CNN Image Classification model using PyTorch trained with CIFAR-10 Dataset

Charlie Shin
University of Alabama Huntsville
Huntsville, USA
as1152@uah.edu

***Abstract*—This study attempts to replicate the convolutional neural network (CNN) described in "Classification of Image with Convolutional Neural Network and TensorFlow on CIFAR-10 Dataset" by Gunjan Sharma and Vijay K. Jadon. Due to the absence of parameters such as batch size and learning rate in the original paper, variations were used to observe effects on model performance and accuracy. The CNN was implemented in PyTorch and trained using Google Colab, achieving a maximum test accuracy of 70.89%, while the reference paper had a test accuracy of 98.93%. Experiments showed that changes in batch size and learning rate impacted training dynamics and final accuracy. Additionally, the CNN outperformed a support vector machine (SVM) with PCA, which had a 51.27% accuracy. These findings show the challenges of reproducibility in deep learning research and the sensitivity of CNN performance to hyperparameter variability.**

***Index Terms*—Machine Learning, CNN, SVM, Image Classification**

I. Introduction

Image classification is a common task in machine learning, where images are categorized based on their visual features. It has a broad range of applications, including facial recognition, medical imaging, autonomous driving, and surveillance systems. Image classification problems can be binary (two categories) or multiclass (more than two categories) [1]. An example of multiclass classification is handwritten digit recognition using the MNIST dataset, where the model classifies images into one of ten classes (digits 0–9). In more advanced tasks such as object detection, models not only classify objects but also localize them within an image [2], [3].

A. CNN

For complex image classification tasks, deep learning techniques such as Convolutional Neural Networks (CNNs) are often used [4]. CNNs are optimized to process data structures, such as images. Their architecture includes convolutional layers, pooling layers, activation functions, and fully connected layers. Through backpropagation, a method that calculates the gradients of the loss function with respect to the network's weights based on labeled examples, these layers learn hierarchical representations of given features [5].

Convolutional layers apply filters to detect edges, textures, and shapes, while pooling layers reduce spatial dimensions to improve computational efficiency and reduce the chance of overfitting [3], [4]. A common pooling method is max pooling, which selects the maximum value within a specified window and down samples the feature map [6]

Activation functions introduce nonlinearity into the network. A commonly used activation function, Rectified Linear Unit (ReLU), accomplishes this by thresholding negative values to zero (ReLU(x)=max(0,x)), helping the network learn complex patterns and improving training efficiency due to its reduced computation [7]. Moreover, it can also help handle the vanishing gradient problem, an issue in the training of deep networks. During backpropagation, as the gradients of the loss function are propagated backward through the layers, activation functions with saturating regions (like sigmoid or tanh), the gradient can become very small as the activation moves into these regions, which can impact the learning in the earlier layers negatively. ReLU, with its constant gradient of 1 for positive inputs, allows gradients to significant reduction [8]. At the output layer, the softmax function is used for classification by converting raw scores into a probability distribution over possible classes [7]. The final fully connected layers of the CNN map extracted features to class labels, with the model performing accurate image classification.

B. SVM

Traditional machine learning algorithms, such as Support Vector Machines (SVMs), remain effective tools for image classification, especially in high-dimensional feature spaces. SVMs use a supervised machine learning algorithm to find an optimal hyperplane that maximizes the margin between classes, which results in reliable classification boundaries [9].

For multiclass problems, SVMs are extended using techniques such as one-against-one and one-against-all [10]. While SVMs do not extract features from raw image data, they can be effectively combined with preprocessing techniques or dimensionality reduction methods such as Principal Component Analysis (PCA). PCA is a non-parametric method that reduces data dimensionality while preserving features, improving SVM performance on large datasets [11].

The objectives of this research are as follows:

- 1) Replicate the image classification accuracy reported in [11], but implemented with PyTorch in a Google Colab environment.
- 2) Investigate the effect of varying batch size and learning rate on CNN training behavior and test accuracy.

- 3) Compare the CNN model's performance with that of an SVM classifier using a Radial Basis Function (RBF) kernel, trained on the same dataset.

The CNN implementation will use ReLU and softmax activation functions, consistent with [12]. The SVM classifier will use a Radial Basis Function (RBF) kernel for nonlinear classification.

II. Problem Statement

Traditional classifiers like SVMs and Decision Trees perform well on low-dimensional or linearly separable data. However, their computational cost and performance are impacted when applied to complex datasets with high-dimensional features, such as CIFAR-10, which consists of 10 image classes with 6,000 samples each. For full optimization, these datasets require more advanced models.

CNNs address these limitations through its capabilities in learning multi-level, spatially-aware features. With the availability of GPUs and open-source deep learning libraries such as TensorFlow and PyTorch, CNNs can now be trained on complex datasets using platforms like Google Colab or Jupyter Notebook.

In the conference paper "Classification of Image with Convolutional Neural Network and TensorFlow on CIFAR-10 Dataset", Sharma and Jadon achieved a 98.93% test accuracy using a six-layer CNN on Google Colab with TensorFlow over 25 training epochs [12]. However, their paper does not disclose key parameters such as batch size, regularization methods, or the implementation of batch normalization, which affects the reproducibility of their model. This work aims to replicate their results using PyTorch while analyzing how batch size and learning rate influence model performance. Additionally, the results are compared to those obtained using an SVM with an RBF kernel to examine the effect of deep learning on the dataset.

III. Materials and Method

In the paper [12], the authors proposed a simple workflow using the CIFAR-10 dataset to train a basic CNN with a limited number of convolutional layers, the SGD optimizer, a learning rate of 0.001, and no dropout regularization. Despite the model's simplicity, it yielded good results for both testing and validation accuracy. Fig. 1 shows their proposed image classification workflow. The model consists of standard components such as convolutional and max pooling layers, followed by fully connected layers and a softmax classifier. The authors also stated that they avoided using techniques such as dropout or batch normalization.

In this study, the CNN and core architecture were replicated as described in [12]; however, the model was implemented using the PyTorch framework instead of TensorFlow. This choice was made due to PyTorch's debugging capabilities and to evaluate the consistency and performance of different open-source deep learning libraries when implementing the same architecture. Additionally, PyTorch's extensive libraries made

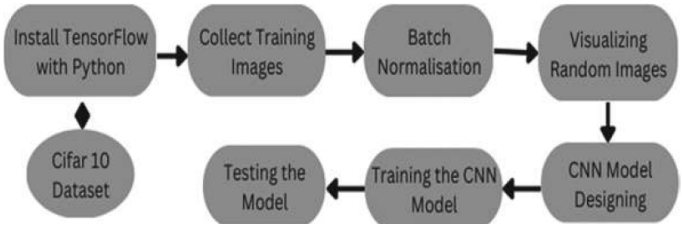


Fig. 1: Workflow of Proposed Image Classification [12].

it an ideal choice for reproducibility studies and future model development.

As noted, [12] did not specify certain parameters, which introduces minor experimental variations in training, such as different batch sizes and learning rates, to observe their effects on performance and computational efficiency.

A. Dataset

This study uses the CIFAR-10 dataset [13, 14], a commonly used benchmark in image classification research. Developed by the Canadian Institute for Advanced Research, CIFAR-10 contains 60,000 color images divided into ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each image is 32 × 32 pixels in resolution and the dataset is pre-processed and split into 50,000 training images and 10,000 test images. An example from the dataset is shown in Fig. 2.

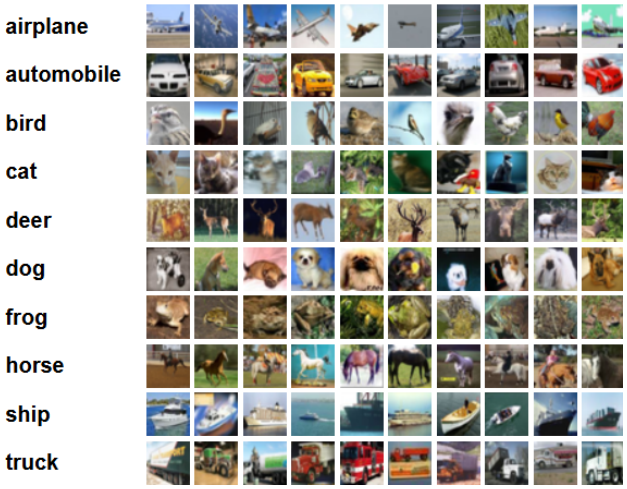


Fig. 2: Ten random images from each class [13, 14].

B. Image Pre-processing

The dataset was downloaded directly from the source website [13] and uploaded to Google Drive. The data path was modified in the code to the file's location, and the 'unpickle' function was used to extract the data files. The training and test datasets were then processed and normalized. All images were parsed into PyTorch tensors and standardized channel-wise. An additional validation set was created by randomly splitting the original training data into 80% training and 20% validation subsets. Datasets were wrapped in PyTorch

'DataLoader' objects with shuffling enabled for the training set.

C. CNN Architecture

The CNN model (Table I) follows the architecture described in [12], consisting of two convolutional blocks with ReLU activations and max pooling, followed by two dense layers and a softmax output.

TABLE I: Proposed CNN Model Architecture [12]

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 32)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 64)	73,792
dense_1 (Dense)	(None, 10)	650
Total params:		**84,586**
Trainable params:		**84,586**
Non-trainable params:		**0**

The model was implemented and trained using the 'CrossEntropyLoss' function and the Stochastic Gradient Descent (SGD) optimizer.

D. Mathematical Expressions

a) Convolutional Neural Network (CNN): The convolution operation applied to an input image f with a kernel h and a result matrix with a m rows and n columns is defined as [15]:

$$G(m,n) = (f*h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \quad (1)$$

b) Rectified Linear Unit (ReLU): The ReLU activation function introduces nonlinearity [7]:

$$f(x) = \max(0, x) \quad (2)$$

c) Max Pooling: Max pooling selects the highest value within a window of size X to downsample [6]:

$$f(X) = \max_{x \in X} x \quad (3)$$

d) Softmax: The softmax function at the output layer computes the discrete probability distribution $\sum_{k=1}^{K} p_k$ for K classes from raw scores o [7]:

$$p_k = \frac{exp(o_k)}{\sum_{k=0}^{n-1} exp(o_k)} \quad (4)$$

e) Loss function: The loss function used during training is categorical cross-entropy where the y_k is the true probability and trhe p_k is the predicted probability of class k [16]:

$$L = -\sum_{k=1}^{K} y_k \log(p_k) \quad (5)$$

f) Support Vector Machine (SVM): SVM finds the optimal separating hyperplane that maximizes the margin between classes. The decision boundary is defined as [9]:

$$f(x) = w^T x + b \quad (6)$$

For nonlinear separation, the Radial Basis Function (RBF) kernel is used [17]:

$$K(x, x_i) = \exp\left(-\tau \|x - x_i\|^2\right) \quad (7)$$

E. Training

The model was trained for 25 epochs using the following hyperparameters:

- Learning Rates: 0.0001, 0.001, and 0.5
- Batch Sizes: 20, 100, 500
- Optimizer: SGD with momentum = 0.9

Accuracy and loss were tracked for the training and validation sets per epoch. GPU acceleration was not available during training due to usage limits in Google Colab. Early stopping was not employed, though overfitting was monitored using validation loss. Each experiment was repeated five times to compute a confidence interval.

F. SVM Classifier

To compare with the CNN, a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel was trained on the same dataset. Images were flattened into 3072-dimensional vectors (32 × 32 × 3), and 'StandardScaler' was used to normalize the pixel values. A Principal Component Analysis (PCA) was applied to reduce dimensionality, with the optimal number of components determined to be $k = 448$.

The scikit-learn library's 'SVC' class was used with the following parameters:

- Kernel: RBF
- C (penalty parameter): 1.0
- Gamma: 'scale'

Training, validation, and testing accuracy were recorded. A classification report with accuracy, F1-score, recall, and precision was generated using scikit-learn, along with confusion matrices to visualize performance.

G. Performance Evaluation

To assess the performance of the classification models, the confusion matrix and several derived metrics (precision, recall, F1 score, and accuracy) were provided by classes from the Scikit-learn library such as 'classification_report' and 'confusion_matrix'.

a) Confusion Matrix: The confusion matrix provides a summary of prediction results. Each row of the matrix represents the instances in an actual class, while each column represents the instances in a predicted class [18].

b) Performance Metrics: The following metrics were computed for each class [18]:

- **True Positive (TP):** Correctly predicted positive.
- **False Positive (FP):** Incorrectly predicted positive (Type I error).
- **False Negative (FN):** Incorrectly predicted negative (Type II error).
- **True Negative (TN):** Correctly predicted negative.

1) Precision: Precision measures the ratio of correctly predicted positive observations to the total predicted positives:

$$\text{Precision} = \frac{TP}{TP + TN} \tag{8}$$

2) Recall: Recall (Sensitivity) measures the ratio of correctly predicted positive observations to all observations in the actual class:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9}$$

3) F1 Score: F1 score is the harmonic mean of precision and recall:

$$\text{F1-Score} = \frac{1}{\frac{1}{Recall} + \frac{1}{Precision}} \tag{10}$$

4) Accuracy: Accuracy is the ratio of correctly predicted instances over the total number of instances:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

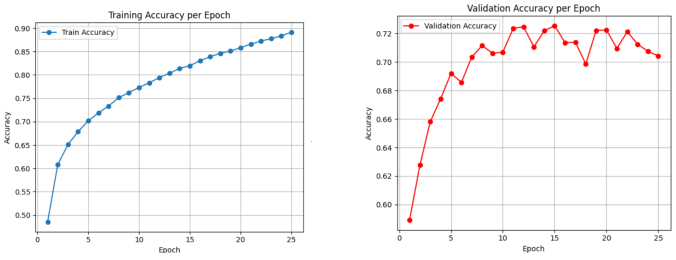
IV. Results

A. CNN Performance

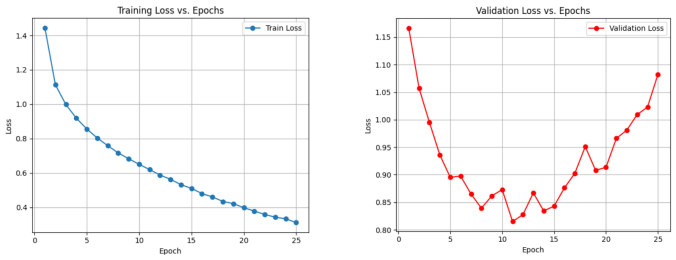
The PyTorch CNN achieved a peak test accuracy of 70.89% after 25 training epochs using a learning rate of 0.001 and a batch size of 20. The training and validation accuracy curves show a steady rise, but the validation loss graph indicates signs of overfitting, likely due to the relatively smaller validation dataset compared to the training set (Fig. 3). The confusion matrix showed good performance in distinguishing dissimilar classes such as "frog" and "truck," but had difficulty with visually similar classes like "cat" and "dog" (Fig. 4).

B. Effect of Batch Size and Learning Rate

To evaluate the model's sensitivity to hyperparameters, additional experiments were carried out by varying the batch size (20, 100, 500) and learning rate (0.0001, 0.001, 0.5). The overall results (Table II) indicated that lower learning rates improved stability but required more epochs to converge (Fig. 5). A high learning rate of 0.5 resulted in training instability and misclassifications, as shown in the confusion matrix (Fig. 6). And finally, as mentioned previously, smaller batch sizes slightly improved test accuracy but overfitting was observed (Fig. 3(d)). With the exception of the high learning rate case, all models had a common trend of misclassifying visually similar images.



(a) Training Accuracy (b) Validation Accuracy



(c) Training Loss (d) Validation Loss

Fig. 3: Training and Validation Metrics for CNN (Learning Rate = 0.001, Batch Size = 100)

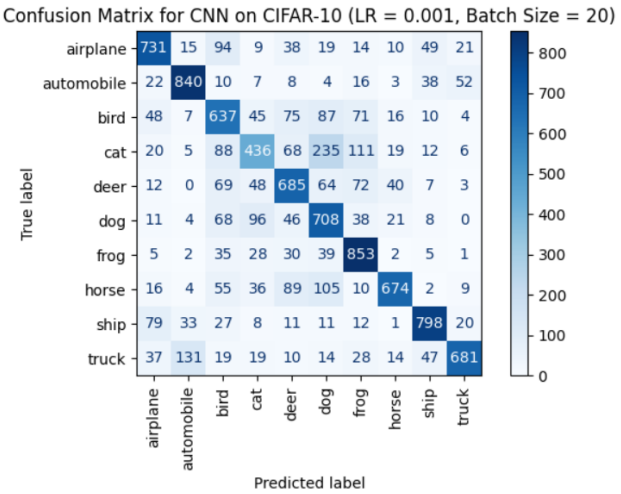


Fig. 4: Confusion Matrix for CNN (LR = 0.001, Batch Size = 20)

C. SVM Performance

As a non-deep learning baseline, the SVM classifier with a radial basis function (RBF) kernel was trained on PCA-reduced data (448 components), and yielded a test accuracy of 51.27%. Although considerably lower than the CNN's performance, this is consistent with expectations for traditional classifiers on high-dimensional image datasets.

Fig. 7 shows the SVM confusion matrix. Like the CNN, the SVM performed well on dissimilar classes (e.g., animals vs. vehicles), but poorly on visually similar ones.

TABLE II: Comparison of CNN Performance with Varying Hyperparameters and SVM

Model	Learning Rate	Batch Size	Performance	
			Test Accuracy (95% CI)	Weighted F1 Score
CNN	0.001	100	0.7018 (0.6929, 0.7108)	0.7061
CNN	0.0001	100	0.6069 (0.5950, 0.6187)	0.6110
CNN	0.5	100	0.1862 (0.1748, 0.1977)	0.0943
CNN	0.001	20	0.7089 (0.7048, 0.7129)	0.7036
CNN	0.001	500	0.6497 (0.6399, 0.6595)	0.6411
SVM (PCA)	N/A	N/A	0.5127	0.5112

*CI = Confidence Interval

(a) Training Accuracy (b) Validation Accuracy

(c) Training Loss (d) Validation Loss

Fig. 5: Training and Validation Metrics for CNN (Learning Rate = 0.0001, Batch Size = 100)

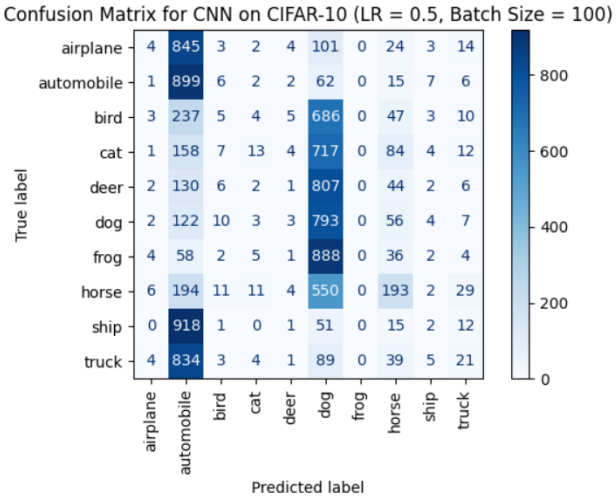


Fig. 6: Confusion Matrix for CNN (LR = 0.5, Batch Size = 100)

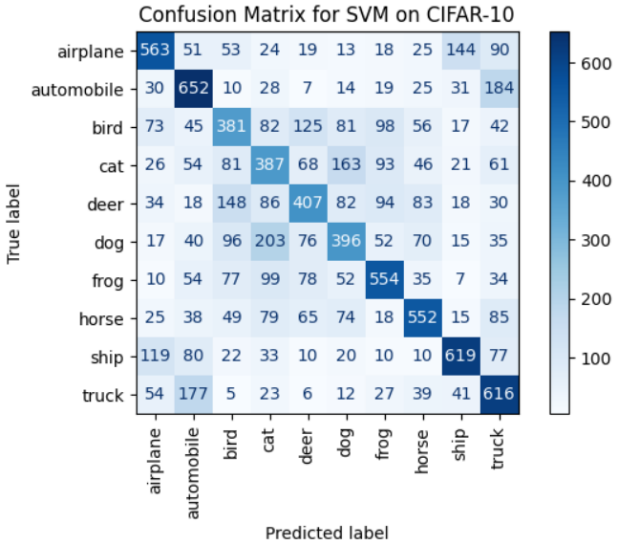


Fig. 7: Confusion Matrix for SVM with PCA-Reduced Dimensionality

D. Discussion

While Sharma and Jadon's study reported a test accuracy of 98.93% using a six-layer CNN, the PyTorch implementation here achieved a peak accuracy of only 70.89% [12]. This may be attributed to several factors, including differences in data augmentation, optimizer settings, or weight initialization that was not fully specified in their work. Additionally, [12] used the Keras version of the CIFAR-10 dataset, which may have involved preprocessing steps optimized for classification in TensorFlow, unlike the raw dataset used here [13].

In replicating the model, estimates in particular parameters were made. For example, momentum for the SGD optimizer was set to 0.9 as it was not previously specified. Overfitting observed at low learning rates might have been reduced by optimizing the batch size or introducing regularization techniques. Furthermore, optimizers like Adam could have improved performance.

Comparison with traditional models showed the advantages of CNNs. Training the SVM on the full dataset took over four hours and yielded significantly lower accuracy (51.27%), while the CNN completed five training iterations within the same timeframe. This showed the efficiency of mini-batch training in deep learning, especially for large-scale image datasets like

CIFAR-10. Although simpler classifiers were not explored, the SVM results suggested limited use for traditional methods in this context.

Training was also constrained by resource and time limitations. GPU/TPU availability in Google Colab was inconsistent, which extended total training time to over two days. Additionally, since [12] did not report their batch normalization method, implementation differences may have contributed to the differences in results. In this work, batch normalization was added after preliminary experiments showed poor convergence without it.

With additional time, future work could include:

- Using deeper architectures with dropout or L2 regularization.
- Increasing the number of epochs with early stopping or model checkpointing.
- Incorporating advanced optimizers.
- Re-implementing the model in TensorFlow to allow for framework comparison.

E. Data and Code Availability

The code for this project is publicly available and can be found here: GitHub

V. Conclusion

Despite following the general model structure, the replicated implementation in PyTorch yielded a peak test accuracy of only 70.89%, considerably lower than the reported 98.93%. However, it did outperform the SVM classifier (51.27%), the observed gap in accuracy from the reference study [12] suggests that reimplementation using different frameworks or further architectural refinement may be necessary to obtain a similar performance.

Acknowledgment

The Scikit Learn Python Library, Torch, and the Google Colab environment were used for the computations performed in this paper.

References

[1] *What is Image Classification?* GeeksforGeeks, May 2024. URL: https://www.geeksforgeeks.org/what-is-image-classification/.

[2] *Papers With Code : Image Classification*. Paperswithcode.com, 2011. URL: https://paperswithcode.com/task/image-classification.

[3] Waseem Rawat and Zenghui Wang. "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review". In: *Neural Computation* 29 (Sept. 2017), pp. 2352–2449. DOI: 10.1162/neco_a_00990.

[4] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. Dec. 2015. URL: https://arxiv.org/pdf/1511.08458.

[5] Hecht-Nielsen. "Theory of the backpropagation neural network". In: *International Joint Conference on Neural Networks* (1989). DOI: 10.1109/ijcnn.1989.118638. URL: https://ieeexplore.ieee.org/document/118638.

[6] Hossein Gholamalinezhad and Hossein Khosravi. *Pooling Methods in Deep Neural Networks, a Review*. URL: https://arxiv.org/pdf/2009.07485.

[7] Fred Abien and Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. URL: https://arxiv.org/pdf/1803.08375.

[8] Chi-Feng Wang and · Follow. *The Vanishing Gradient Problem The Problem, Its Causes, Its Significance, and Its Solutions Listen Share Title Image // Source*. URL: https://www.khoury.northeastern.edu/home/vip/teach/MLcourse/2.5_NN/VanishingGradientProblem.pdf (visited on 04/29/2025).

[9] Md IMRAN Hossain. *Support Vector Machine**. ResearchGate, Dec. 2022. URL: https://www.researchgate.net/publication/365892847_Support_Vector_Machine (visited on 10/02/2024).

[10] Chih-Wei Hsu and Chih-Jen Lin. *A Comparison of Methods for Multi-class Support Vector Machines*. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/multisvm.pdf.

[11] Jonathon Shlens. *A Tutorial on Principal Component Analysis*. URL: https://arxiv.org/pdf/1404.1100.

[12] Gunjan Sharma and Vijay K Jadon. "Classification of Image with Convolutional Neural Network and TensorFlow on CIFAR-10 Dataset". In: *Lecture notes in electrical engineering* (Jan. 2024), pp. 523–535. DOI: 10.1007/978-981-99-7077-3_51. (Visited on 03/21/2025).

[13] Alex Krizhevsky. *CIFAR-10 and CIFAR-100 datasets*. Toronto.edu, 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[14] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Apr. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[15] Piotr Skalski. *Gentle Dive into Math Behind Convolutional Neural Networks*. Medium, Apr. 2019. URL: https://medium.com/data-science/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

[16] Chris Hughes. *A Brief Overview of Cross Entropy Loss - Chris Hughes - Medium*. Medium, Sept. 2024. URL: https://medium.com/@chris.p.hughes10/a-brief-overview-of-cross-entropy-loss-523aa56b75d5.

[17] Yixiao Sun. *Support Vector Decision Making*. 2024. URL: https://econweb.ucsd.edu/~yisun/Support-Vector-Decision-Making.pdf (visited on 04/29/2025).

[18] Zohreh Karimi. *(PDF) Confusion Matrix*. ResearchGate, Oct. 2021. URL: https://www.researchgate.net/publication/355096788_Confusion_Matrix.