NEURAL NETWORKS - DEEP LEARNING SECOND ASSIGNMENT

Dimitrios Diakoloukas 10642, Electrical and Computer Engineering, *AUTH, Student*

***Abstract*—This study presents the implementation of multiple Support Vector Machines (SVMs) for multi-class image classification on the CIFAR-10 dataset. The CIFAR-10 dataset, using 60,000 images across 10 categories, was preprocessed using Principal Component Analysis (PCA) to reduce dimensionality while preserving 90% of the data's variability. The SVM models were then trained and evaluated, often employing cross-validation to ensure robust performance. To assess the model's efficacy, classification metrics such as accuracy, confusion matrices, and precision-recall curves were analyzed. Additionally, comparisons were made with other classifiers, including *k*-Nearest Neighbors (*k*-NN) and a Multi-Layer Perceptron (MLP). This study highlights the effectiveness of SVMs in image classification tasks and explores the influence of hyperparameter tuning and loss functions on model performance.**

I. Introduction

IMAGE classification is an essential task in computer vision, used in areas like autonomous driving and facial recognition. The CIFAR-10 dataset, a widely recognized dataset, presents us with the problem of classifying images into 10 distinct categories, each of which has natural objects such as animals and vehicles. Support Vector Machines (SVMs), known for their strong theoretical foundations and versatility, are commonly applied to such tasks because of their ability to handle high-dimensional data effectively. The main point of this assignment is to further experiment with SVMs.

II. Data Loading and Preprocessing

This section explains how I loaded the CIFAR-10 dataset and prepared it so that it could be used with the SVM implementations but also (*k*-NN) algorithm and MLP implementation for testing purposes.

A. Loading CIFAR-10 Data

The CIFAR-10 dataset contains 60,000 small color images (32x32 pixels) across 10 different classes. There are 50,000 images for training and 10,000 images for testing. Each image is stored as a set of pixel values and the data is divided into multiple files.

I used the following steps to load the data:

- **Unpacking Data Files:** The training data is stored in five files, each containing 10,000 images. I created a helper function called unpickle to open each file and load the data inside.
- **Combining Training Files:** I combined the data from all five training files into one large set. This gave us a total of 50,000 images for training. I did the same for the labels, so each image has a label showing which class it belongs to.
- **Loading Test Data:** The test data is stored in a separate file. I loaded this file in the same way to get 10,000 test images and their labels.

B. Data Preprocessing

After loading the data, I prepared it for use with machine learning by following these steps:

- **Reshaping Images:** Each image was originally stored as a long row of numbers (a flat array). I reshaped these numbers back into the original 32x32 size with 3 color channels (Red, Green, and Blue).
- **Normalizing Pixel Values:** The pixel values in each image were originally between 0 and 255. To make the data easier to work with, I scaled these values to be between 0 and 1 by dividing each value by 255. This helps the model perform better and faster.

The final output of our data loading and preprocessing is:

- `x_train`: A set of 50,000 training images, each with shape (32, 32, 3) and pixel values between 0 and 1.
- `y_train`: A set of 50,000 labels for the training images.
- `x_test`: A set of 10,000 test images, also with shape (32, 32, 3) and pixel values between 0 and 1.
- `y_test`: A set of 10,000 labels for the test images.

This data is now ready to be used in our CNN and MLP algorithms as well as the k-NN algorithm, making it easy to classify each image based on its pixel values.

III. Differences Between Linear, Sigmoid, RBF, and Polynomial Kernels

Kernels are functions used in machine learning, especially in Support Vector Machines (SVMs), to transform data into a higher-dimensional space where it may become linearly separable. Below, I describe the differences between the linear, sigmoid, radial basis function (RBF), and polynomial kernels:

A. Linear Kernel

The linear kernel is the simplest kernel function and is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

- **Usage:** It is used when the data is linearly separable.

- **Advantages:** Computationally efficient and interpretable.

- **Disadvantages:** Limited to linearly separable problems.

B. Sigmoid Kernel

The sigmoid kernel is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)$$

Where:

- γ is a scaling parameter.
- r is a shift parameter.

- **Usage:** It is inspired by neural networks and mimics their activation function.

- **Advantages:** Suitable for specific tasks involving non-linear data.

- **Disadvantages:** Its performance can be sensitive to parameter choices, and it may not be positive semi-definite for all values of γ and r.

C. Radial Basis Function (RBF) Kernel

The RBF kernel, also known as the Gaussian kernel, is given by:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

Where $\gamma > 0$ controls the spread of the kernel. - **Usage:** It is highly effective in cases of non-linearly separable data.

- **Advantages:** It can map input data into an infinite-dimensional space and is widely used for non-linear problems.

- **Disadvantages:** Requires careful tuning of γ to prevent overfitting or underfitting.

D. Polynomial Kernel

The polynomial kernel is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)^d$$

Where:

- γ is a scaling parameter.
- r is a shift parameter.
- d is the degree of the polynomial.

- **Usage:** Suitable for capturing complex non-linear relationships.

- **Advantages:** Capable of representing intricate structures in data.

- **Disadvantages:** High degree polynomials can lead to overfitting, and computational cost increases with d.

E. Summary Table

Kernel	Expression	Key Features
Linear	$\mathbf{x}_i^\top \mathbf{x}_j$	For linearly separable data.
Sigmoid	$\tanh(\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)$	Mimics neural network activation functions.
RBF	$\exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$	Excellent for non-linear problems.
Polynomial	$(\gamma \mathbf{x}_i^\top \mathbf{x}_j + r)^d$	Captures complex non-linear structures.

IV. SVM USING LIBSVM: ANALYSIS WITHOUT AND WITH PCA

In this section, we applied Support Vector Machines (SVM) using the LIBSVM library to classify images from the CIFAR-10 dataset, but did not apply grid search techniques for hyperparameter tuning.

A. No PCA Implementation Explanation

In the initial implementation without PCA to simplify the task, we selected two classes (label 0 and label 1) and performed binary classification. The key steps involved were:

- **Data Preparation**: Images from the two selected classes were reshaped and filtered. The dataset was split into 60% for training and 40% for testing. We also used 10000 samples instead of 60000 that the dataset has originally since without PCA the implementation took a long long time to finish.
- **Training SVM Models**: Using LIBSVM, we trained SVM models with four different kernel types: Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid. Each kernel was tested to observe its performance.
- **Performance Evaluation**: The models were evaluated using metrics such as accuracy, confusion matrices, and classification reports. Additionally, the training and evaluation times were recorded.

*1) **Results and Observations**:* The SVM models were evaluated using four different kernels, and their performance is summarized below:

1. **Linear Kernel:** The linear kernel had the longest training time (50.81 seconds) and achieved an accuracy of 77.10%. Precision and recall were balanced for both classes, but it struggled to capture non-linear relationships in the data.

2. **Polynomial Kernel:** The polynomial kernel trained faster (40.07 seconds) but had a slightly lower accuracy of 75.90%. While it showed high precision for Class 0, its recall was low, indicating difficulty in correctly identifying all samples from Class 0. The overall performance was inconsistent.

3. **Radial Basis Function (RBF) Kernel:** The RBF kernel had the shortest training time (31.30 seconds) and achieved

the best accuracy of 81.37%. Precision and recall were well-balanced, making it the most effective kernel for this task.

4. **Sigmoid Kernel:** The sigmoid kernel's training and evaluation times were slightly longer than those of the RBF kernel, and its accuracy was 78.60%. While its performance was better than the polynomial kernel, it was slightly weaker than the RBF kernel.

The implementation in Python for this is provided in the `svm_libsvm_no_pca.py` file, and the results are stored in the `svm_results_2_classes.txt` file.

Below are some confusion matrices showing the performance of the SVM model for different kernels. Each matrix represents the distribution of true labels versus predicted labels, highlighting the accuracy for each kernel:

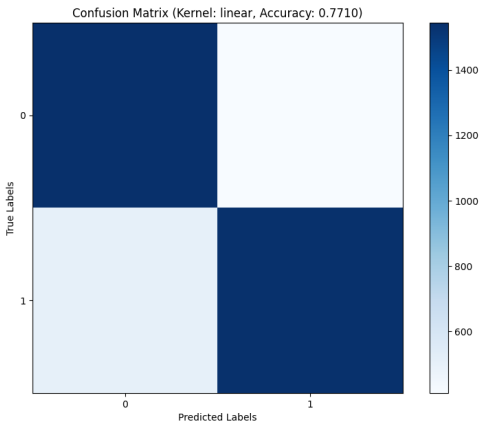

Fig. 1. Confusion Matrix for Linear Kernel.

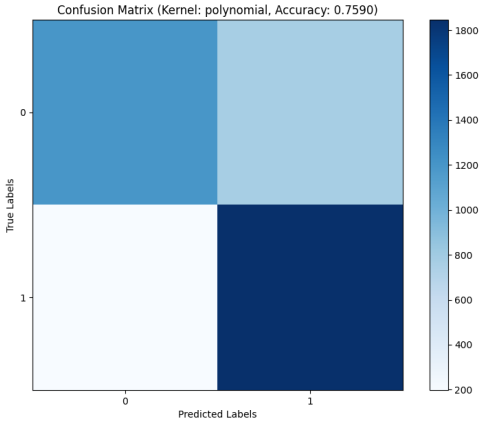

Fig. 2. Confusion Matrix for Polynomial Kernel.

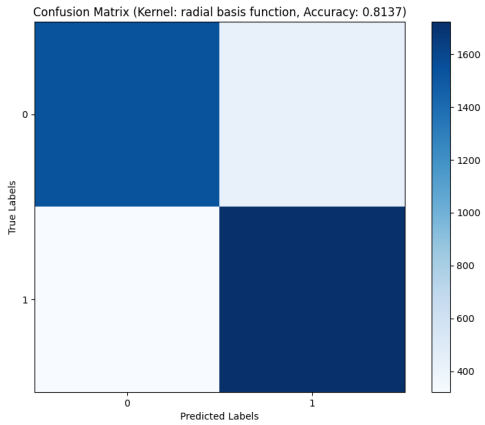

Fig. 3. Confusion Matrix for RBF Kernel.

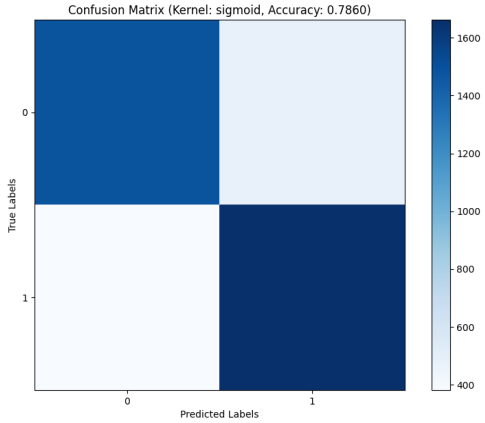

Fig. 4. Confusion Matrix for Sigmoid Kernel.

Conclusion: The RBF kernel performed the best in terms of accuracy and balanced predictions. It is the most suitable kernel for this binary classification task, while the linear and polynomial kernels showed limitations in handling non-linear data.

B. PCA Implementation Explanation

To address the high dimensionality of the CIFAR-10 dataset and reduce computational complexity and be capable to use the entire 10 classes and 60000 sample dataset and finish training in a reasonable time, we implemented Principal Component Analysis (PCA). The key steps involved were:

- **Dimensionality Reduction:** PCA was used to reduce the dataset's dimensionality while retaining 90% of the original variance. This ensured that most of the critical information from the images was preserved.
- **Data Preparation:** After applying PCA, the dataset was split into 60% for training and 40% for testing. The reduced dataset allowed faster training and evaluation while still enabling meaningful performance comparisons.
- **Training SVM Models:** Similar to the no-PCA implementation, SVM models were trained using four kernel types: Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid.
- **Performance Evaluation:** The models were evaluated using accuracy, confusion matrices, classification reports, and computation times.

*1) **Results and Observations**:* The results for the SVM models with PCA applied are summarized below:

1. **Linear Kernel:** Training took 869.94 seconds, and the accuracy was 40.51%. Precision and recall were low across all classes, indicating that the linear kernel struggled even with dimensionality reduction.

2. **Polynomial Kernel:** Training time reduced significantly to 196.78 seconds, and accuracy improved to 46.25%. Class-wise performance showed variability, with some classes achieving better precision and recall, but overall results were inconsistent.

3. **Radial Basis Function (RBF) Kernel:** This kernel achieved the best accuracy of 53.29% with a training time of 120.26 seconds. Precision and recall were more balanced across classes compared to other kernels, making it the most effective kernel even after PCA.

4. **Sigmoid Kernel:** Despite the reduced training time of 70.17 seconds, accuracy dropped to 20.07%. The sigmoid kernel performed poorly, with low precision and recall across all classes, suggesting that it was not suitable for this task.

The implementation in Python for this is provided in the `svm_libsvm_all.py` file, and the results are stored in the `svm_results.txt` file.

Below are some confusion matrices showing the performance of the SVM model for different kernels after PCA. Each matrix highlights the predicted versus true labels and the corresponding accuracy.

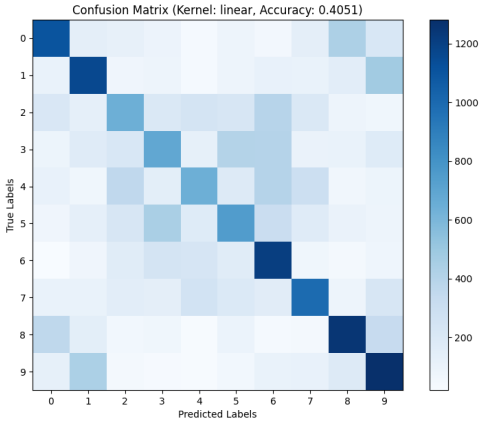

Fig. 5. Confusion Matrix for Linear Kernel.

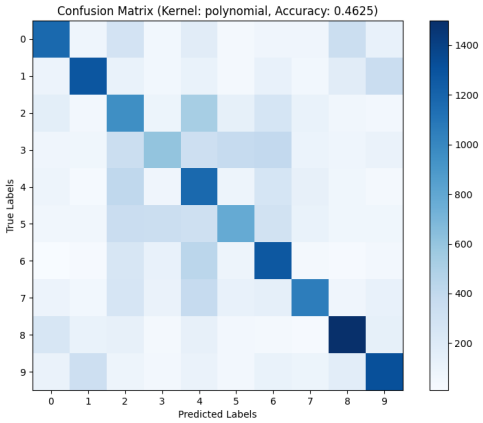

Fig. 6. Confusion Matrix for Polynomial Kernel.

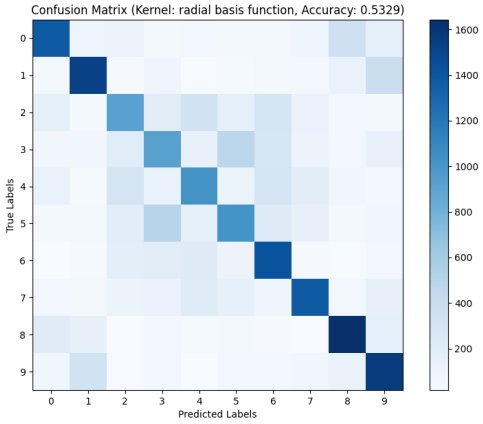

Fig. 7. Confusion Matrix for RBF Kernel.

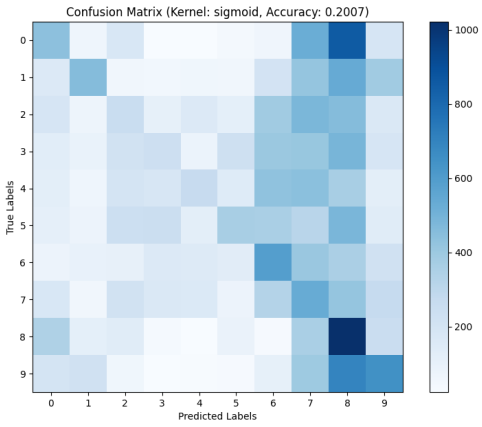

Fig. 8. Confusion Matrix for Sigmoid Kernel.

Conclusion: Applying PCA reduced the training and evaluation times significantly. However, while the RBF kernel maintained its superior performance, other kernels showed mixed results. The linear and sigmoid kernels performed poorly, while the polynomial kernel showed moderate improvement. Overall, PCA allowed faster training with some trade-offs in accuracy, particularly for simpler kernels.

This analysis allowed us to compare the performance of SVM models with and without PCA and across different kernels. The results, including accuracy and confusion matrices, clearly show the trade-offs of dimensionality reduction and kernel selection in image classification.

V. Grid Search Implementation Analysis

A. *Grid Search Implementation*

To find the best settings for SVM models, we used a method called grid search. This tested different kernel types (Linear, Polynomial, RBF, and Sigmoid) with combinations of regularization (C) and kernel (γ) parameters. Here's how it was done:

- **Parameter Testing:** We tested $C = \{0.1, 1, 10\}$ and $\gamma = \{0.001, 0.01, 0.1, 1\}$ for all kernels.
- **Kernel Types:** Different kernels were tried to check how well they could handle the data. These were Linear, Polynomial, RBF, and Sigmoid kernels.
- **Measuring Performance:** We checked how accurate the models were and used scores like precision, recall, and F1 to see how well they worked for different classes.
- **Training Time:** We recorded how long it took to train each model to compare their efficiency which will be demonstrated below. Although it is important to note that the training time for the grid search operation to select the best parameters took way to long specifically 16 hours and 21 minutes on my personal CPU.

*1) **Results and Observations**:* Here are the results for each kernel:

- **Linear Kernel:**
  - Training Time: 176.15 seconds
  - Testing Accuracy: 40.54%
  - Observations: This kernel was simple and fast but couldn't handle the complexity of the data well.
- **Polynomial Kernel:**
  - Training Time: 258.34 seconds
  - Testing Accuracy: 42.78%
  - Observations: The polynomial kernel did better than the linear one but took more time to train.
- **RBF Kernel:**
  - Training Time: 256.84 seconds
  - Testing Accuracy: 9.59%
  - Observations: The RBF kernel performed very poorly, likely because its settings were not suitable for the dataset. It seems like it has so low accuracy which is almost random guessing.
- **Sigmoid Kernel:**
  - Training Time: 101.78 seconds
  - Testing Accuracy: 38.16%
  - Observations: The sigmoid kernel was quick to train but didn't give good results overall.
- **k-Nearest Neighbors (k-NN):**
  - Testing Accuracy: 35.80%
  - Observations: k-NN didn't work as well as the SVM models. Its accuracy varied a lot for different classes, and it struggled with complex patterns in the data.

*2) **Conclusion**:* From the grid search, we learned:

- The **Polynomial kernel** had the best accuracy (42.78%), although it took longer to train.
- The **Linear kernel** was faster to train but not as accurate for this dataset.
- The **RBF kernel** did not work well because the settings needed to be adjusted better. That is why I implemented another approach with grid search for hyperparameter tuning for RBF kernel later on.
- The **Sigmoid kernel** was the worst performer with low accuracy.
- **k-NN** had lower accuracy than SVM models and was inconsistent across different classes.

The final implementation for this can be found in the `svm_libsvm_all_grid.py` file, and the results are saved in `grid_svm_knn_results.txt`.

*3) **Confusion Matrices**:* The following confusion matrices show how well the models predicted the correct labels for the dataset:

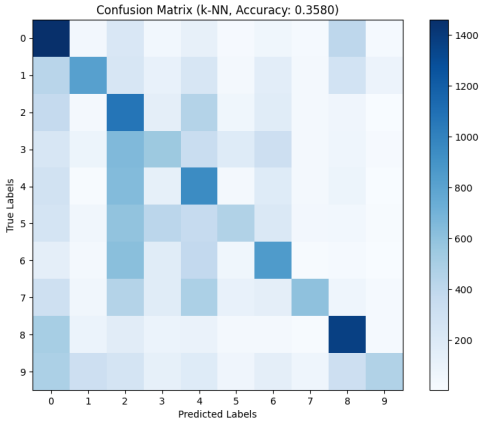

Fig. 9. Confusion Matrix for k-NN.

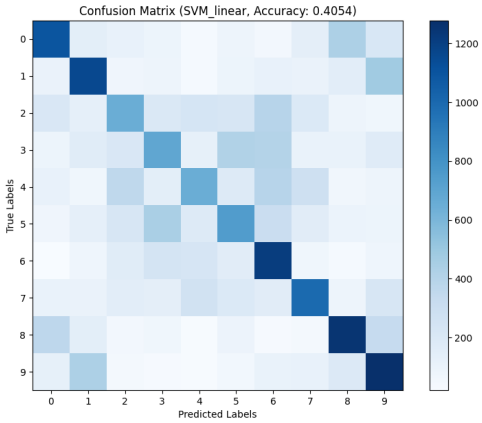

Fig. 10. Confusion Matrix for Linear Kernel.

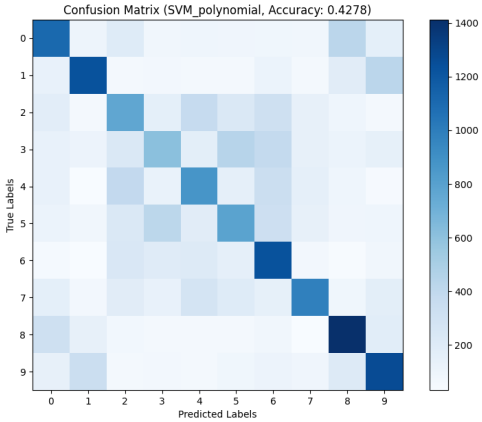

Fig. 11. Confusion Matrix for Polynomial Kernel.

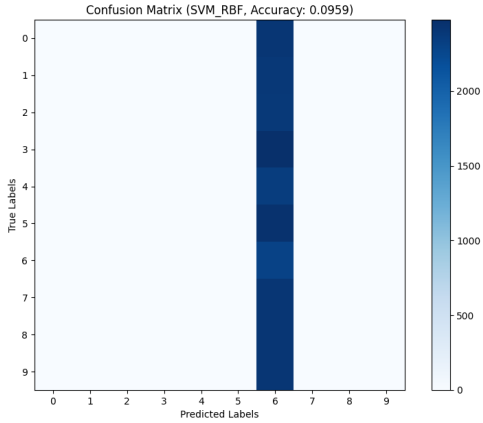

Fig. 12. Confusion Matrix for RBF Kernel.

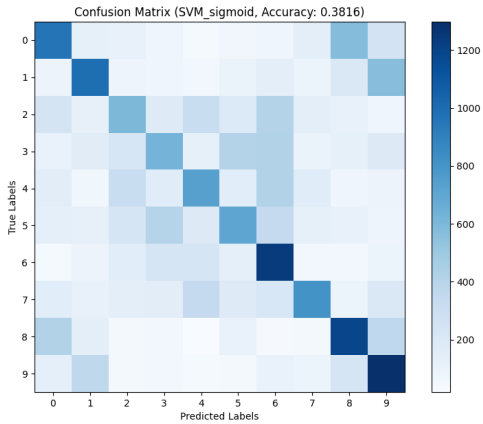

Fig. 13. Confusion Matrix for Sigmoid Kernel.

Final Thoughts: The polynomial kernel worked best in terms of accuracy and training time. The linear kernel is a good choice if speed is more important. The RBF kernel might improve with better settings. That is why in section VI I demonstrated a more specific grid search with other configurations for the RBF kernel. The sigmoid kernel and k-NN were less effective, showing they are not ideal for this dataset.

VI. Grid Search and RBF Kernel Analysis

A. Grid Search Implementation for RBF Kernel

To optimize the performance of the SVM with an RBF (Radial Basis Function) kernel, a grid search was performed. This method tested combinations of C and γ values to identify the best settings. Because the previous time the grid search process took more than 16 hours to execute I now decided to go back to the 2 class implementation but using the entire dataset for them. The key steps were:

- **Dataset Preparation:** The CIFAR-10 dataset was filtered to include only two classes (class 0 and class 1). The maximum of samples was used for the 2 classes basicaly 12000 samples. PCA reduced the feature dimensions to 100 components to ensure faster training and testing.
- **Parameter Grid:** The grid search tested combinations of regularization parameters $C = \{0.001, 0.01, 0.1, 1, 10, 100\}$ and kernel parameters $\gamma = \{0.001, 0.01, 0.1, 1, 10, 100\}$. Thus, I used an increased amount of parameters for more detailed testing.
- **Performance Metrics:** Accuracy and F1 score were computed for each parameter combination. Additionally, the percentage of support vectors used in the training set was monitored to evaluate model efficiency.
- **Computational Efficiency:** The time taken to train each model was recorded to compare computational costs for different parameter settings.

*1) **Results and Observations**:* The grid search results are summarized in the `rbf.txt` file, and the best parameters identified are:

- **Best Parameters:** $C = 100$, $\gamma = 0.001$
- **Best Accuracy:** 91.60%
- **Number of Support Vectors:** 3062
- **Total Training Time:** 362 seconds

A detailed breakdown of accuracy for all parameter combinations is shown below but the full breakdown is in `rbf.txt` file:

- For $C = 0.001$:
  - $\gamma = 0.001$: 72.75%
  - $\gamma = 0.01$: 70.00%
  - Higher γ: Accuracy dropped to around 50%.
- For $C = 100$:
  - $\gamma = 0.001$: 91.60%
  - $\gamma = 0.01$: 91.25%
  - Higher γ: Accuracy dropped significantly.

*2) **Final Model Evaluation**:* The best model was retrained using the optimal parameters ($C = 100$, $\gamma = 0.001$). The evaluation results were as follows:

- **Accuracy:** 91.60%
- **Number of Support Vectors:** 3062
- **Total Training Time:** 362 seconds

The results confirm that the selected parameters allow the RBF kernel to achieve high accuracy while maintaining a manageable number of support vectors, which keeps the model efficient.

*3) **Visualizing the Results**:* The accuracy for each combination of C and γ is visualized as a heatmap:

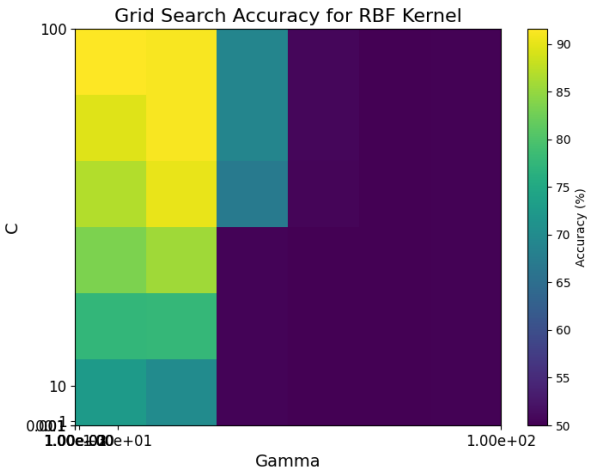

Fig. 14. Grid Search Accuracy for RBF Kernel.

Code and Results: The code for this implementation is available in `RBF_SVM.py`, and detailed results are logged in the `rbf.txt` file.

VII. Implementation Using Sklearn SVM

A. Motivation for Sklearn SVM

After experimenting with the libsvm library, I decided to try Scikit-learn's SVM implementation for the following reasons:

- Built-in functions, such as 'RandomizedSearchCV', make it easier to perform hyperparameter optimization efficiently.
- The ability to integrate with other Scikit-learn tools simplifies preprocessing, cross-validation, and evaluation.

B. Implementation Steps

The process consisted of the following key steps:

- **Data Preparation:**
  - The CIFAR-10 dataset was loaded, containing 50,000 training samples and 10,000 test samples, each with shape $(32 \times 32 \times 3)$.
  - Images were flattened into vectors of size 3,072 (32 x 32 x 3) for compatibility with the SVM model.
  - Labels corresponding to the 10 classes were extracted.
- **Dimensionality Reduction with PCA:**
  - PCA (Principal Component Analysis) was applied to the training and test sets, retaining 90% of the variance in the data.
  - This reduced the number of features from 3,072 to 99 components, which:
    - Improved computational efficiency.

  - Preserved the majority of critical information from the dataset.
- **Hyperparameter Search:**
  - A search for optimal hyperparameters was conducted using ‘RandomizedSearchCV‘ with 2-fold cross-validation.
  - The following hyperparameter ranges were tested:
    - C **(Regularization Parameter):** {0.1, 1, 10, 100}.
    - γ **(Kernel Coefficient):** {’scale’, ’auto’, 0.001, 0.01, 0.1, 1}.
    - **Kernel Type:** {’linear’, ’rbf’, ’poly’, ’sigmoid’}.
  - A total of 5 random combinations of these parameters were tested, and the best combination was selected based on the cross-validated accuracy.
- **Training the Final Model:**
  - Using the best hyperparameters found, a final SVM model was trained on the full PCA-transformed training set.
  - The ‘OneVsOneClassifier‘ strategy was applied to handle the multi-class classification task.
- **Evaluation:**
  - The final model was evaluated on the test set. Key metrics, such as accuracy and prediction times, were recorded.
  - The first 10 predictions from the test set were saved for a qualitative review of model performance.

C. What Was Tested and Observed

During the implementation, the following were tested:

- **Effect of PCA:**
  - Dimensionality reduction was tested to observe its impact on computational cost and model performance. Retaining 90% variance with PCA reduced the feature space to 99 dimensions.
- **Hyperparameter Combinations:**
  - Various values for C, γ, and kernel type were tested to find the optimal configuration.
  - The search aimed to balance regularization (C) and the sensitivity of the kernel function (γ).
- **Kernel Comparison:**
  - Four kernel types were tested: Linear, RBF, Polynomial, and Sigmoid. The RBF kernel was found to outperform others significantly.

D. Results and Observations

- **PCA Results:**
  - PCA reduced the feature space from 3,072 to 99 dimensions, retaining 90% of the variance. This step reduced computational complexity without significantly impacting performance.
- **Hyperparameter Search Results:**
  - The best parameters after the search were:

    ’kernel’: ’rbf’, ’gamma’: ’auto’, ’C’: 100

  - The search process took 10,122.85 seconds (approximately 2.8 hours) and tested 5 random combinations of parameters.
- **Test Accuracy:**
  - Using the best parameters, the model achieved a test accuracy of 55.61%, demonstrating moderate classification performance for the CIFAR-10 dataset.
- **Evaluation Time:**
  - Evaluation on the test set took 206.37 seconds.
- **Example Predictions:**
  - First 10 predictions on the test set were: [3, 8, 8, 0, 4, 6, 3, 6, 2, 1].
  - These predictions indicated the model’s ability to classify various CIFAR-10 categories, though not perfectly.

E. Visualization of Hyperparameter Search

The classification accuracy for each hyperparameter combination tested during the ‘RandomizedSearchCV‘ process is visualized below:

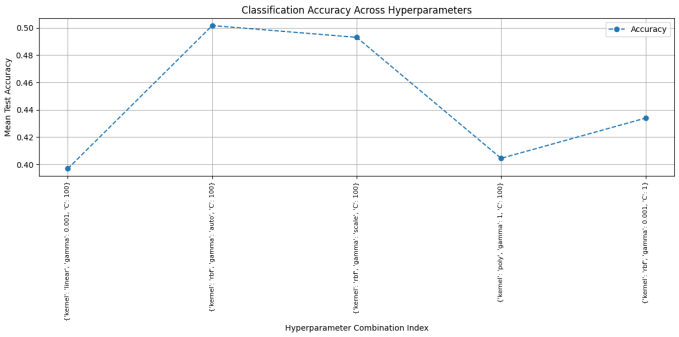

Fig. 15. Classification Accuracy Across Hyperparameters

F. Conclusion

The implementation using Scikit-learn SVM provided a streamlined approach to training and tuning SVM models. The results showed:

- **Effectiveness of PCA:** Dimensionality reduction significantly lowered computational requirements while preserving essential data features.
- **Optimal Parameters:** The RBF kernel with ‘gamma=’auto’‘ and ‘C=100‘ achieved the best balance of accuracy and generalization.
- **Limitations:** Although the test accuracy of 55.61% is moderate, further improvements could be achieved by exploring additional hyperparameters, preprocessing methods, or using a more advanced model.

The implementation code is available in the `SVC_based.py` script, and results are logged in the `svc_results_with_pca.txt` file.

VIII. Comparisons to MLP Implementation

After comparing the train and test accuracy of the SVM models across all kernels and the k-NN algorithm, I later trained and evaluated a Multilayer Perceptron (MLP) model. Below is a detailed description of the MLP architecture, training process, and performance comparison.

A. *Model Architecture*

The MLP consists of several dense (fully connected) layers, designed to efficiently handle the CIFAR-10 classification task:

- **Flatten Layer:** Converts 2D images (32x32x3) into 1D vectors for input into the dense layers.
- **Three Dense Layers:**
  - Layer 1: 512 units with ReLU activation.
  - Layer 2: 256 units with ReLU activation.
  - Layer 3: 128 units with ReLU activation.

  Each dense layer is followed by:
  - **Batch Normalization:** Stabilizes learning and accelerates convergence.
  - **Dropout:** Regularization to prevent overfitting (Dropout rates: 40% for first two layers, 30% for the third layer).
- **Output Layer:** 10 units with softmax activation, producing probabilities for the 10 CIFAR-10 classes.

The MLP as we demonstrated in the previous assignment can learn complex patterns in the CIFAR-10 dataset.

B. *Compilation and Training*

- **Optimizer:** Adam optimizer with a learning rate of 0.001, which adapts the learning rate during training for efficient convergence.
- **Loss Function:** Categorical Cross-Entropy, suitable for multi-class classification tasks.
- **Evaluation Metric:** Accuracy was used to monitor both training and validation performance.
- **Training Configuration:** The model was trained for 30 epochs with a batch size of 60, balancing computational efficiency and learning stability.

C. *Results and Observations*

- **Train and Test Accuracy:**
  - The MLP achieved a test accuracy of 44.63%, outperforming the k-NN models and some of the SVM models like the linear and sigmoid from time to time. Although the RBF kernel SVM seems to be a lot better in accuracy since it managed a 55.61% accuracy in the Cifar-10 dataset.
  - Training was way faster than grid-search-based SVMs but required more computational resources due to the large number of parameters.
- **Visualization:**
  - Accuracy and loss curves for training and validation were plotted and saved as `training_history_MLP_BEST.png`.
  - Correct and incorrect predictions from the test set were visualized, highlighting areas where the model performed well and struggled.
  - Sample predictions were saved in `sample_predictions_MLP_BEST.png`.

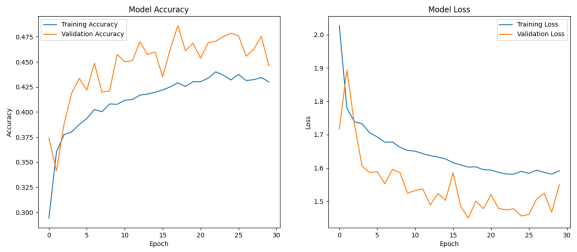

Fig. 16. MLP training history

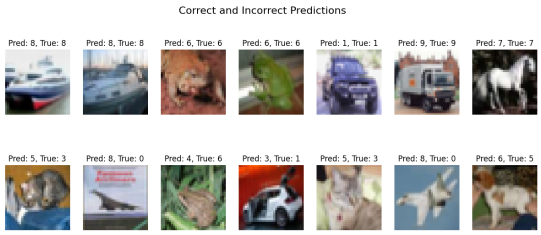

Fig. 17. True and false predictions from MLP

D. *Comparative Insights*

- **MLP vs. SVM:**
  - While SVM models required extensive hyperparameter tuning (via grid search or randomized search) to achieve reasonable accuracy, the MLP naturally handled the non-linear patterns in the data with minimal tuning. Although the Radial Basis Function Kernel SVM seems to have higher accuracy and precision than my MLP.
- **MLP vs. k-NN:**
  - The MLP outperformed k-NN by a significant margin, demonstrating its ability to learn complex hierarchical representations, unlike k-NN, which relies on simple distance metrics.
- **Regularization Benefits:**
  - The use of dropout and batch normalization in the MLP contributed to its improved generalization and faster convergence compared to SVMs.

IX. References Section

References

[1] https://www.ibm.com/topics/knn
[2] https://www.csie.ntu.edu.tw/ cjlin/libsvm/
[3] https://keras.io/api/layers
[4] https://scikit-learn.org/1.5/modules/svm.html
[5] https://www.cs.toronto.edu/ kriz/cifar.html