# Evaluation of multiple approaches to recognize and classify handwritten digits

Raúl Balanzá García[1, *]

[1] *Faculty of Information Technology, Czech Technical University in Prague*

(Dated: December 31, 2021)

The goal of this experiment is to try different approaches to recognize a series of handwritten digits and decide which is the most accurate one. The tests range from simple probability distributions to state-of-the-art methods such as CNNs. To accomplish this task, the MNIST dataset (and some of its variants) are used as the data source. This dataset, that consists of binary images of digits, is well-known in the AI community for being the standard for testing any image recognition system.

## I. GETTING STARTED

*Image recognition* is the ability of computer systems to identify and extract different kinds of information from images. This technologies are used to perform many visual tasks, such as labeling the content of images, guiding autonomous robots, or creating self-driving cars.

In this case, the goal is to classify handwritten digits, labelling them appropriately according to the digit that they represent. From all the different approaches that can be taken to solve the problem efficiently, here we focus in a few of them:

- Training and tuning **Convolutional Neural Networks** (CNN) to classify the data
- Using a prediction-based model with the **Zero-shot learning** (ZSL) technique
- Testing several **Nearest Neighbor** ($k$-NN) classifiers, that group the data using a similarity measure
- Checking whether the data adjusts properly to several **probability distributions**

All of the previously mentioned approaches were evaluated, and the results are presented in the next sections.

## II. THE DATASET

The chosen dataset for this project has been the **MNIST** (Modified National Institute of Standards and Technology) [1] database of evenly distributed handwritten digits from 0 to 9. It is a standard dataset used in computer vision and deep learning, that consists of a training set of 60,000 samples, and a test set of 10,000 samples.

The digits have been centered in a fixed-size image and size-normalized, as it can be seen in Fig. 1. The pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255.

Although this dataset is effectively solved, it can be used as the basis for developing, evaluating, and using several learning approaches for image classification. This includes estimating the performance of some models, and how to explore possible improvements.

There also exist several variants of this dataset with different types of data, such as *Fashion-MNIST* (with pictures of clothing), or *C-MNIST* (the original dataset adding randomly 10 different colors into their backgrounds and foregrounds).

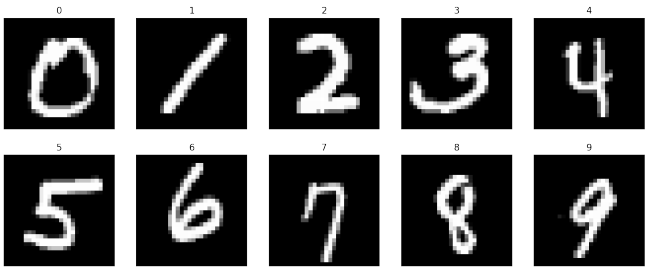


FIG. 1. Sample digits extracted from the MNIST dataset

## III. CONVOLUTIONAL NEURAL NETWORKS

The first evaluated approach was using *Convolutional Neural Networks*. A CNN is a Deep Learning algorithm that assigns importance (learnable weights and biases) to various aspects of the input samples and is able to differentiate them among each other. The pre-processing required using this approach is much lower when compared to other classification algorithms, and the architecture is analogous to that of the connectivity pattern of neurons in the human brain.

In this case, a base model was created, and then several tweaks were tested to check whether the performance of the classifier improved.

### A. Evaluation

For evaluating each model, *k-fold cross-validation* [2] was used, with $k = 5$. The value of $k$ was chosen to provide a baseline for both repeated evaluation and to optimize the running time. Each test set consisted of 20% of the training dataset ($\approx 12,000$ samples), close to the actual test set size. The training dataset was shuffled prior to being split, and the shuffling was performed each time, so that any evaluated model had the same

* balanrau@fit.cvut.cz

train and test datasets in each fold, making evaluation as accurate as possible. The baseline model was trained for **10 epochs** with a default batch size of **32 examples**. The test set for each fold was used to evaluate the model both during each epoch of the training run, to facilitate creating learning curves, and at the end of the run, so that the performance can be estimated.

Data about the resulting history from each run, as well as the classification accuracy of the fold was recorded. The classification accuracy scores collected during each fold can be summarized by calculating the mean and standard deviation. This provides an estimate of the average expected performance of the model trained on this dataset, with an estimate of the average variance in the mean.

### B. Initial model

First of all, the pixel values of grayscale images were normalized (i.e. rescaled them to the range $[0, 1]$). Then, the two main aspects of the model were defined: (i) the feature extraction front end, comprised of convolutional and pooling layers; and (ii) the classifier backend, that will make a prediction.

For the convolutional front-end, a single convolutional layer with a small filter size (3,3) and number of filters (32) followed by a max pooling layer was used.

Given that the problem is a multi-class classification task, it required an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each class. Therefore, it was decided to use a *softmax activation function* [3]. Between the feature extractor and the output layer, a dense layer with 100 nodes to interpret the features was added. All layers used the *ReLU activation function* [4] and the *He weight initialization scheme* [5], both recommended best practices.

The configuration for the stochastic gradient descent optimizer featured a learning rate of 0.01 and a momentum of 0.9. Finally, the categorical cross-entropy loss function, suitable for multi-class classification, was optimized.

After training this model taking into account the previously described parameters, all cases reached more than 98% accuracy, which can be considered a very positive result.

In this case, it can be seen in Fig. 2 that the model generally achieves a good fit, with train and test learning curves converging. There is no clear sign of over/underfitting.

Next, a summary of the model performance was calculated. In this case the model had an estimated skill of about 98.657% and a standard deviation of 0.133, which is acceptable. The distribution of accuracy scores can be observed in Fig. 3.

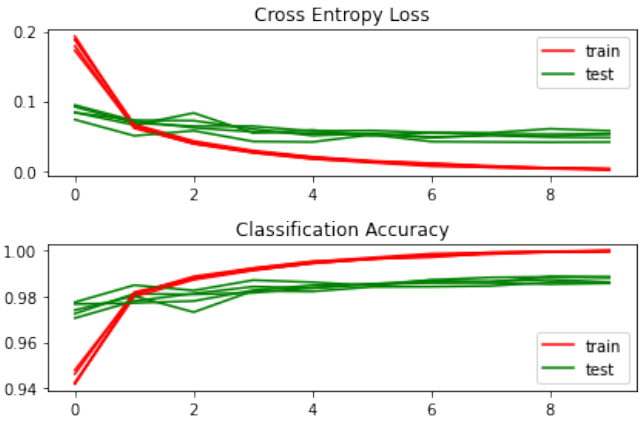


FIG. 2. Diagnostic plot of the initial CNN model, showing model performance on the train and test set during each fold

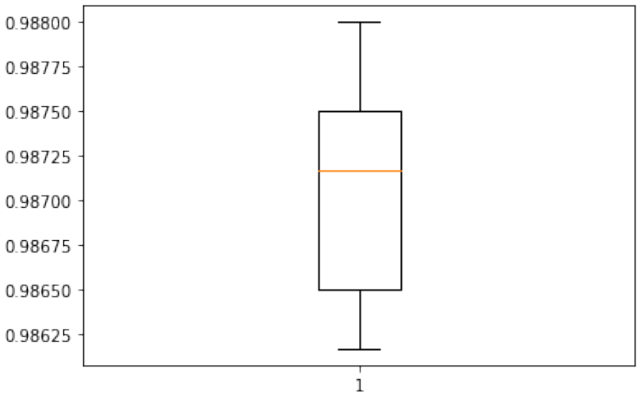

FIG. 3. Box and whisker plot summarizing the distribution of accuracy scores of the initial CNN model

### C. Model optimization

Once the initial model was designed and its results were evaluated, several aspects of the learning algorithm were explored to look for potential improvements.

The first tested approach, that could accelerate the learning and improve the model performance, was **batch normalization** [6], a method used to make artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling (i.e. *standardizing* the output). After implementing it into the initial model, a similar model performance is observed when comparing the results to the baseline. In this case, the speed of learning (improvement over epochs) does not appear to be different from the baseline model. The results suggest that this approach does not offer any benefit in this case.

As a second approach, the model configuration was modified by **increasing the depth of the feature extractor** part, following a *VGG-like pattern* of adding more convolutional and pooling layers with the same sized filter, while increasing the number of filters. In

TABLE I. CNN experiment results

| Model | Mean | Std. deviation |
|---|---|---|
| Initial | 98.657 | 0.133 |
| Batch normalization | 98.707 | 0.082 |
| Depth increasing | 98.845 | 0.151 |

this case, a double convolutional layer was added with 64 filters each, followed by another max pooling layer. Now, results show some improvement over the baseline, while keeping a good fit on the problem with no clear signs of overfitting. The estimated performance shows a small improvement, with a higher accuracy on average.

### D. Results

The results for the three tested models are shown in Table I. Finally, the performance of the depth-increased model was evaluated against the test dataset to determine how well it performs in practice. It was observed that the model achieved an accuracy of 99.310%.

Several additional approaches can be tested that could potentially improve the model. However, the current results are already promising and show that this Deep Learning technique is very suitable for recognizing the considered dataset.

## IV. ZERO-SHOT LEARNING

The second approach that was taken into account was *Zero-shot learning*. ZSL [7] is a problem setup in machine learning where, at test time, a learner observes samples from classes that were not observed during training, and needs to predict the class they belong to. These methods generally work by associating observed and non-observed classes through some form of auxiliary information, which encodes distinguishing properties of objects, as it can be seen in Fig. 4.

ZSL overcomes the problem of classical approaches of needing to add new classes and retrain the model if it has to recognize additional types of data. In this case, the objective is to recognize some digits that were not included in the training dataset.

### A. Model description

The chosen approach for this experiment consisted of 2 layers and was built upon creating relationships between features, attributes and classes with the help of a linear model.

1. The **first layer** defines the relationship between features and attributes using the weights in that layer.

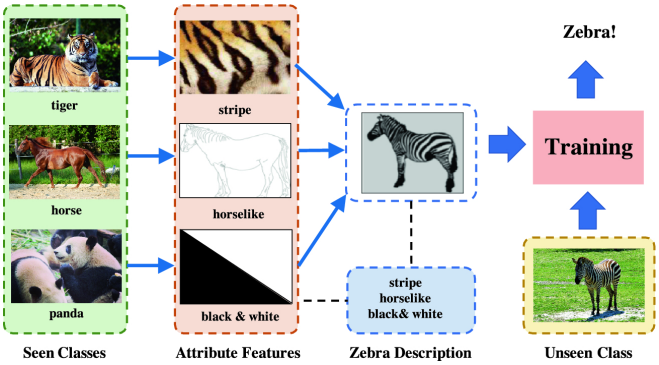


FIG. 4. Example of ZSL classification procedure using animals

2. The **second layer** deals with modelling the relationship between attributes and classes where the prescribed attribute signatures is fixed.

In the model, $z$ classes were chosen for **training**, and each of them had a signature composed of $a$ attributes. The signatures were represented in a boolean signature matrix $S$.

This helped in defining the soft link between attributes and classes. The samples available during training are denoted by $X \in R^{(d \times m)}$ where $d$ is the dimensionality of data and $m$ is the number of samples.

Ground truth labels for each training instance belonging to one of the $z$ classes are denoted by $Y \in \{-1, 1\}^{(m \times z)}$. Now, ground truths for each sample could be obtained from $Y$ and $S$.

If a linear predictor in a classical approach was used, then it would be required to optimize the loss function for output $Y$ and dot product of transpose of $X$ and $W$, where $W$ contains the parameters to be learned.

At the **inference** stage the objective was to distinguish between a new set of $z'$ classes. To do so, the available information was their attributes signatures, $S' \in [0, 1]^{(a \times z')}$.

When designing a model, it is desired that the Euclidean norm of the representation of any (training) attribute signature, $s \in [0, 1]^a$, must be controlled so that ideally the representation of all signatures on the feature space have a similar Euclidean norm. This allows fair comparisons between signatures, and prevents problems that stem from highly unbalanced training sets.

Then, given a new instance, $x$, the prediction is given by $argmax(transpose(X) \cdot V \cdot Si')$ where $V$ is given by $W = V \cdot transpose(S), V \in R^{(d \times a)}$.

### B. Training the model

To apply the previous procedure to the dataset under study, first *Label Encoder* was used to encode target digits into values between 0 and $(C - 1)$, where $C$ is the total number of classes. Then, only $z$ digits were chosen for training, in this case those were $\{0, 1, 2, 7, 8, 9\}$.

TABLE II. ZSL experiment results

| Logistic Regression model | Training accuracy | Inference accuracy |
|---|---|---|
| L2 reg. - Solver: *saga* | 72.423 | 46.537 |
| L2 reg. - Solver: *lbfgs* | 63.925 | 48.061 |
| Elastic-net reg. - Solver: *saga* | 72.056 | 46.814 |

After encoding the target values, the weight matrix was created using logistic regression and attributes were created with unsupervised learning.

The signature matrix was built with 2 components of each *Principal Component Analysis* (PCA) and *Locally Linear Embedding* (LLE).

PCA is a technique for feature extraction that combines input variables in such a way that the least important variables are ignored while keeping the important information. This method also avoids overfitting.

LLE is a lower-dimensional projection of the data which preserves distances within local neighborhoods. It can be thought of as a series of local PCA which are globally compared to find the best non-linear embedding.

Finally, new weight matrix was calculated from this signature matrix $S$ and matrix $V$. From this, new predictions can be made about the unseen digits.

### C. Results

After testing several *Logistic Regression* models that could potentially fit the training sample data, the results shown in Table II were obtained, reaching a training accuracy of up to 72.5% in the training phase and of 48.06% in the inference phase.

Although these results might not seem remarkable when compared to the previous CNN approach (section III), they almost twice as good as a random prediction for unseen digits in the best case (with a 25% of accuracy considering 4 unseen digits), proving the efficiency of this predictive approach.

However, this experiment should only be considered as a reference to be used in datasets with a potentially increasing number of classes, as the MNIST dataset has a very well defined amount of classes that could be easily considered in training time.

## V. *K*-NEAREST NEIGHBORS

The next approach to be tested was a simpler one than those of the previous experiments, and was based on the *k-Nearest Neighbors* algorithm. $k$-NN [8] is a non-parametric classification method used for classification and regression. The input consists of the $k$ closest training examples in a data set. In classification, the output is a class membership. An object is classified to the most common class among its $k$ nearest neighbors, as in Fig. 5 ($k$ is a positive integer, typically small).

The neighbors are taken from a set of objects for which the class is known: this can be thought of as the training set for an algorithm of supervised learning. $k$-NN is sensitive to the local structure of the data.

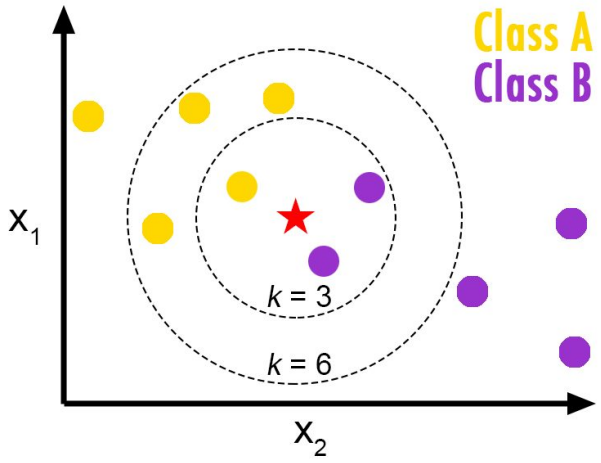


FIG. 5. k-NN classification example: with $k = 3$, the sample $\star$ would be classified into class B; with $k = 6$, it would be classified into class A

### A. Experiment description

In this case, the **experimentation** phase consisted of dividing the dataset into a training set, composed of a random set of 90% of the sample data; and a testing set, composed of the remaining 10%.

Then, using the ***PCA*** dimensionality reduction technique (defined in section IV B), the training dataset was reduced from 784 (28x28 represented as a 1D vector) dimensions to different values of $d$ dimensions, and its training accuracy was measured using a $k$-NN classifier with different values of $k$ and the Euclidean (L2) distance.

Finally, in the **evaluation** phase the $d$ value that achieved the best accuracy (i.e. the one that kept most of the important information while removing noisy data), together with the best $k$ value, were chosen and then evaluated against the whole MNIST dataset.

### B. Results

The values of accuracy obtained in the *experimentation* phase after reducing the data to different dimensions and comparing with various amounts of neighbors are summarized in Table III. As it can be observed, the best result was to reduce the data to 100 dimensions and to use a 1-NN classifier (i.e. the nearest neighbor).

In the *testing* phase with the whole MNIST dataset, the obtained accuracy was of 97.16%.

This result is still not as good as the initial CNN approach (section III). However, the simplicity of training this model compared with CNNs, and considering that

TABLE III. $k$-NN experiment, training results

| **Dimensions** ($d$) | **Neighbors** ($k$) | **Accuracy** (%) |
|---|---|---|
| 5 | 1 | 69.517 |
| 20 | 1 | 96.900 |
| 100 | 1 | 97.600 |
| 200 | 1 | 97.250 |
| 5 | 3 | 72.133 |
| 20 | 3 | 97.017 |
| 100 | 3 | 97.517 |
| 200 | 3 | 97.317 |
| 5 | 5 | 73.750 |
| 20 | 5 | 96.983 |
| 100 | 5 | 97.300 |
| 200 | 5 | 97.066 |

the difference is of only 2.15% in accuracy, makes this approach interesting to explore.

Potential optimizations could be to explore different distance measures (such as L1, L3 or Mahalanobis distance [9], among others) or using Wilson's editing algorithm [10] to preprocess the data, removing noisy samples.

## VI. PROBABILITY DISTRIBUTIONS

The final approach to be tested was to assume that the data followed different probability distributions, and to try estimating its parameters. More precisely, the tested distributions were multidimensional *Bernoulli*, *Multinomial*, and *Gaussian*.

Some smoothing techniques were applied on the data to avoid overfitting and, because each of these probability distributions are suitable for different kinds of data, it was required to preprocess the samples in some cases.

For each of the following experiments, in the **experimentation** phase the dataset was again divided into a training set, composed of a random set of 90% of the sample data; and a testing set, composed of the remaining 10%. Then, in the **evaluation** phase, the best obtained values were chosen and then evaluated against the whole MNIST dataset.

### A. Bernoulli distribution

The Bernoulli distribution is a discrete (binary) probability distribution whose parameters are the probability of 1 (success), $p$; and the probability of 0 (failure), $q = (1 - p)$.

It was then assumed that each dimension of MNIST followed this distribution, so data needed to be *converted* from positive natural values in the range $[0, 255]$ to binary values. To accomplish that, a binarization threshold $b = 0.5$ was set, so that values $> t * 255$ were considered 1, and 0 otherwise.

Then, the *simple truncation* ($\tilde{p}_{cd}$) smoothing technique was applied:

$$\tilde{p}_{cd} = \begin{cases} \epsilon & if\ \hat{p}_{cd} < \epsilon \\ 1 - \epsilon & if\ \hat{p}_{cd} > 1 - \epsilon \\ \hat{p}_{cd} & otherwise \end{cases} \qquad 0 \leq \epsilon \leq 0.5$$

where $\hat{p}_{cd}$ is the probability of the $d$-th dimension of the $c$-th class weight vector. Finally, several experiments were performed using different values for $\epsilon$.

The obtained values of the experimentation phase can be found in Figure 6. We can observe that the best result was obtained with $\epsilon = 1 * 10^{-25}$, which led to a training accuracy of 83.47%.

In the testing phase, an accuracy of 84.38% was achieved.

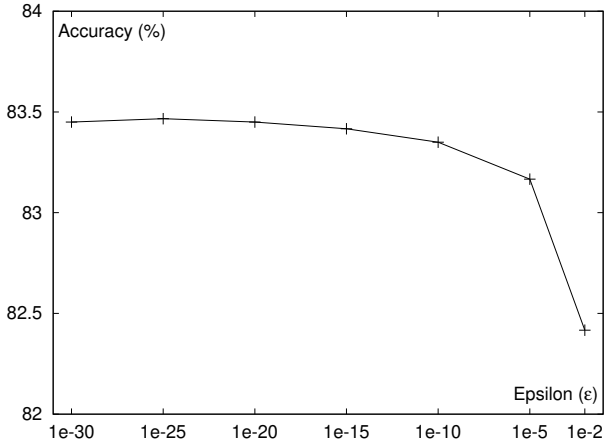


FIG. 6. Bernoulli training results with different $\epsilon$ values

### B. Multinomial distribution

The Multinomial distribution is a generalization of the binomial distribution that gives the probability of any particular combination of numbers of successes for various categories. For example, it models the probability of counts for each side of a $k$-sided die rolled $n$ times. For n independent trials each of which leads to a success for exactly one of k categories, with each category having a given fixed success probability, this distribution gives the probability of any particular combination of numbers of successes for the various categories.

It was then assumed that each sample digit of MNIST followed this distribution. In this case, the distribution already works with positive integer values, so data did not need to be preprocessed.

Then, the *Laplace* ($\tilde{p}_{cd}$) smoothing technique was applied:

$$\tilde{p}_{cd} = \frac{\hat{p}_{cd} + \epsilon}{\sum_d (\hat{p}_{cd} + \epsilon)} \qquad \epsilon > 0$$

where $\hat{p}_{cd}$ is defined as in the Bernoulli distribution (VI A). Finally, several experiments were performed using different values for $\epsilon$.

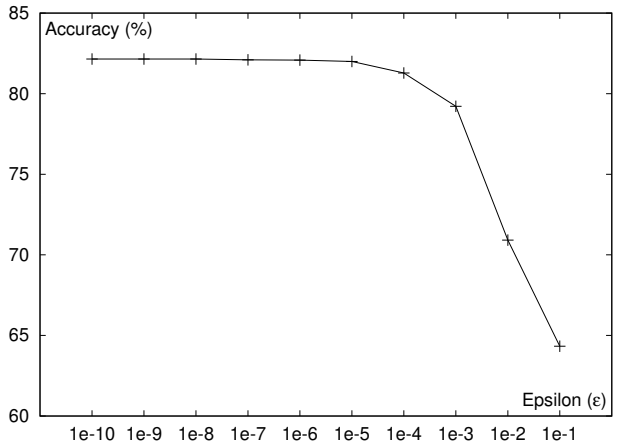


FIG. 7. Multinomial training results with different $\epsilon$ values

The obtained values of the experimentation phase can be found in Figure 7. We can observe that the best result was obtained with $1 * 10^{-10} \leq \epsilon \leq 1 * 10^{-8}$, which led to a training accuracy of 82.15%.

In the testing phase, an accuracy of 83.67% was achieved. This results are very similar to the ones obtained previously with the Bernoulli distribution (about 1% difference), but in this case the parameters are more complex (positive integers vs. binary values), so for now the best option would be to choose the previous distribution.

### C. Gaussian distribution

The Gaussian distribution (a.k.a normal distribution) is a bell-shaped curve, and it is assumed that during any measurement values will follow a normal distribution with an equal number of measurements above and below the mean value.

For this last experiment, it was also assumed that each sample digit of MNIST followed this distribution. In this case, preprocessing was not needed as the distribution also works already with positive real values.

Then, the *flat smoothing* ($\tilde{\Sigma}_c$) technique was applied:

$$\tilde{\Sigma}_c = \alpha * \hat{\Sigma}_c + (1 - \alpha) * I$$

where $\hat{\Sigma}_c$ is the covariance matrix for each class $c$, and $I$ is the identity matrix. Finally, several experiments were performed using different values for $\alpha$.

The obtained values of the experimentation phase can be found in Figure 8. We can observe that the best result was obtained with $\alpha = 1 * 10^{-4}$, which led to a training accuracy of 95.73%.

In the testing phase, an accuracy of 95.82% was achieved. This results are the best out of the three tested distributions, with an improvement of over 10%, which seems reasonable considering the amount of parameters that the distribution has.

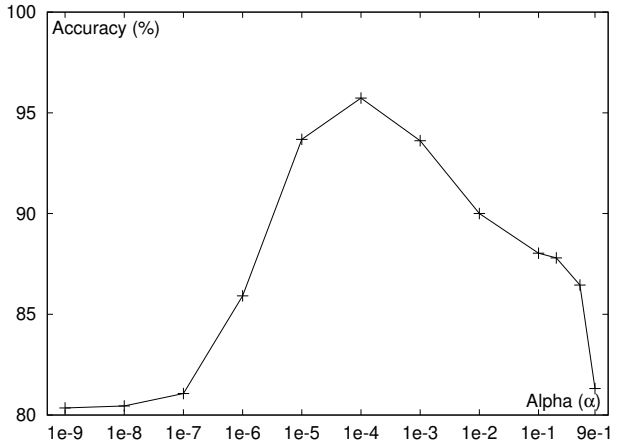


FIG. 8. Gaussian training results with different $\alpha$ values

TABLE IV. Summary of results (best accuracies)

| **Approach** | **Training** | **Testing** |
|---|---|---|
| Convolutional Neural Network | 98.845 | 99.310 |
| Zero-shot learning[a] | 72.423 | 48.061 |
| $k$-Nearest Neighbors | 97.600 | 97.160 |
| Bernoulli distribution | 83.470 | 84.380 |
| Multinomial distribution | 82.150 | 83.670 |
| Gaussian distribution | 95.730 | 95.820 |

[a] For ZSL, the inference phase was considered as the testing phase.

## VII. FINAL CONCLUSIONS

After running all the experiments and gathering relevant data, a summary of the best obtained values of accuracy in both the training and testing phases of all experiments is observed in Table IV.

As it can be seen, the *Convolutional Neural Network* is the best performing one, followed closely by the *k-Nearest Neighbors approach*. Considering the difficulty of training both approaches, and due to the small difference in both results, **the most appropriate choice in this case would be to go for the *k*-NN model**.

Moreover, when analyzing the results for the probability distributions, we can find promising results for the Gaussian case, with an accuracy that is close to that of the two best approaches. However, for the considered dataset the samples do not fit as well into the Bernoulli and Multinomial distributions, although the results are in acceptable ranges.

Finally, the *Zero-shot learning* approach obtained the worst results. However, that model is, in fact, characterized for having the least amount of information input

of all the considered approaches. It was concluded that altough in MNIST the amount of classes is very well-defined, the model obtained acceptable results that would be remarkable for an expandable class dataset with more dynamic information.


[1] Y. LeCun, C. Cortes, and C. Burges, Mnist handwritten digit database, ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010).

[2] T. Fushiki, Estimation of prediction error by using k-fold cross-validation, Statistics and Computing **21**, 137 (2011).

[3] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, Activation functions: Comparison of trends in practice and research for deep learning, arXiv preprint arXiv:1811.03378 (2018).

[4] G. Lin and W. Shen, Research on convolutional neural network based on improved relu piecewise activation function, Procedia computer science **131**, 977 (2018).

[5] S. K. Kumar, On weight initialization in deep neural networks, CoRR **abs/1704.08863** (2017), 1704.08863.

[6] J. Bjorck, C. Gomes, B. Selman, and K. Q. Weinberger, Understanding batch normalization, arXiv preprint arXiv:1806.02375 (2018).

[7] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly, CoRR **abs/1707.00600** (2017), 1707.00600.

[8] N. S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, The American Statistician **46**, 175 (1992).

[9] P. C. Mahalanobis, On the generalized distance in statistics (National Institute of Science of India, 1936).

[10] D. L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man, and Cybernetics **SMC-2**, 408 (1972).