in F1 scores (e.g., 0.7887 for gamma=1). Increasing C (e.g., C=0.1) further improved the test accuracy up to 84.3% for moderate gamma=0.01, showing the model's ability to balance complexity and generalization. Overall, the results highlight the sensitivity of SVM performance to hyperparameters, particularly C and gamma, and demonstrate the importance of selecting these values carefully to avoid overfitting and achieve optimal generalization.

When comparing the performance of the polynomial kernel with a degree of 2 to that of degree 5, it is evident that the 2nd-degree kernel produced better results, achieving a training accuracy of 94.81%, a test accuracy of 88.75%, and 36.34% support vectors, compared to 99.835%, 81%, and 39.29%, respectively, for the 5th-degree kernel. These findings indicate that the 5th-degree polynomial kernel is significantly more powerful but prone to overfitting, whereas the 2nd-degree kernel strikes a better balance between model complexity and generalization. In this case, training CIFAR-10 with only two classes, the additional power of the 5th-degree kernel is unnecessary and counterproductive. Below we can see a diagram of how the C and gamma parameters affected the test accuracy of the model.

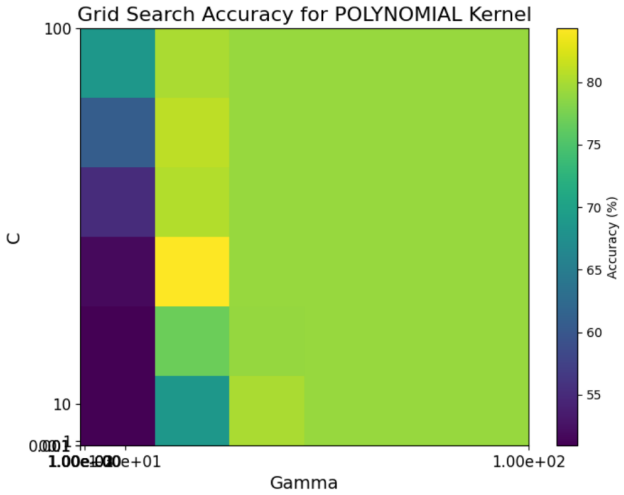

Fig. 5. C vs gamma vs test accuracy (PCA and rotations)

RADIAL BASIS FUNCTION (RBF) KERNEL:

With lower values of C (e.g., C=0.001) and smaller gamma (e.g., gamma=0.001), the model achieves moderate performance, with accuracies reaching around 70% and F1 scores of approximately 0.72. However, increasing gamma while keeping C low results in significant overfitting to the training data, as evidenced by perfect training accuracy but a sharp decline in test accuracy (approximately 50%), accompanied by poor F1 scores. This pattern indicates that the model struggles to generalize well under these conditions. As C increases to 0.01 and 0.1, and gamma is appropriately tuned (e.g., gamma=0.001 or gamma=0.01), there is a marked improvement in both test accuracy and F1 score, peaking at 87.1% test accuracy and 0.87 F1 score for C=0.1 and gamma=0.01. This configuration also significantly reduces the number of support vectors, implying a simpler and more efficient decision boundary. However, for larger gamma values (e.g., gamma=0.1), the model again tends to overfit, as shown by the high percentage of support vectors and degraded test performance.

For the results obtained with C values ranging from 1 to 100, the performance of the model exhibits a trend dependent on the value of gamma. When gamma is set to smaller values such as 0.001 or 0.01, the accuracy and F1 scores remain high, with the test accuracy exceeding 88% in some cases (e.g., C=10, gamma=0.01). As gamma increases to values such as 0.1, 1, or higher, the performance sharply declines, with test accuracy dropping to around 50%, irrespective of the C value.

Additionally, the number of support vectors is influenced by the C and gamma combination. For smaller gamma values, the number of support vectors decreases, indicating a better generalization and decision boundary. However, for larger gamma, the support vector count approaches the size of the training set, suggesting overfitting to the data. These results highlight that while higher C values can help capture complex patterns, they are only effective when gamma is appropriately chosen to balance model complexity and generalization.

The RBF and polynomial (degree 5) kernels differ significantly in their behavior and performance. The RBF kernel generally provides smoother and more flexible decision boundaries, which adapt well to non-linear data, especially when hyperparameters like gamma and C are well-tuned. In contrast, the polynomial kernel with degree 5 tends to create more complex, higher-order decision boundaries that may lead to overfitting, particularly in datasets with noise or limited training samples. The polynomial kernel can capture intricate patterns in data, but it often requires careful tuning to prevent over-complexity. Below we can see a diagram of how the C and gamma parameters affected the test accuracy of the model.

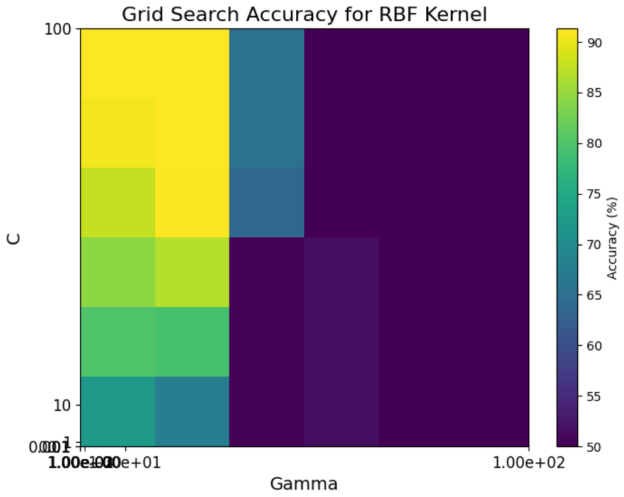

Fig. 6. C vs gamma vs test accuracy (PCA and rotations)

C. Training and Evaluating Classifiers

Now we need to compare the performance of the SVM to different classifiers. In order to do that, I fisrt initialized the classifiers, and then I trained every single one of them to a set of training images (only for the 2 out of 10 CIFAR-10 classes) with corresponding labels. After training, the classifiers are evaluated on a test set, where they predict labels for the test images and compute accuracy based on these predictions. The initialization, training and evaluation has been done by using the KNeighborsClassifier and NearestCentroid packages from sklearn.

First i only normalized the pixel values of the dataset and the results were:

```
k-NN (k=1) Accuracy Normalized: 71.50%
k-NN (k=3) Accuracy Normalized: 69.45%
Nearest Centroid Accuracy Normalized: 72.30%
(myenv2) [karatisd@cn93 nn_project_2]$
```

Fig. 7. Results for different classifiers (only normalized). Time it took: 18.814 seconds in Aristotelis Cluster

After that, i tried using PCA and applying rotations to the dataset. The results were:

```
Optimal number of components: 209 for 95.0% variance

k-NN (k=1) Accuracy Normalized (Augmented): 75.75%
k-NN (k=3) Accuracy Normalized (Augmented): 73.75%
Nearest Centroid Accuracy Normalized (Augmented): 72.30%
(myenv2) [karatisd@cn93 nn_project_2]$
```

Fig. 8. Results for different classifiers (using normalization, rotations and PCA). Time it took: 2 minutes in Aristotelis Cluster

When comparing the classifiers with and without PCA and rotations, it is evident that incorporating these techniques improved performance across all classifiers. For instance, the k-NN (k=1) classifier saw an increase in accuracy from 71.50% to 75.75%, and the k-NN (k=3) improved from 69.45% to 73.75%. ALthough, the nearest centroid classifier stayed the

same from 72.30% to 72.30%. These enhancements highlight the effectiveness of PCA in reducing dimensionality while retaining significant variance, coupled with rotation augmentation for robust feature extraction.

However, when comparing these classifiers with the SVM using the RBF kernel (with PCA and rotation), the SVM achieves significantly superior performance, indicating its stronger ability to model complex decision boundaries compared to simpler methods like k-NN and nearest centroid classifiers.

The differences in performance can be attributed to the nature of the algorithms. The SVM, with its ability to map data into higher-dimensional spaces using the RBF kernel, is more flexible and capable of capturing complex decision boundaries, leading to better generalization. In contrast, k-NN is a distance-based algorithm that struggles with high-dimensional data or small training sets, which can explain the not great results. The Nearest Centroid classifier, on the other hand, averages the data points of each class, resulting in a simpler decision boundary that works well for well-separated classes but struggles with more complex patterns. Thus, the SVM outperforms the k-NN and Nearest Centroid classifiers, demonstrating its effectiveness in handling complex, high-dimensional datasets like CIFAR-10.

D. Training and Evaluating Multilayer Perceptron (MLP)

To compare the performance of the SVM with a neural network, I implemented a Multi-Layer Perceptron (MLP) with a single hidden layer using hinge loss as the loss function. The MLP is designed to process the CIFAR-10 dataset (only the 2 out of 10 classes), which has been preprocessed to fit the input shape of the network. The hidden layer consists of 512 neurons with ReLU activation, followed by a dropout layer to prevent overfitting. The output layer uses a linear activation function to work with the hinge loss, enabling the model to perform multi-class classification in a One-vs-All format.

The model is trained using the Stochastic Gradient Descent (SGD) optimizer for improved convergence. An early stopping mechanism monitors the validation loss during training, halting the process if no improvement is observed after a set number of epochs. After training, the model evaluates its performance on a test set, and the results are visualized using plots. This implementation provides an effective baseline for comparing the MLP's performance to the SVM on the same dataset. The code for the MLP can be found in the ***MLP.py*** file and the corresponding main function, in the ***main_MLP.py***. Below are the results when training the MLP for only 2 out of 10 CIFAR-10 class, with a learning rate of 0.01, a dropout rate of 0.1, a single hidden layer consisting of 512 neurons with relu ativation function trained for 50 epochs with a batch size of 32. Also, i used a preprocess matrix of [1, 0, 0, 0] meaning only normalization will be used.

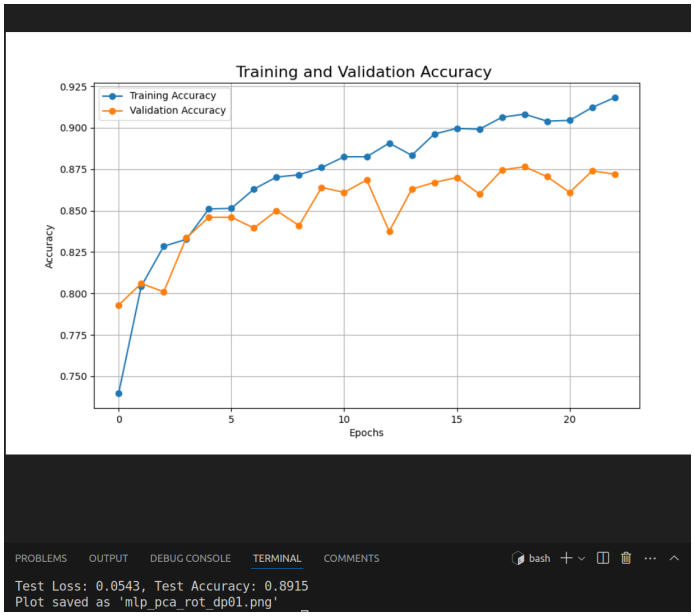

Fig. 9. Training vs Validation accuracy of MLP (baseline). Time it took: 1 minute in Aristotelis Cluster

Next, in order to better the results, i tried using PCA and rotations to the dataset, while also changing the dropout rate to 0.4. The results were:

Fig. 10. Training vs Validation accuracy of MLP (with PCA and rotations). Time it took: 3 minutes in Aristotelis Cluster

The results of the MLP with a single layer of 512 neurons, hinge loss, PCA for dimensionality reduction, and data augmentation via rotations achieved a test accuracy of 91.75%, with a test loss of 0.0422. In comparison, the SVM with the RBF kernel achieved a slightly lower accuracy of 91.35% (for C=100 and gamma=0.001) under optimized hyperparameters. This indicates that the MLP outperformed the RBF kernel in this case. The RBF kernel relies on a fixed similarity measure

and requires careful hyperparameter tuning to capture non-linear patterns effectively. In contrast, the MLP benefits from greater flexibility and the ability to adapt its features during training, likely enabling it to generalize better to the test set. However, the higher computational cost and complexity of training an MLP compared to the SVM should be noted.

In clonclusion, for the task of classifying 2 out of 10 CIFAR-10 classes, the MLP with a single hidden layer and hinge loss, proves to be a more suitable choice in terms of simplicity, as the additional complexity and computational power of an SVM tend to result in overfitting.