Benchmarking CNN vs. NN: A Performance Analysis with GPU and CPU comparison

Jayant Yadav
Dept. of Electronics and Communication Engineering
International Institute of Information Technology
Naya Raipur, India
jayant24101@iiitnr.edu.in

***Abstract*—Image classification has advanced significantly over the years, but this progress comes with increased model complexity. While high-end systems can handle these models with ease, low-resource devices struggle due to slower processing speeds and higher energy consumption, making real-time applications difficult. This study benchmarks two widely used models—Convolutional Neural Networks (CNNs) and traditional Neural Networks (NNs)—to analyze their performance under constrained computational conditions. Both models follow an identical architecture and are trained on the same datasets to ensure a fair comparison.**

***Index Terms*—Image Classification, Convolutional Neural Network, Neural Networks, Benchmarking, CPU vs GPU**

I. Introduction

Image classification has advanced significantly over the years, leading to increasingly complex models. While these high-capacity architectures perform well on powerful hardware, they are often impractical for low-end devices with limited computational resources.

This study benchmarks two fundamental image classification architectures—Convolutional Neural Networks (CNNs) and standard Neural Networks (NNs). To ensure a fair comparison, both models are designed with minimal complexity, avoiding batch normalization, regularization, and dropout while maintaining competitive accuracy.

To evaluate their efficiency across different hardware constraints, we test the speed and accuracy of both models on a CPU and GPU setup. Additionally, four diverse datasets are used to analyze performance under varying data conditions.

II. Methodology

The four data sets which have been used are as stated:

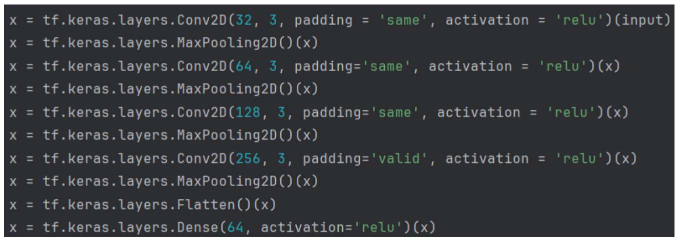
- SVHN
- CIFAR10
- FASHION-MNIST
- CIFAR100

Both the CNN and NN models are designed with five layers each. To maintain consistency, all datasets are trained for 10 epochs, except for CIFAR-100. Due to its larger size and increased classes (100 classes), CIFAR-100 is trained for 30 epochs for both CNN and NN. A batch size of 64 has been used for training, as it provides an optimal balance between efficiency and performance.

The model were trained upon Google Colab, the hardware provided by Colab being Intel Xenon for CPU and NVIDIA T4 as GPU

A. Architecture for CNN (Convolution Neural Network)

The architecture begins with an input layer which has a desired input shape per model, followed by 5 Conv2D and MaxPooling2d layers.



```
x = tf.keras.layers.Conv2D(32, 3, padding = 'same', activation = 'relu')(input)
x = tf.keras.layers.MaxPooling2D()(x)
x = tf.keras.layers.Conv2D(64, 3, padding='same', activation = 'relu')(x)
x = tf.keras.layers.MaxPooling2D()(x)
x = tf.keras.layers.Conv2D(128, 3, padding='same', activation = 'relu')(x)
x = tf.keras.layers.MaxPooling2D()(x)
x = tf.keras.layers.Conv2D(256, 3, padding='valid', activation = 'relu')(x)
x = tf.keras.layers.MaxPooling2D()(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
```

Fig. 1: Architecture of Convolution Neural Network (CNN)

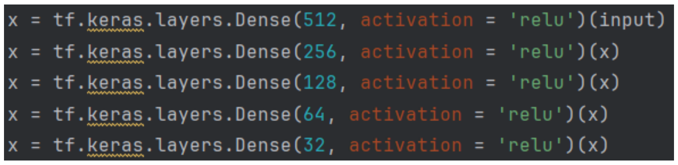
The model consists of four convolutional layers followed by a fully connected (dense) layer. Each layer doubles the number of filters onto the next (32 $\rightarrow$ 64 $\rightarrow$ 128 $\rightarrow$ 256), allowing the model to learn more complex patterns deeply. The kernel size remains fixed at 3×3 across all layers to maintain a consistency throughout.

For the first three convolutional layers, padding='same' is used to preserve spatial dimensions, preventing information loss at the edges. However, the last convolutional layer has padding='valid', if it is set to 'same' the spacial dimensions get reduced to negative which is not possible.

After the final convolutional layer, the output is flattened into a 1D vector, making it compatible with the fully connected layers. The dense layer has half the number of neurons compared to the previous layer, which helps reduce model complexity and also save some time while preserving essential information. The final output layer determines the class predictions.

B. Architecture for Neural Network (NN)

The architecture begins with an input layers which has a desired input shape per model, followed by 5 Dense layers.



```
x = tf.keras.layers.Dense(512, activation = 'relu')(input)
x = tf.keras.layers.Dense(256, activation = 'relu')(x)
x = tf.keras.layers.Dense(128, activation = 'relu')(x)
x = tf.keras.layers.Dense(64, activation = 'relu')(x)
x = tf.keras.layers.Dense(32, activation = 'relu')(x)
```

Fig. 2: Architecture of Neural Network

The number of neurons in the first layer is determined by the input size, which depends on the pixel dimensions of the dataset (length × breadth). This varies across datasets.

There are 5 fully connected Dense layers, the number of neurons in each layer half as we progress (512 $\rightarrow$ 256 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 32). This gradual decrease in the number of neurons ensures feature abstraction, the model only focuses on the details important while discarding the other noises.

Reducing the number of neurons ensures that there is no overfitting , increasing the computational efficiency.

C. Training Parameters

The models were trained using the Adam optimizer with a default learning rate of 0.001. The loss function used is Categorical Crossentropy, suitable for multi-class classification problems. Each hidden layer uses the Rectified Linear Unit (ReLU) activation function, while the Softmax function is applied at the output layer to obtain class probabilities.

All datasets were trained for 10 epochs, except for CIFAR-100, which was trained for 30 epochs due to its higher complexity (100 output classes). A batch size of 64 was maintained across all experiments for consistency and efficiency.

III. Experimental Evaluation and Comparison

This section presents the evaluation and comparison of the CNN and NN models on both CPU and GPU across key performance metrics. Visual representations are included wherever necessary to support the observations.

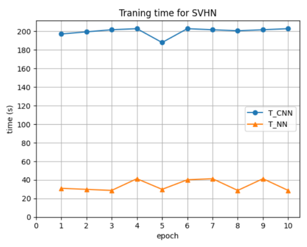
A. CPU performance (NN VS CNN)

1) Training Time per Epoch: The training time per epoch for CNNs was significantly higher—averaging over 110 seconds—compared to NNs. This disparity stems from the architectural complexity of CNNs: convolutional layers apply multiple filters to scan spatial hierarchies within the input image, substantially increasing computational overhead. As the number of filters increases with depth, so does the time required to process each image. In contrast, NNs operate on flattened pixel vectors without spatial awareness, allowing faster computation through sequential dense layers.

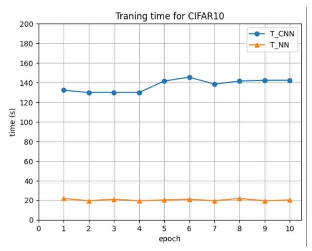
Padding—an essential operation in CNNs—also contributes to slower training on CPUs. Depending on the type used ('same' or 'valid'), padding introduces additional computations to preserve or reduce spatial dimensions, respectively. These extra operations place further load on the CPU, which lacks the parallel processing power of a GPU.

Additionally, VRAM—exclusive to GPUs—offers a major performance edge. CNN operations involving kernels, filters, and pooling layers are memory-intensive and benefit from GPU acceleration. CPUs, on the other hand, handle fully connected layers more efficiently, as they primarily rely on matrix multiplications using weights and biases.

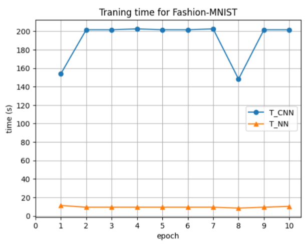
The graphs which show the plot pf training time per epoch are stated below, the blue line indicating time taken by CNN and the orange indicating time taken by NN.



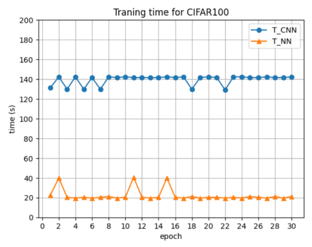
(a) SVHN



(b) CIFAR10



(c) Fashion-MNIST



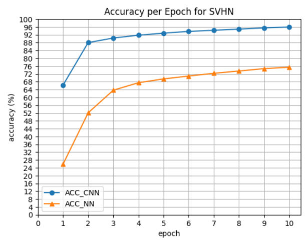
(d) CIFAR-100

Fig. 3: Time taken per epoch for training the datasets on each model.

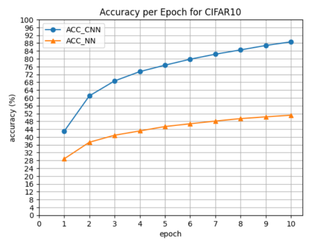
2) Accuracy per Epoch: While CNNs exhibit higher training times on CPU, they often compensate with faster and more stable convergence in terms of accuracy. Their ability to extract spatial hierarchies through convolutional layers enables them to achieve high accuracy even with relatively shallow architectures. In contrast, traditional NNs—lacking spatial awareness—tend to converge slower.

However, in grayscale datasets such as Fashion-MNIST, the performance gap narrows. NNs demonstrate competitive accuracy due to the reduced complexity of input channels and the alignment of grayscale image data with the structure of dense layers. The absence of color information simplifies the learning task, allowing dense networks to perform comparably in terms of classification performance.

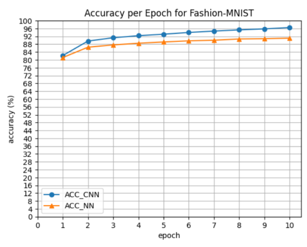
The graphs showcasing the accuracy per epoch and pasted below.



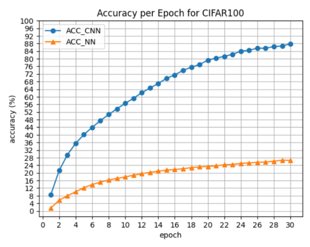
(a) SVHN



(b) CIFAR10



(c) Fashion-MNIST



(d) CIFAR-100

Fig. 4: Accuracy per Epoch for training the datasets on each model.

As shown in Figure 6c, the training accuracy for CNN and NN models on grayscale data (Fashion-MNIST) is nearly identical, despite CNNs taking significantly longer to train (see Figure 3c for time comparison). This indicates that for grayscale image classification on low-end systems, traditional NNs can offer comparable accuracy with far better efficiency. Converting input images to grayscale before training a dense network may thus be a practical alternative in resource-constrained environments.

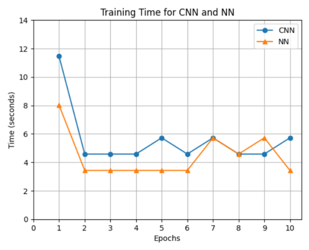
B. GPU performance (NN VS CNN)

1) Training Time per Epoch: Unlike the significant training time gap observed on CPU—where CNNs consistently took over 110 seconds per epoch and NNs were substantially faster—the difference on GPU is reduced to an average of just around 2 seconds.

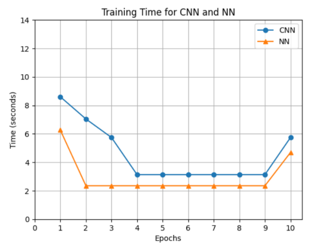
This dramatic shrink is attributed to the GPU's architecture, which is optimized for parallel operations and memory-intensive tasks. CNNs benefit from this the most due to their convolutional layers, padding, and pooling operations, all of which are accelerated by GPU cores and VRAM.

In contrast, NNs—being less computationally intensive—do see improvements on GPU, but the performance boost is not as drastic. As a result, the previously wide time gap between CNN and NN essentially collapses when both are run on GPU.

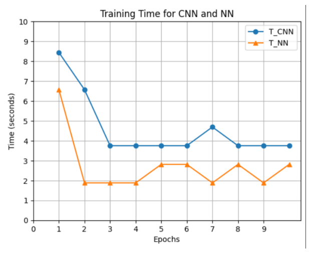
The graphs regarding the above are attested below.



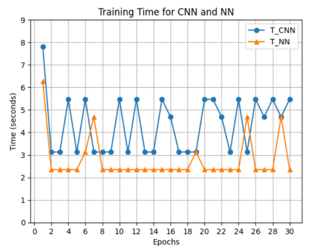
(a) SVHN



(b) CIFAR10



(c) Fashion-MNIST



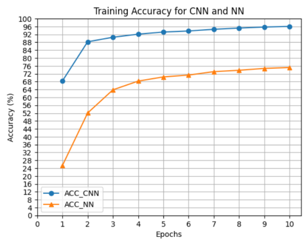
(d) CIFAR-100

Fig. 5: Training time per Epoch for training the datasets on each model.

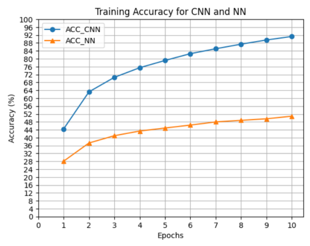
The four figures above illustrate that the training time for CNNs drastically drops when executed on GPU—nearly matching that of NNs. This suggests that even low-end GPUs can outperform CPUs when handling convolutional models, making them more suitable for training in power- or resource-constrained environments.

2) Accuracy per Epoch: While GPU usage significantly reduces training time, it does not substantially affect model accuracy. This is because VRAM and parallel computation primarily accelerate matrix operations—they do not enhance the model's ability to learn. Accuracy is governed by the model's architecture, capacity, and how well it generalizes from the dataset.

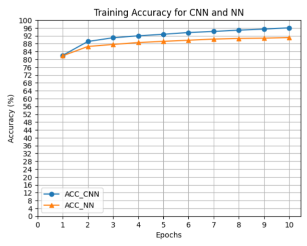
The accuracy-per-epoch graphs below closely resemble those in Figure 4, demonstrating that even low-end GPUs can yield fast training without sacrificing accuracy. However, improving accuracy—especially on complex datasets like CIFAR-100—requires more expressive architectures. Such changes, while effective, may be infeasible on low-power or resource-constrained devices.



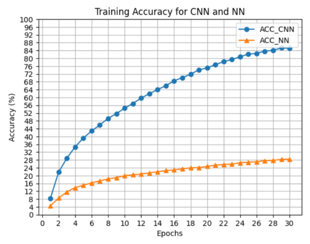
(a) SVHN



(b) CIFAR10



(c) Fashion-MNIST



(d) CIFAR-100

Fig. 6: Accuracy per Epoch for training the datasets on each model.

C. Generalization vs Memorization

While CNNs often achieve higher training accuracy due to their ability to capture spatial hierarchies, their lightweight architectures are prone to memorization rather than generalization. In contrast, NNs, despite slower convergence and lower training accuracy, tend to generalize better—indicating more effective learning and reduced risk of overfitting.

As the training accuracy and test accuracy remains identical for CPU and GPu, the data of GPU training accuracies and test accuracies has been selected.

The data for test accuracy and last training epoch is attested in the following table.

TABLE I: Final Training Accuracy vs Test Accuracy for CNN and NN Models

Dataset	Model	Train Accuracy (%)	Test Accuracy (%)	Generalization Gap (%)
CIFAR-10	CNN	91.29	74.38	16.91
	NN	50.89	47.72	3.17
Fashion-MNIST	CNN	96.07	91.88	4.19
	NN	91.10	87.06	4.04
SVHN	CNN	96.08	90.77	5.31
	NN	75.18	72.73	2.45
CIFAR-100	CNN	85.14	37.53	47.61
	NN	28.44	20.49	7.95

As observed in Table I, CNN models exhibit a significantly larger generalization gap compared to their NN counterparts across all datasets. This indicates a tendency for CNNs to memorize training data rather than generalize, particularly when constrained to lightweight architectures. The effect is most pronounced in complex datasets such as CIFAR-100, where CNNs achieve high training accuracy but fail to maintain similar performance on unseen test data—reflecting a gap of 47.61

In contrast, NNs—though achieving lower training accuracy—demonstrate a narrower generalization gap, suggesting more consistent learning. Notably, NNs perform this generalization with comparable training durations on GPU and even shorter times on CPU, making them more suitable for low-resource environments dealing with high-output-class datasets.

IV. Conclusion and Future Work

This study presented a systematic comparison between Convolutional Neural Networks (CNNs) and traditional Neural Networks (NNs) in terms of accuracy, computational time, and parameter efficiency across diverse datasets, including CIFAR-10, CIFAR-100, Fashion-MNIST, SVHN, and GTSRB. Through controlled experiments using uniform 5-layer architectures, we demonstrated that CNNs consistently outperform NNs on image classification tasks, particularly when spatial hierarchies in input data are present. Additionally, our results emphasized the significant influence of VRAM capacity and hardware acceleration (CPU vs GPU) on model training time and generalization capability.

While this work provides a foundation for hardware-aware model selection, it deliberately avoids the use of regularization techniques (e.g., dropout, batch normalization) to preserve architectural neutrality. The study also contributes insight into the performance gap between grayscale and RGB image processing within lightweight networks.

In future work, we aim to extend this investigation by incorporating Tensor Processing Units (TPUs) to benchmark their efficiency against GPUs in both training and inference phases. Moreover, we plan to explore model compression techniques such as pruning and quantization, enabling deployment in edge environments with constrained resources. The long-term goal includes real-time validation of these models in safety-critical systems, such as intelligent traffic sign recognition pipelines, to assess generalization under latency and memory pressure.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems, vol. 25, 2012.

[3] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, *Efficient processing of deep neural networks: A tutorial and survey*, Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[4] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural networks*, Advances in Neural Information Processing Systems, vol. 28, 2015.

[5] M. Abadi et al., *TensorFlow: A system for large-scale machine learning*, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 265–283, 2016.

Appendix

Appendix

A. Code and Resources

All code, datasets, and experimental notebooks are publicly available:

- **GitHub Repository:** https://github.com/Darkops-cpu/cnn-vs-nn-showdown.git

- **Colab Notebook 1:** https://colab.research.google.com/drive/1dZeni4hGdYHMLVAxyuxwxdFBVoXnidH9?usp=sharing
- **Colab Notebook 2:** https://colab.research.google.com/drive/1uosqazGG_HxTzaqZsG5mYhhDk0VnDTgi?usp=sharing

B. Google Colab Hardware Specifications

All experiments were conducted on Google Colab using the following configuration:

- **GPU:** NVIDIA Tesla T4 (16 GB VRAM)
- **CPU:** 2 × Intel Xeon @ 2.20 GHz (virtualized environment)
- **RAM:** 12.7 GB available
- **Platform:** Google Colab (Ubuntu 18.04 LTS backend)

C. Sample Hyperparameters Used

Uniform settings were applied across all experiments unless otherwise specified:

- **Learning Rate:** 0.001
- **Optimizer:** Adam
- **Batch Size:** 64
- **Epochs:** 20
- **Loss Function:** Categorical Crossentropy

D. Additional Graphs

Due to space constraints, complete training curves (e.g., Accuracy vs. Epoch, Loss vs. Epoch) are available in the GitHub repository under the `/visualizations` directory.