American International Journal of Computer Science and Information Technology
Volume.10, Number 5; September-October, 2025;
ISSN: 2837-388X | Impact Factor: 8.61
https://zapjournals.com/Journals/index.php/aijcsit
Published By: Zendo Academic Publishing

EFFICIENT DEEP CNN FOR HANDWRITTEN DIGIT RECOGNITION: A PYTORCH AND SKORCH APPROACH

Daniel U. Okon

Article Info
Keywords: Handwritten Digit Recognition; Deep Learning; Convolutional Neural Networks; PyTorch; Skorch; MNIST; Hyperparameter Optimization.

DOI
10.5281/zenodo.17136040

Abstract
Handwritten digit recognition remains a benchmark problem for evaluating machine learning and deep learning algorithms. In this study, we propose a simple yet effective convolutional neural network (CNN) architecture for multi-class classification of the MNIST dataset, implemented with PyTorch and Skorch. The model employs hyperparameter optimization via grid search to fine-tune the activation functions, kernel sizes, optimizers, and dropout rates. The experimental results demonstrate state-of-the-art performance, achieving 99% accuracy, precision, recall, and F1-score across all 10 digit classes. Comparative analysis against baseline models—including decision trees, SVMs, ANNs, and conventional CNNs—confirms that the proposed model consistently outperforms prior approaches. These findings highlight the effectiveness of lightweight, well-tuned CNN architectures for digit recognition tasks and demonstrate the utility of PyTorch and Skorch as efficient frameworks for model design, training, and deployment.

I. INTRODUCTION

In recent years, artificial intelligence (AI) has become increasingly mainstream. Guo et al. (2016) observed that ML and DL have emerged as specialized topics in response to the growing requirement to filter and comprehend huge volumes of visual information supplied by various applications. Deep learning is a subtype of machine learning in which algorithms can learn without being explicitly programed. Deep learning is a powerful method for automatically extracting characteristics from images, allowing us to accomplish difficult tasks such as object detection and categorization (Fourie, 2003). While a low-resolution image can affect a vast quantity of visual input, the information inside it remains essentially intact. Figure 1 shows the loss of detail and resolution from the original image. The image resolutions contain critical information for selecting where to search for the

Faculty of Computing, Department of Cybersecurity, University of Port Harcourt, Port Harcourt Rivers, Nigeria
E-mail: daniel.okon@uniport.edu.ng

appropriate classes. Recently, there has been an explosion in the amount of available visual materials. Object detection is linked to visual information. The significance of object recognition and categorization has increased dramatically. The ability to recognize visual objects is essential for human interaction and knowledge of the natural environment. Humans have outstanding visual recognition skills as proven by their near-instantaneous capacity to recognize familiar things, people, and meals. People can detect an object instantly regardless of changes in size, perspective, lighting, or orientation (DiCarlo et al., 2012). Several attempts have been made in computer vision research to develop systems capable of recognizing objects in the same way that humans do. Ideally, we would employ a strategy to create a deep learning model that excels at all types of problems and consistently generates excellent outcomes. The underlying idea of deep learning is that just as each problem in the real world provides a unique set of challenges, each set of obstacles necessitates a unique set of techniques and methods for overcoming them (Fourie, 2003). According to Pourjavan (2019), AIs deal with these obstacles by using a variety of deep learning models, each of which is made up of several layers of perceptron's that transform and adapt data to solve problems. Image segmentation, or the process of dividing an image into homogeneous segments, is required for a variety of image processing applications, such as object detection, image recognition, feature extraction, and classification. Because of the recent explosion in this field of study, deep learning has emerged as a critical tool for dealing with these types of difficulties, and it has broad potential applications across many industries (Jabreel and Moreno, 2019). As classified by LeCun et al. (2022), Figure 3 depicts the 11 categories into which the handwritten digits from the MNIST database fall. Each category corresponds to a different integer value ranging from 0 to 9. It is frequently used as a benchmark against which other ML and DL techniques can be measured (Priyansh et al., 2020), (Lejeune, 2020). It is a huge dataset of 700,000 photos, each 28 28 pixels in size and with a single grayscale dimension. During the training phase, 60,000 photographs are used, whereas only 10,000 images are used during the testing phase.

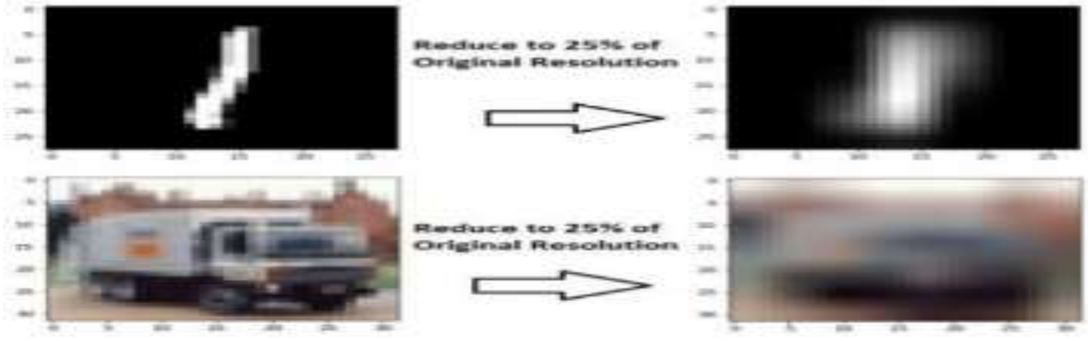

Reduce to 25% of Original Resolution
Reduce to 25% of Original Resolution

Fig. 1: Information loss due to the resolution

In this paper, we introduced a simple yet effective neural network technique that detects multiclassification in digital handprint images from the MNIST dataset using two libraries: PyTorch and Skorch. PyTorch is a free and open-source programming language that can be used for a variety of tasks, such as computer vision (Sikka, 2021), image segmentation (Liu, 2020), natural language processing (Rothman , 2021), and object identification (Ammirato et al., 2019). PyTorch was created in January 2016 by Facebook's Artificial Intelligence Research. Like Numpy arrays, PyTorch's Tensor multidimensional array data format enables quick iteration on the construction of advanced neural networks. The PyTorch framework allows researchers to rapidly create and train CNN models on huge datasets. Regarding PyTorch's versatility, speed, and user-friendliness, Kadam et al. (2020) and Jiang and Zhang (2021) have propelled it to the top of the list of the most extensively used deep learning tools in both the commercial world and academic institutions. The classification accuracy of the experiment was determined to be 88.39% using the Fashion MNIST data set. Ragab et al. (2022) proposed a capsule network-

based approach for detecting COVID-19 in X-ray images. The convolutional neural network was built using PyTorch, and the final model has a success rate of 95%. The real-time graph processing capabilities of PyTorch and the ease of tensor implementation on CPU and GPU are two of the reasons for its popularity. This quick and easy tweak to the built-in model optimization speeds debugging and provides a pipeline for deploying PyTorch models to cloud servers. Skorch is a high-level library for PyTorch that provides full scikit-learn compatibility, although a lack of visualization tools makes monitoring and analysis difficult (Lopez, 2020). It is a wrapper that converts between PyTorch and Scikit-learn. Consequently, models can be trained with Skorch using the well-known scikit-learn API. Scikit-learn is a free machine learning software library (Lopez, 2020) that includes several different types of results. Models can be trained with Skorch using the well-known scikit-learn API. Scikit Learn is a free machine learning software library (Lopez, 2020) that includes several classification, regression, and clustering models. Skorch is an expert in dealing with neural networks. It contains several useful features, such as the following: Neural Network Building Blocks: torch.nn. Callbacks, analytics, and other scorch bells and whistles that are supported through modules. Datasets can be directly passed to the fit method or packaged in a Data Loader object beforehand. Models may be trained in parallel across several devices (GPUs, CPUs, and TPUs) with minimal extra code, allowing compatibility with various Scikit-learn approaches. Skorch automatically manages resources, eliminating the need for any callback functions (Lopez, 2020). Skorch is a powerful tool that uses standard scikit-learn function sets to train, analyze, optimize, and upgrade ML models, such as grid search, learning rate schedulers, parameter freezing and unfreezing, and early stop and check.
The following are the main objectives of this study:

- To propose a simple yet effective and efficient deep CNN using PyTorch for multi-classification of handwritten MNIST digits.
- A precise comparison of the proposed and baseline models using several assessment metrics approaches.

The rest of the paper is divided into the following sections: Section II provides a detailed literature overview of the topic at hand, while Section III provides in-depth information regarding the data chosen. Section IV presents the proposed approach, while Section V presents the experimental setups and implementation of the model. Sections VI and VII explain the critical analysis and the results. Finally, Section VII provides a comprehensive summary of the topic and future research directions.

II. LITERATURE REVIEW

Image classification is a subset of computer vision that has significantly evolved over the last few years. Before the emergence of deep learning, machine learning performance was minimal. Deep neural networks are recognized for their ability to examine complex data structures. Neural networks with convolutional and fully-connected functionalities have been designed to attain state-of-the-art performance in a variety of applications, including speech recognition, image classification, natural language processing, and bioinformatics. Images from the MNIST database have been classified using numerous techniques explored in the literature. Hu et al. (2015) proposed a model in which neural networks are used to directly identify hyperspectral images in the spectral domain. The proposed classifier comprises five weighted layers that are applied to each spectral signature for discrimination, resulting in improved performance. The classification performance of this technique is superior to that of conventional methods, such as SVM and standard deep learning-based methods, as demonstrated by experimental findings on many hyperspectral image datasets on the MNIST dataset. Deep learning approaches often rely on the SoftMax layer to obtain the characteristics of lower parameters. Tang et al. (2013) proposed a model that implemented the linear SVM model instead of the SoftMax function. Learning minimizes the margin-based loss rather than decreasing the cross-entropy loss. The MNIST and CIFAR-10 datasets were used to measure the performance of the model. Removing the soft-max layer with SVMs is beneficial for classification tasks. Rather than using traditional feature extraction techniques, Ahlawat et al. (2020) used a hybrid model technique where CNN and SM models were used, CNN was used as a feature extraction model, and SM was used for binary handwritten image classification using the MNIST dataset. The comparison of binary and normalized preprocessed datasets and their implications on model performance has received relatively less attention. The preprocessing results are then used as input for neural networks to train the model (Shamsuddin et al., 2018). Feature extraction is a preprocessing phase that seeks to reduce the dimensionality of the input while obtaining useful information, as many classifiers struggle to effectively analyze the raw pictures or data. The efficacy of a classifier may depend on the features' quality as much as on the classifier's architecture. In the 1980s, neural networks were among the most widely used classifiers for character recognition, as proven by the effectiveness of back-propagation neural network models (LeCun et al., 2022). Lauer et al. (2007) emphasized the issues of feature extraction techniques for the recognition of MNIST handwritten data. To address the issue in a black box approach without requiring any pre-existing expertise with the data, Lauer proposed LeNet5, a trainable feature extractor neural network model, LeNet5. To improve the generalizability of LeNet5, support vector machines were used to perform the classification process. Mohapatra et al. (2015) described a multi-resolution feature extraction technique (discrete cosine S-transform) for image classification that extracts the 400 DCST feature vector from each normalized training input data. An ANN model is used along with feature extraction techniques to classify the handwritten images. For the MNIST dataset, this framework achieved 98% accuracy. Liu et al. (2003) conducted a thorough evaluation of the efficiency of many suggested classifiers, contrasting the linear and polynomial classifiers, the KNN classifier, and other neural networks. Cardoso et al. (2013) implemented the outcomes of a biologically based feature extraction model in a higher-dimensional space to train a linear classifier for digit recognition using the MNIST and USPS datasets. A comprehensive analysis study of two neural network models, CNN and LSTM, has been projected using the MNIST dataset to measure the model size, performance, time consumption, and complexity of each model for upcoming digital comprehension on reconfigurable hardware (Kaziha et al., 2019). Individuals do not typically write the same number in the same manner at a particular time. Because of this within-class variation in the form of a character, character classification presents a significant challenge. Multiple feature extraction techniques, such as biologically inspired model map transformation cascade (MTC) for feature extraction (Cardoso et al., 2013), higher order SVD (Savas et al., 2007), a genetic algorithm (GA)-based feature selection approach (De Stefano et al., 2014), fuzzy model-based recognition (Hanmandlu et al., 2007), and a kernel and Bayesian discriminant-based classifier (Wen et al., 2012), have been presented to classify the shape similarity within a class to enhance the recognition accuracy. Combining the structural risk reduction capability of SVM and the deep feature extraction capabilities of CNN has proven to be extremely effective in many applications (Tang et al., 2013; Xue et al., 2016; Ahlawat et al., 2020; Niu and Suen, 2012). This study demonstrates that the combination (neural network and ML models) of CNN and SVM could be quite effective for handwriting recognition. (Niu and Suen, 2012) combined CNN and SVM models for the MNIST digit database and claimed a performance accuracy of 97.81%.

III. DATASET SELECTION

In this study, we used the MNIST hand-printed dataset for the experiment. The MNIST hand-printed digits dataset is a set of images of hand-printed digits that are often used for training image recognition algorithms using various machine learning and deep learning approaches, as proposed by Kaziha et al. (2019), Niu and Suen (2012), and Tang et al. (2013). The dataset contains 60,000 training images and 10,000 test images, each of which is 28x28 pixels in size. The images are grayscale and have been normalized to fit within the range of

0-255. There are 10 target classes in the dataset ranging from digits 0 to 9. The images were taken in a range of sizes and contrasts, along with preprocessing with the removal of nondigital images. Many things are partially veiled. Figure 2 shows the MNSIT dataset.

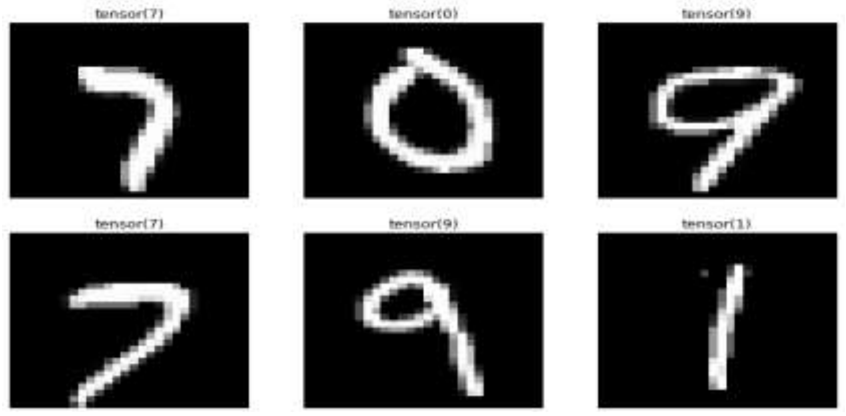


Fig. 2: MNIST dataset

IV. PROPOSED APPROACH

In This section, an overview of the neural network proposed for the multiclassification of handprint digits on MNIST. Using the PyTorch vision library (pytorch.org, n.d.), we were given access to a thoroughly vetted and ready-to-use data set. After being imported using Data Loader, the dataset was normalized and transformed to a tensor format using PyTorch. ToTensor is fundamental because it translates the visual into a numerical form that the model can comprehend. The image is scaled from 255 to 1 after being separated into its three-component channels (i.e., image, width, and height). Transformer Meanwhile, the tensor generated using ToTensor is normalized by adjusting the data so that the mean is close to zero and the standard derivation is close to one. Figure 3 shows a diagram of the suggested neural network's design and its parameters.

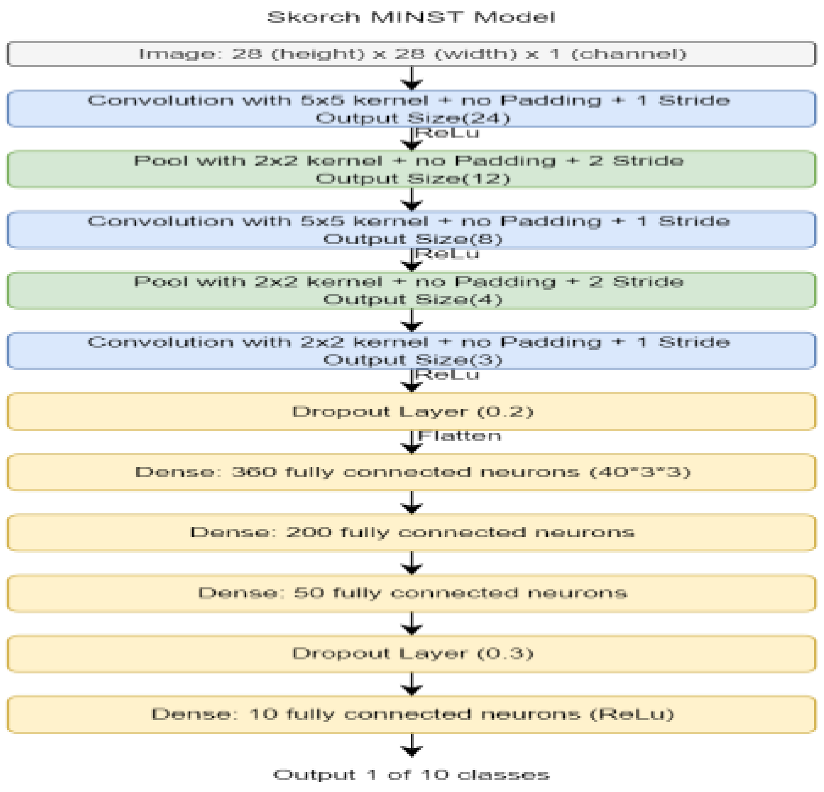

Skorch MINST Model
Image: 28 (height) x 28 (width) x 1 (channel)
Convolution with 5x5 kernel + no Padding + 1 Stride
Output Size(24)
ReLu
Pool with 2x2 kernel + no Padding + 2 Stride
Output Size(12)
Convolution with 5x5 kernel + no Padding + 1 Stride
Output Size(8)
ReLu
Pool with 2x2 kernel + no Padding + 2 Stride
Output Size(4)
Convolution with 2x2 kernel + no Padding + 1 Stride
Output Size(3)
ReLu
Dropout Layer (0.2)
Flatten
Dense: 360 fully connected neurons (40*3*3)
Dense: 200 fully connected neurons
Dense: 50 fully connected neurons
Dropout Layer (0.3)
Dense: 10 fully connected neurons (ReLu)
Output 1 of 10 classes

Fig. 3. Proposed model architecture.

The PyTorch model consists of following levels: the input layer, the output layer (with 10 neurons), and three hidden layers. The first two layers have a kernel size of 5x5 and a stride of 1, and the third has a kernel size of 2x2 with a stride of 1. There is no padding between the layers. For the proposed model, grid search was used to determine the best hyperparameters, including activation function, neurons for each convolution layer, optimizer, and learning rate. GridSearchCV performs an exhaustive search for an estimator over the parameter values provided. This module provides a flexible method for tuning model hyperparameters without having to directly define the logic required to execute the search. GridSearchCV (ghosh, 2022) is more efficient than trial and error because it only evaluates the parameter combinations you specify, rather than all potential combinations. GridSearchCV ensures that you are selecting the optimum combination of parameters because all possible combinations are tested. However, it is a lengthy process due to the need to examine every conceivable permutation that the given hyperparameter can produce must be examined. Table 1 shows the parameters of the proposed model used by the grid search to obtain the optimal features. The model generated 128 unique features with 384 model fits using a cross-validation threshold of 0.3 on the training dataset. Based on this, the following grid search settings were chosen as optimal: To prevent linearity and control how much the network model learns from the training dataset, the ReLU activation function selected through grid search was applied to the convolution hidden layers, which have 10, 20, and 40 input and output features, respectively. Two max pooling layers with a kernel size of 2x2 and stride 2 were also used to minimize the dimensionality of the feature maps and the number of learned parameters. This reduces the computational resources required for model training. Subsequently, the 2D convolution layer is converted into a fully connected linear layer, and the neuron size is determined using the following formula (OpenGenus, 2020):

$$\left(\frac{\mathrm{Cl} - \mathrm{KS} + 2\mathrm{PA}}{\mathrm{ST}}\right) + 1 \qquad (1)$$

CI, convolutional layer output size; KS, kernel size; PA, padding; and ST, stride. Similar to the input hidden layers, the output layer is a linear layer of 10 neurons, and its activation function, softmax (EDUCBA, 2022), returns the probability of all classes equaling 1.

$$\mathrm{Soft}(\mathrm{yi}) = \left(\frac{\exp(\mathrm{yi})}{\sum \mathrm{j}\exp(\mathrm{yi})}\right) \qquad (2)$$

The model is trained over 20 epochs with a batch size of 32 and a grid search-chosen cross-entropy loss. Cross entropy loss measures the effectiveness of the categorization model, with loss near zero representing the best model and loss near one indicating the worst model. The goal is to reach as close to zero as possible (Brownlee, 2019). Overfitting is a major issue that occurs when a model is trained in such a way that it becomes biased toward a single class or dataset. Dropout layers with dropouts of 0.2 and 0.3, calculated by grid search, were employed to regularize the model and reduce overfitting. The Adam optimizer was used to update the model’s properties, such as weights, and a learning rate of 0.001 was used to help tweak the weights so that the model could train even better. The neural network iterates several times with forward passes, and the model modifies the weights based on the loss with backward passes with each pass. Once the model is trained, its performance is evaluated using different assessment metrics. These metrics help us determine how accurately the model will predict the unseen dataset. The two metrics used to evaluate the model performance on the MNIST dataset are as follows:

Confusion Matrix

A confusion matrix is a table used to evaluate the classification model's accuracy. The table comprises four rows and four columns, each of which represents the number of correct and incorrect predictions made by the model for each class. The first row of the table represents the true positives of the classifier, while the second row represents its false positives. The third and fourth rows represent the classifier's false negatives and true negatives, respectively.

Classification Report

A classification report is another evaluation metric used to assess model performance. The report consists of the following evaluation metrics:

- Accuracy is calculated by dividing the sum of all correctly predicted values by the total number of values (true positives + false positives + false negatives + true negatives). This value can be interpreted as the percentage of times the classifier correctly predicts the label of a given instance.
- Precision is calculated by taking the sum of all true positive values and dividing it by the total number of predicted positive values (true positives + false positives). This value can be interpreted as the percentage of times the classifier correctly predicts a positive label.

$$\text{Pre} = \left(\frac{\text{All Positive}}{\text{All Postive} + \text{False Positive}}\right)$$

- The recall is calculated by dividing the sum of all true positive values by the total number of actual positive values (true positives + false negatives). This value can be interpreted as the percentage of times the classifier correctly detects a positive label.

$$\text{Recall} = \left(\frac{\text{All Positive}}{\text{All Positive} + \text{False Negative}}\right)$$

- The F1 score is calculated as the sum of precision and recall. This value can be interpreted as the balance between precision and recall.

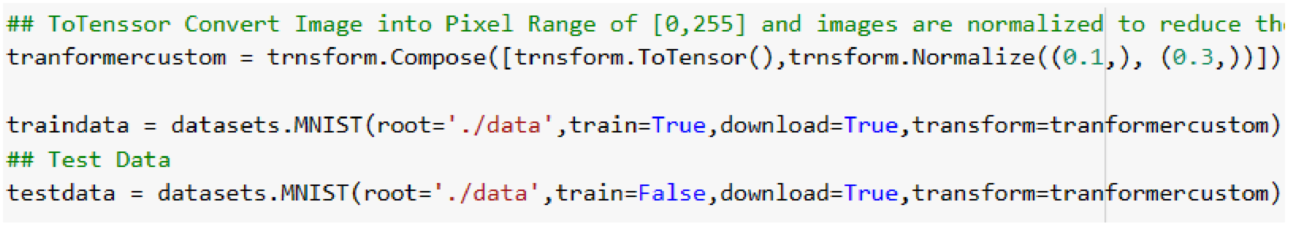
$$\text{F1 Score} = 2 \text{ X } \left(\frac{\text{Pre x Recall}}{\text{Pre} + \text{Recall}}\right)$$

Table. 1 Grid search parameters

Parameter	Values
Learning Rate (lr)	[0.0001, 0.001]
Epochs	[15,20]
Batch Size	[32,64]
Optimizer	[Adam, ReLu]
Nonlinear	[ReLu, Softmax]
Conv layer 1 Input Neuron	[10]
Conv layer 2 Out Neuron	[20,30]
Conv layer 3 Out Neuron	[40,50]

V. IMPLEMENATAION

The The experiment was conducted on a Jupyter notebook using Google collaboratory, a free cloud-based Jupyter notebook environment that requires no setup or installation and provides a prebuilt environment setup with most common libraries installed as well as gpu access for fast deep learning model computation. Pytorch was used for the experiments. The Pytorch library has various datasets, including cigar 10 and moist. Therefore, the MNIST dataset was loaded as shown in Figure 4.

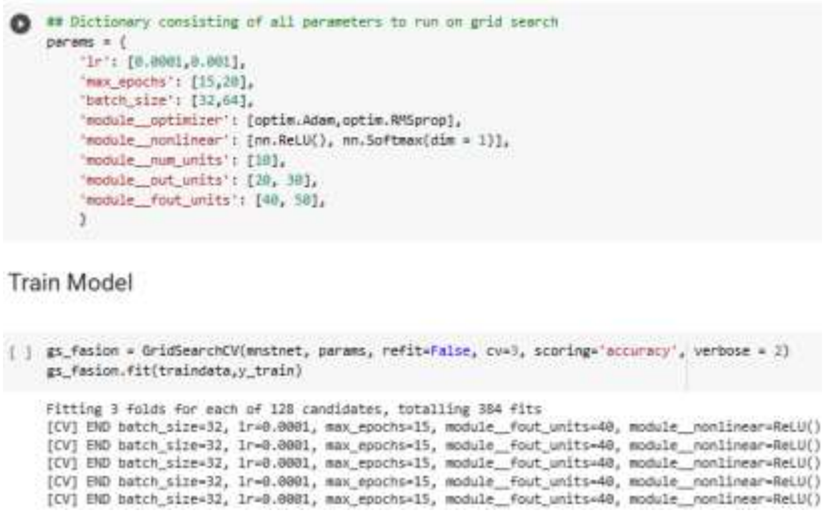


```
## ToTenssor Convert Image into Pixel Range of [0,255] and images are normalized to reduce th
tranformercustom = trnsform.Compose([trnsform.ToTensor(),trnsform.Normalize((0.1,), (0.3,))])

traindata = datasets.MNIST(root='./data',train=True,download=True,transform=tranformercustom)
## Test Data
testdata = datasets.MNIST(root='./data',train=False,download=True,transform=tranformercustom)
```

Fig. 4. MINST data loader

After loading the training and test datasets, Pytorch was used to generate the model, and Skroch was used to implement grid search to discover the most ideal parameters for the model to provide the best accuracy. Figures 5 and 6 depict the proposed model's implementation using grid search.



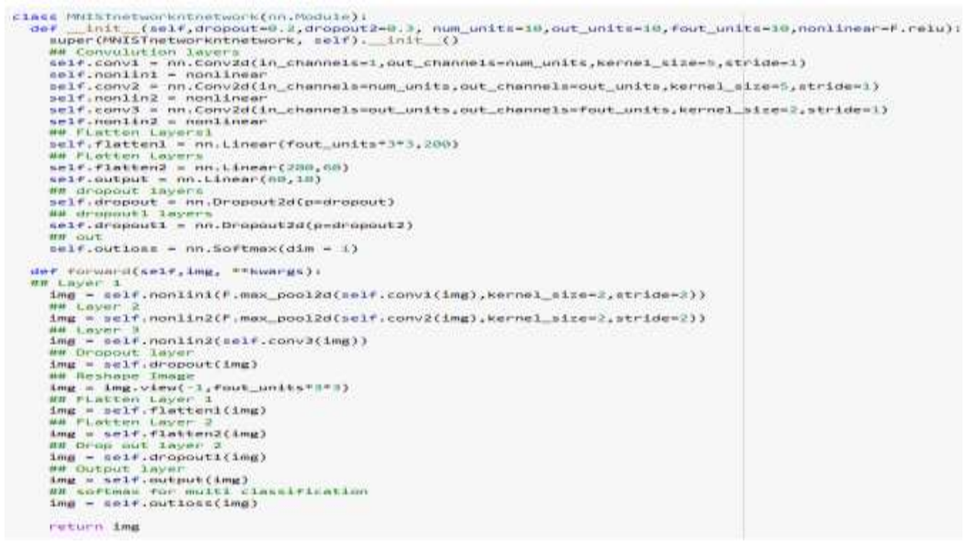
```
## Dictionary consisting of all parameters to run on grid search
params = {
    'lr': [0.0001,0.001],
    'max_epochs': [15,20],
    'batch_size': [32,64],
    'module__optimizer': [optim.Adam,optim.RMSprop],
    'module__nonlinear': [nn.ReLU(), nn.Softmax(dim = 1)],
    'module__num_units': [10],
    'module__out_units': [20, 30],
    'module__fout_units': [40, 50],
    }
```

Train Model

```
gs_fasion = GridSearchCV(mnstnet, params, refit=False, cv=3, scoring='accuracy', verbose = 2)
gs_fasion.fit(traindata,y_train)
```

```
Fitting 3 folds for each of 128 candidates, totalling 384 fits
[CV] END batch_size=32, lr=0.0001, max_epochs=15, module__fout_units=40, module__nonlinear=ReLU()
[CV] END batch_size=32, lr=0.0001, max_epochs=15, module__fout_units=40, module__nonlinear=ReLU()
[CV] END batch_size=32, lr=0.0001, max_epochs=15, module__fout_units=40, module__nonlinear=ReLU()
[CV] END batch_size=32, lr=0.0001, max_epochs=15, module__fout_units=40, module__nonlinear=ReLU()
[CV] END batch_size=32, lr=0.0001, max_epochs=15, module__fout_units=40, module__nonlinear=ReLU()
```

Fig. 5. Implementation of Grid Search



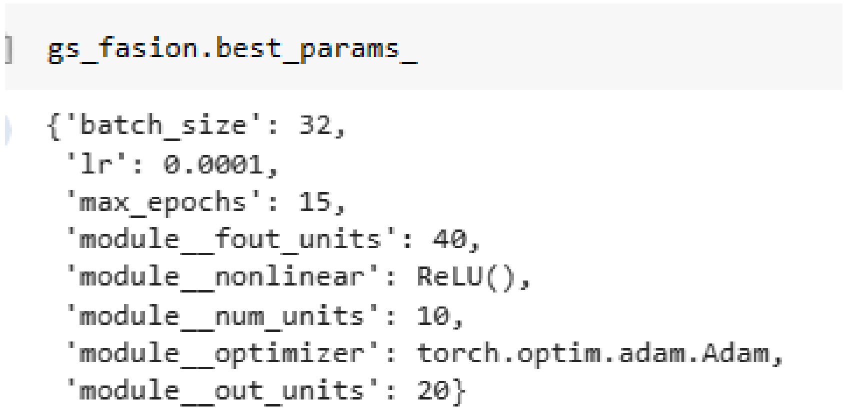
```
class MNISTnetworkntnetwork(nn.Module):
  def __init__(self,dropout=0.2,dropout2=0.3, num_units=10,out_units=10,fout_units=10,nonlinear=F.relu):
    super(MNISTnetworkntnetwork, self).__init__()
    ## Convulution layers
    self.conv1 = nn.Conv2d(in_channels=1,out_channels=num_units,kernel_size=5,stride=1)
    self.nonlin1 = nonlinear
    self.conv2 = nn.Conv2d(in_channels=num_units,out_channels=out_units,kernel_size=5,stride=1)
    self.nonlin2 = nonlinear
    self.conv3 = nn.Conv2d(in_channels=out_units,out_channels=fout_units,kernel_size=2,stride=1)
    self.nonlin2 = nonlinear
    ## Flatten Layers1
    self.flatten1 = nn.Linear(fout_units*3*3,200)
    ## Flatten Layers
    self.flatten2 = nn.Linear(200,60)
    self.output = nn.Linear(60,10)
    ## dropout layers
    self.dropout = nn.Dropout2d(p=dropout)
    ## dropout1 layers
    self.dropout1 = nn.Dropout2d(p=dropout2)
    ## out
    self.outloss = nn.Softmax(dim = 1)

  def forward(self,img, **kwargs):
  ## Layer 1
    img = self.nonlin1(F.max_pool2d(self.conv1(img),kernel_size=2,stride=2))
    ## Layer 2
    img = self.nonlin2(F.max_pool2d(self.conv2(img),kernel_size=2,stride=2))
    ## Layer 3
    img = self.nonlin2(self.conv3(img))
    ## Dropout layer
    img = self.dropout(img)
    ## Reshape Image
    img = img.view(-1,fout_units*3*3)
    ## Flatten Layer 1
    img = self.flatten1(img)
    ## Flatten Layer 2
    img = self.flatten2(img)
    ## Drop out layer 2
    img = self.dropout1(img)
    ## Output layer
    img = self.output(img)
    ## softmax for multi classification
    img = self.outloss(img)

    return img
```

Fig. 6. Proposed model code

The model was trained again with the optimal grid search settings. Figure 7 shows the optimal parameters discovered by the grid search. Skorch NeuralNetClassfier (López, 2020) was used as a wrapper to run the Pytorch model. As stated in Section IV, the model was rerun using 20 epochs, a batch size of 32, and a lr rate of 0.0001. Figure 8 displays the proposed Skroch model implementation.

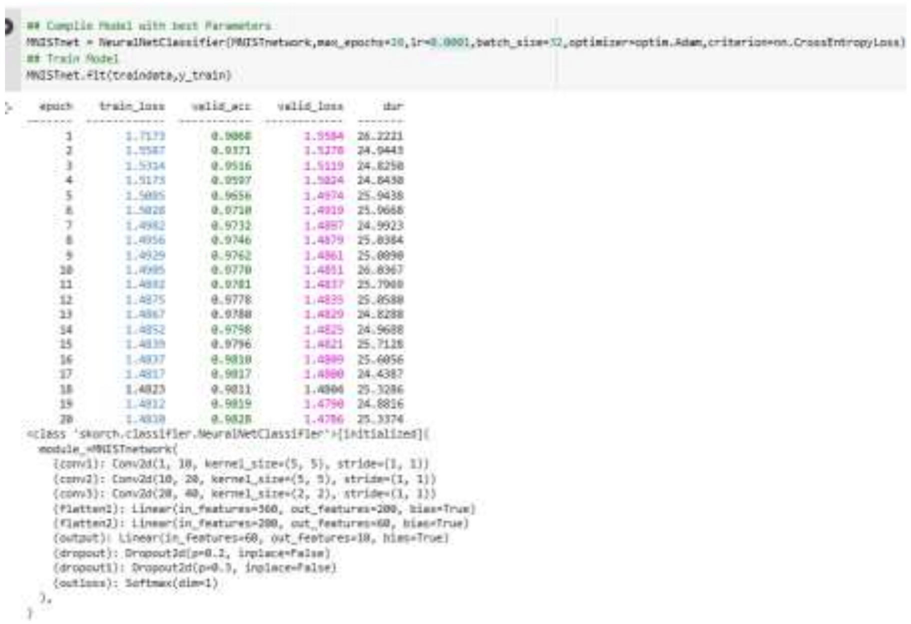


```
gs_fasion.best_params_
```

```
{'batch_size': 32,
 'lr': 0.0001,
 'max_epochs': 15,
 'module__fout_units': 40,
 'module__nonlinear': ReLU(),
 'module__num_units': 10,
 'module__optimizer': torch.optim.adam.Adam,
 'module__out_units': 20}
```

Fig. 7. Best Parameters

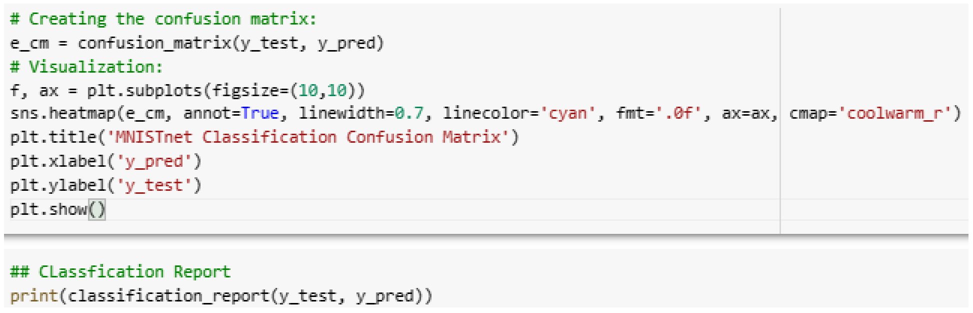
The model's performance was then evaluated on the test dataset using a confusion matrix and classification report. The model predicted the digits for each image in the test data and was then compared to the original digit labels for each image to determine the accuracy of the model. Figure 9 shows the creation of a confusion matrix and a classification report on a test dataset. Both functions are included in the sklearn package, so they are called and used to perform evaluations using these metrics.



```
## Complie Model with best Parameters
MNISTnet = NeuralNetClassifier(MNISTnetwork,max_epochs=20,lr=0.0001,batch_size=32,optimizer=optim.Adam,criterion=nn.CrossEntropyLoss)
## Train Model
MNISTnet.fit(traindata,y_train)
```

```
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.7173       0.9068        1.5584  26.2221
      2        1.5587       0.9371        1.5270  24.9443
      3        1.5334       0.9516        1.5119  24.8250
      4        1.5173       0.9597        1.5024  24.8430
      5        1.5085       0.9656        1.4974  25.9438
      6        1.5026       0.9710        1.4919  25.9668
      7        1.4982       0.9732        1.4897  24.9923
      8        1.4956       0.9746        1.4879  25.0384
      9        1.4929       0.9762        1.4861  25.0090
     10        1.4905       0.9770        1.4851  26.0367
     11        1.4882       0.9781        1.4837  25.7900
     12        1.4875       0.9778        1.4835  25.0580
     13        1.4867       0.9788        1.4829  24.8288
     14        1.4852       0.9798        1.4825  24.9688
     15        1.4839       0.9796        1.4821  25.7128
     16        1.4837       0.9810        1.4809  25.6056
     17        1.4817       0.9817        1.4800  24.4387
     18        1.4823       0.9811        1.4806  25.3286
     19        1.4812       0.9819        1.4790  24.8816
     20        1.4810       0.9828        1.4786  25.3374
<class 'skorch.classifier.NeuralNetClassifier'>[initialized](
  module_=MNISTnetwork(
    (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
    (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
    (conv3): Conv2d(20, 40, kernel_size=(2, 2), stride=(1, 1))
    (flatten1): Linear(in_features=360, out_features=200, bias=True)
    (flatten2): Linear(in_features=200, out_features=60, bias=True)
    (output): Linear(in_features=60, out_features=10, bias=True)
    (dropout): Dropout2d(p=0.2, inplace=False)
    (dropout1): Dropout2d(p=0.3, inplace=False)
    (outloss): Softmax(dim=1)
  ),
)
```

Fig. 8. Proposed Skorch Model Implementation



```
# Creating the confusion matrix:
e_cm = confusion_matrix(y_test, y_pred)
# Visualization:
f, ax = plt.subplots(figsize=(10,10))
sns.heatmap(e_cm, annot=True, linewidth=0.7, linecolor='cyan', fmt='.0f', ax=ax, cmap='coolwarm_r')
plt.title('MNISTnet Classification Confusion Matrix')
plt.xlabel('y_pred')
plt.ylabel('y_test')
plt.show()
```

```
## CLassfication Report
print(classification_report(y_test, y_pred))
```

Fig. 9. Confusion matrix and classification report

VI. EXPERIMENTAL ANALYSIS AND RESULTS

This section summarizes the experimental process used in this research. The performance of the model was evaluated using the MNIST pictures dataset. Figures 11 and 12 exhibit the confusion matrix and classification report, respectively, to demonstrate the effectiveness of the model. The proposed model performed remarkably well in trials on the MNIST dataset. The proposed model went through a learning curve using the training dataset. To train the model, we used batch processing. A training set and a test set were generated from the entire dataset. The training set contains sixty thousand images, whereas the test set contained 10,000 images. The proposed model's loss was determined using categorical crossentropy throughout the training procedure. The model performed admirably in the analysis. After putting the proposed model through its paces using test data that neither the model nor the testers have seen before, it obtains an accuracy of 0.99. The proposed model performed remarkably well on the MNIST dataset using the testing set, with a 0.99 precision score and 0.99 recall and F1-score, respectively. Figure 10 displays the loss of the proposed model during training and testing, showing a consistent decrease in loss with each training and testing epoch. This indicates that the model learned its parameters well and did not overfit or underfit. Figure 12 shows the classification analysis results of the proposed model. The complete findings of the proposed model's application to the MNIST dataset are detailed in the Classification report. The number of samples included in the testing set is displayed on the support. The classification report includes class-specific precision, recall, and f1 scores, as well as the proposed model's overall accuracy and the results of the other assessment metrics via macro and weighted average measures. Figure VII graphically depicts the confusion matrix. The confusion matrix displays the correct and incorrect classifications of the proposed neural network. Classifier-corrected examples are shown in blue, while cases misclassified by the neural network are indicated in red. The confusion matrix demonstrates the effectiveness of the model in class identification, revealing that just 142 out of 10,000 rows were incorrectly labeled. The digit 9 class had the most miss classified labeled rows, followed by the digit 2 class. The PyTorch model contains only four misclassified images in Class 1, making it the most accurate class.

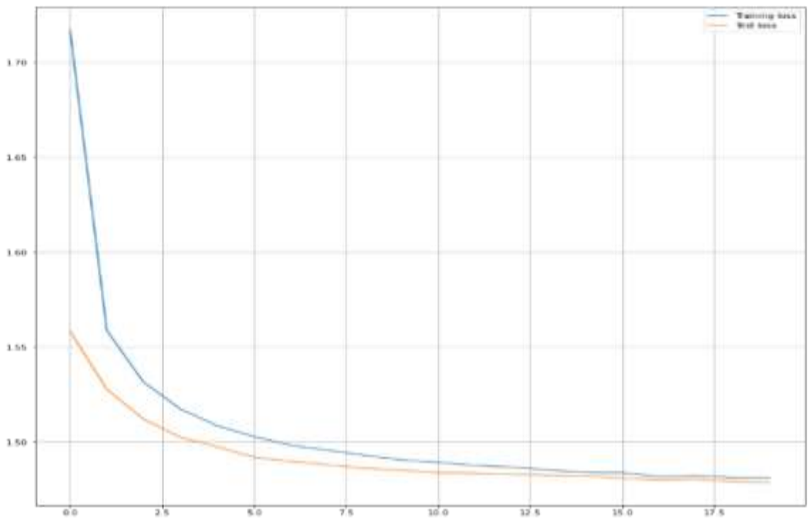
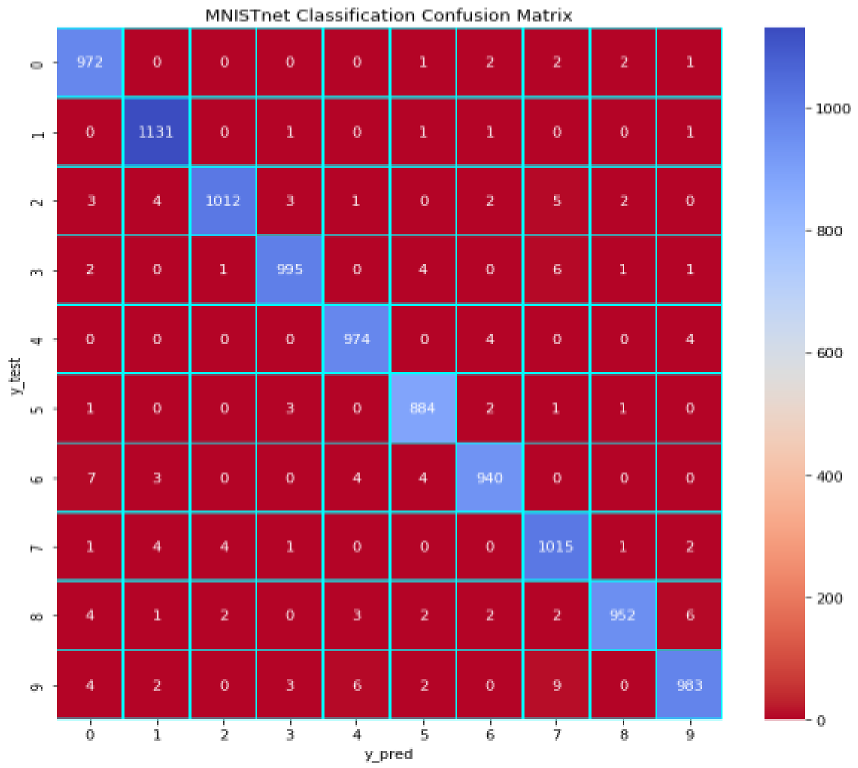
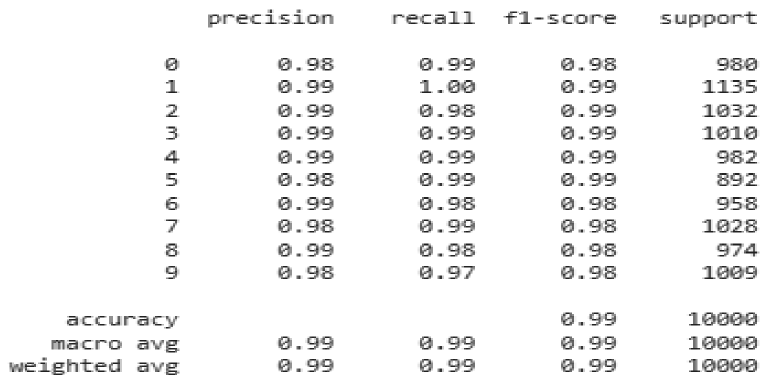


Fig. 10. Training and test loss



y_test \ y_pred	0	1	2	3	4	5	6	7	8	9
0	972	0	0	0	0	1	2	2	2	1
1	0	1131	0	1	0	1	1	0	0	1
2	3	4	1012	3	1	0	2	5	2	0
3	2	0	1	995	0	4	0	6	1	1
4	0	0	0	0	974	0	4	0	0	4
5	1	0	0	3	0	884	2	1	1	0
6	7	3	0	0	4	4	940	0	0	0
7	1	4	4	1	0	0	0	1015	1	2
8	4	1	2	0	3	2	2	2	952	6
9	4	2	0	3	6	2	0	9	0	983

Fig. 11. Model confusion matrix


```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.99      1.00      0.99      1135
           2       0.99      0.98      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.98      0.99      0.99       892
           6       0.99      0.98      0.98       958
           7       0.98      0.99      0.98      1028
           8       0.99      0.98      0.98       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

Fig.12. Model classification report

VII. CRITICAL ANALYSIS

This section compares the performance of the proposed model to that of various baseline models. The experiment was conducted in a nearly identical scenario, using the same dataset and the same distribution of test and training datasets. The proposed approach's performance was examined using both machine learning and deep learning models. The MNSIT dataset was also employed in the baseline models. In each baseline research, the experiment was conducted with several parameterized settings, and the results were displayed. Each baseline model employed a different assessment measure although all used accuracy. The baseline models used several ML and DL techniques, from decision trees to CNNs with varying parameters, layers, and epochs. Table II compares the proposed model to various baseline models. According to the table, our proposed model performed the best and had a maximum accuracy of 0.99%, whereas all baseline models had the lowest accuracy. The results unequivocally demonstrate that the proposed model outperforms the existing techniques. The model gained approximately 10% accuracy gain from diverse machine learning models, which is a significant margin. It also gained 7% and 1% accuracy gain from different AI and CNNs, respectively. Furthermore, the substantial accuracy improvement from ML suggests that DL works better for image classification because convolution models can easily extract significant features during the training phase, which ML models cannot do.

Table 2. Comparison of baseline models

VIII. DISCUSSION

This study reinforces the role of big data analytics and deep learning in addressing the classification task scalability and accuracy challenges. Traditional ML models struggle to manage high-dimensional datasets and achieve limited performance in image recognition and similar domains (Guo et al., 2016; Ahlawat & Choudhary, 2020). Convolutional neural networks (CNNs) have consistently demonstrated superior performance for digit recognition owing to their ability to automatically extract spatial features (LeCun et al., 2022; Kadam et al., 2020). The integration of PyTorch and Skorch proved effective in simplifying model development and hyperparameter optimization, enabling efficient training and deployment of CNN architectures (López, 2020; Han & Zhang, 2022). The grid search optimization significantly enhanced the model's performance, aligning with recent findings that hyperparameter tuning is critical for achieving state-of-the-art results in DL (Rahman & Hossain, 2023).

Baseline Models	**Test accuracy (%)**
Decision tree (Gope et al., 2021)	0.90
SVM (Gope et al., 2021)	0.95
Random Classifier (Gope et al., 2021)	0.93
Naïve Bayes (Gope et al., 2021)	0.90
KNN (Gope et al., 2021)	0.86
CNN (Kadam et al., 2020)	0.98
ANN (Pandey et al., 2020)	0.93
Proposed CNN model	0.99

The experimental results—achieving 99% accuracy, precision, recall, and F1-score—are consistent with recent studies that highlight the dominance of CNNs in visual recognition tasks (Khan et al., 2021; Zhang, Sun & Lin, 2023). The proposed CNN architecture achieved a substantial accuracy margin compared with baseline machine learning models such as decision trees, SVMs, and KNN, echoing the observation that deep models capture features inaccessible to conventional algorithms (Hu et al., 2015; Singh, Sharma & Kumar, 2022).

Although the model performed exceptionally well on MNIST, it shares the following limitations of benchmark-based studies: minority misclassifications and dataset simplicity. Recent research emphasizes extending CNN architectures to more complex datasets, such as Fashion-MNIST and CIFAR-10 (Doshi, Patel & Shah, 2022; Ragab et al., 2022). Additionally, CNNs are vulnerable to crafted perturbations that can mislead classifiers (Singh et al., 2022). Therefore, future work should address class imbalance, adversarial defense mechanisms, and TL approaches to ensure broader applicability in real-world scenarios (Alqahtani & Wang, 2024).

IX. CONCLUSION AND FUTURE WORK

This study presented a simple yet effective convolutional neural network using PyTorch for multiclassification of the MNIST hand-written image dataset, which is lightweight, organized, and preprocessed and has 10 classes. The MNIST dataset is an invaluable resource for classification tasks and proof-of-concept projects, as well as for gaining insights into DL techniques. Our two major libraries for implementing the deep neural network are PyTorch and Skorch. The main implementation of the model was performed in Pytorch, with Skroch used as a wrapper atop it to help determine the most effective hyperparameters using grid search. The grid search iteratively tests the model performance with multiple parameters and provides the optimal parameters for the final model implementation. To assess the model's performance across all ten classes, various assessment metrics, such as precision, recall, f1-score, accuracy, and confusion matrix, were established. The proposed model performed remarkably well, with an overall precision, recall, f1-score, and accuracy of 99%. Furthermore, comparison research was conducted using several baseline studies that employ approaches ranging from ML to DL algorithms. The comparison with baseline models clearly reveals that the suggested approach performs extremely well on the MNSIT dataset, with an accuracy improvement of over 10% from ML models and nearly 7% and 1% accuracy gains from ANN and CNN models, respectively. The results show that the proposed deep learning model achieved a very high classification accuracy on the MNIST dataset. Furthermore, the model was able to learn from the data with very little pre-processing, which is a significant advantage. In conclusion, this study has shown that deep learning is a powerful tool for image classification and can be applied to real-world datasets with excellent results.

REFERENCES

Ahlawat, S., & Choudhary, A. (2020). Hybrid CNN-SVM Classifier for Handwritten Digit Recognition. Procedia Computer Science, 167, pp.2554–2560. doi:10.1016/j.procs.2020.03.309.

Ammirato, P., & Berg, A. (2019). A mask-RCNN baseline for probabilistic object detection. [online] Available at: https://arxiv.org/pdf/1908.03621.pdf [Accessed December 22, 2022].

Cardoso, Â. and Wichert, A. (2013). Handwritten digit recognition using biologically inspired features. Neurocomputing, 99, pp.575–580. doi:10.1016/j.neucom.2012.07.027.

De Stefano, C., Fontanella, F., Marrocco, C., & Di Freca, A. S. (2014). GA-based feature selection with ANFIS Approach to Breast Cancer Recurrence. (2016). International Journal of Computer Science Issues, 13(1), pp.36–41. doi:10.20943/ijcsi-201602-3641.

DiCarlo, James J., Zoccolan, D., & Rust, Nicole C. (2012). How does the brain solve visual object recognition? Neuron, 73(3), 415–434. https://doi.org/10.1016/j.neuron.2012.01.010.

EDUCBA. (2022). PyTorch SoftMax | Complete Guide on PyTorch Softmax? [online]. Available from: https://www.educba.com/pytorch-softmax/.

Fourie, C. M. (2003). Deep learning? What deep learning? South African Journal of Higher Education, 17(1). https://doi.org/10.4314/sajhe.v17i1.25201.

Guo Y, Liu Y, Oerlemans A, Lao S, Wu, S. and Lew, M.S. (2016). Deep learning for visual understanding: A review. Neurocomputing [online] 187, pp.27–48. doi:10.1016/j.neucom.2015.09.116.

Hanmandlu, M., & Murthy, O. V. R. (2007). Fuzzy model based recognition of handwritten numerals. Pattern Recognition, 40(6), pp.1840–1854. doi:10.1016/j.patcog.2006.08.014.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. [online]. Available from: https://openaccess.thecvf.com/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf.

Hu, W., Huang, Y., Wei, L., Zhang, F., Li, H., 2015. Deep Convolutional Neural Networks for Hyperspectral Image Classification. Journal of Sensors, [online] 2015, p.e258619. doi:10.1155/2015/258619.

Jabreel, M., & Moreno, A. (2019). A Deep Learning-Based Approach for Multi-Label Emotion Classification in Tweets. Appl Sci, 9(6), 1123. doi:10.3390/app9061123.

Jiang, L., & Zhang, Z. (2021). Research on Image Classification Algorithm Based on Pytorch. Journal of Physics: Conference Series, 2010(1), p.012009. doi:10.1088/1742-6596/2010/1/012009.

Kadam, S. S., Adamuthe, A. C., & Patil, A. B. (2020). CNN model for image classification on MNIST and fashion-MNIST dataset. Journal of Scientific Research, 64(02), 374–384. https://doi.org/10.37398/jsr.2020.640251.

Kaziha, O., & Bonny, T. (2019). A comparison of quantized convolutional and LSTM recurrent neural network models using MNIST. [Online] IEEE Xplore. https://doi.org/10.1109/ICECTA48151.2019.8959793.

Lauer, F., Suen, C. Y. and Bloch, G. (2007). A trainable feature extractor for handwritten digit recognition. Pattern Recognition, 40(6), pp.1816–1824. doi:10.1016/j.patcog.2006.10.011.

LeCun, L., Henderson, J., Le Cun, Y., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (2022). Handwritten Digit Recognition with a Back-Propagation Network. Journal of Computer Science. [Online]. Available from: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

Lejeune, E. (2020). Mechanical MNIST: A benchmark dataset for mechanical metamodels. Extreme Mechanics Letters, 36, p.100659. doi:10.1016/j.eml.2020.100659.

Liu, C. L., Nakashima, K., Sako, H., & Fujisawa, H. (2003). Handwritten digit recognition: benchmarking of state-of-the-art techniques. Pattern Recognit [online] 36(10), pp.2271–2285. Doi: 10.1016/s0031-3203(03)00085-2.

Liu, Y. (2020). 3D Image Segmentation of MRI Prostate Based on a Pytorch Implementation of V-Net. Journal of Physics: Conference Series, 1549, 042074. https://doi.org/10.1088/1742-6596/1549/4/042074.

López, F. (2020). SKORCH: PyTorch Models Trained with a Scikit-Learn Wrapper. [Online] Medium. Available from: https://towardsdatascience.com/skorch-pytorch-models-trained-with-a-scikit-learn-wrapper-62b9a154623e

Mohapatra, R., Majhi, B., & Kumar, S. (2015). Classification performance analysis of MNIST dataset utilizing a multi-resolution technique. [Online]. Available at: http://dspace.nitrkl.ac.in:8080/dspace/bitstream/2080/2403/1/Classification_Mohapatra_2015.pdf [Accessed 30 Dec. 2022].

Niu, X.-X. And Suen, C.Y. (2012). A novel hybrid CNN–SVM classifier for recognizing handwritten digits. Pattern Recognit 45:1318–1325. https://doi.org/10.1016/j.patcog.2011.09.021.

Overview of classification methods in Python with Scikit-Learn", Stack Abuse, 2022. [Online]. Available: https://stackabuse.com/overview-of-classification-methods-in-python-with-scikit-learn/.

Pourjavan, S. (2019). Definitions: machine learning, deep learning and AI understanding. Acta Ophthalmol. 97, S263. https://doi.org/10.1111/j.1755-3768.2019.8214.

Priyansh, P., Ritu, G., Mazhar, K., & Sajid, I. (2020). Multi-digit number classification using MNIST and ANN. International Journal of Engineering Research and, V9 (05). Doi: 10.17577/ijertv9is050330.

Ragab, M., Alshehri, S., Alhakamy, N. A., Mansour, R. F., & Koundal, D. (2022). Multiclass Classification of Chest X-Ray Images for the Prediction of COVID-19 Using Capsule Network. Computational Intelligence and Neuroscience, 2022, pp.1–8. doi:10.1155/2022/6185013.

Rothman, D. (2021). Transformers for natural language processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more. XXXXXXXXXXXX. [online] Google Books. Packt Publishing Ltd. Available from: https://books.google.com.ng/books?hl=en&lr=&id=Cr0YEAAAQBAJ&oi=fnd&pg=PP1&dq=++Rothman [Accessed 27 Dec. 2022].

Savas, B., & Eldén, L. (2007). http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A18008. [Online] Available from: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A18008&dswid=5745 [Accessed 20 Dec. 2022].

Shamsuddin, M. R., Abdul-Rahman, S., & Mohamed, A. (2018). Exploratory Analysis of MNIST Handwritten Digit for Machine Learning Modeling. Communications in Computer and Information Science, pp.134–145. Doi: 10.1007/978-981-13-3441-2_11.

Sikka, B. (2021). Elements of deep learning for computer vision: explore deep neural network architectures, PyTorch, object detection algorithms, and computer vision applications for Python coding. [Online] Google Books. BPB Publications. Available from: https://books.google.com.ng/books?hl=en&lr=&id=n8U0EAAAQBAJ&oi=fnd&pg=PT19&dq=Sikka [Accessed 24 Dec. 2022].

Tang, Y. (2013). Deep learning using linear support vector machines. [Online]. Available at: https://arxiv.org/pdf/1306.0239.pdf.

Wen, Y. (2012). An improved discriminative common vectors and support vector machine based face recognition approach. Expert Syst Appl, 39(4), 4628–4632. doi:10.1016/j.eswa.2011.09.119.

Xue, D.-X., Zhang, R., Feng, H., & Wang, Y.-L. (2016). CNN-SVM for microvascular morphological type recognition with data augmentation. Journal of Medical and Biological Engineering, 36(6), pp.755–764. doi:10.1007/s40846-016-0182-4.

Alqahtani, A., & Wang, H. (2024). Advances in Deep Convolutional Networks for Visual Recognition: A Survey IEEE Access, 12, 12501-12520. doi: 10.1109/ACCESS.2024.3352104

Zhang, Y., Sun, Y., & Lin, X. (2023). Scalable DL for image recognition in big data environments Future Generation Computer Systems, 144, 85-97. https://doi.org/10.1016/j.future.2023.01.005

Doshi, K., Patel, D., & Shah, H. (2022). Lightweight Convolutional Neural Networks for Handwritten Digit Recognition Procedia Comput Sci, 198, 12-20. https://doi.org/10.1016/j.procs.2021.12.003

Rahman, M., & Hossain, S. (2023). Hyperparameter optimization of convolutional neural network models for handwritten digit classification. Applied Intelligence, 53(7), 8451-8463. doi: 10.1007/s10489-022-04125-6

Han, J., & Zhang, W. (2022). PyTorch-based deep learning frameworks for computer vision applications. Journal of Imaging, 8(9), 238. doi: 10.3390/jimaging8090238

Khan M, Akhtar N, Rehman S. 2021. Hybrid Convolutional Neural Network architectures for image classification: A case study on MNIST and CIFAR datasets International Journal of Machine Learning and Cybernetics, 12(11), 3313-3327. DOI: 10.1007/s13042-021-01342-7

Singh, A., Sharma, R., & Kumar, V. (2022). Robust lightweight convolutional neural networks for handwritten digit recognition under adversarial attacks. Neural Computing and Applications, 34(21), 19157-19172. DOI: 10.1007/s00521-022-07450-9