
Illinois Institute of Technology

Department of Computer Science

CS-512

Computer vision

# IIT Final Project - Report

## Image classification using forward forward neural networks

*Author:*
Ishrat Jahan Ananya
Tristan Leduc

*Student Number:*
A20516799
A20517874

April 18, 2023

# Team contribution

**Ishrat Jahan**: Experimenting with different hyperparameters, implementing parts of 'label formatting' experiments, writing report

**Tristan**: Experimenting with different loss functions, implementing parts of 'label formatting' experiments, creating the presentations

# Abstract

Backpropagation has been the reigning Machine Learning paradigm for several years and has been widely used in various applications. Although the approach is effective in many fields, such as speech recognition, natural language processing, and even game playing, it has one big drawback.Limitation of back propagation is that it requires perfect knowledge of the computation performed in the forward pass1 in order to compute the correct derivatives. This limitation can't be entirely solved due to the core nature of the algorithm. Geoffrey Hinton, who had popularized the use of backpropagation, proposed a new approach to overcome the shortcomings of the traditional back-propagation method in his paper titled 'The Forward-Forward Algorithm: Some Preliminary Investigations'. The Forward Forward algorithm replaces the backward pass in the backpropagation method with two forward passes, offering a more computationally efficient alternative. This report explores the Forward Forward algorithm in greater detail, discussing its strengths and limitations and comparing it to the traditional backpropagation method. We also explore the potential applications of the Forward Forward algorithm in various fields and examine its potential impact on the machine learning industry.

## 1 Proposed solution

The baseline implementation, as published by Keras, is a more lightweight version of the conditions described in the paper. Nonetheless, it yields a test error rate of 2.36%, which means 97.64% accuracy. Our implementation process can be divided into three sub-tasks, which are detailed below:

1. Impact of depth and breadth of the model: Like any other neural network architecture, the first intuition is to explore whether tuning the hyper parameters or changing the size of the architecture makes any notable improvement in terms of efficiency, space consumption, or time cost. Hinton's paper discussed using a very small MLP for the task and argued that it is enough to yield results that are on par with the back-propagation counterpart in traditional neural networks. We conducted four experiments with varying numbers of layers and neurons per layer, paired with different combinations of hyper parameters, to check what effect it has on the error rate.

2. Impact of different loss functions: Due to the limited time and narrow scope of our

project, we experimented with only one type of loss function, which is Margin Loss. Margin loss is better at distinguishing class margins and is more robust to outliers. While training, the rest of the parameters are kept consistent with the baseline model for fair comparison. Let y be the true label of an input data point x, and let f(x) be the predicted score for the true label y. Let s be a margin hyperparameter that controls the degree of confidence that the classifier should have in its predictions. Then the margin loss L is defined as:

$$L(x, y) = max(0, s - f(x) + f(x, y'))$$

The margin loss penalizes the classifier when it assigns low confidence to correct predictions or high confidence to incorrect predictions.

3. Impact of label formatting: The original implementation overlays the one-hot encoded labels on top of the images before sending them through the forward pass as it is a supervised task. However, the issue with this is that it limits the datasets we can use. Ideal images, like the MNIST dataset, must have a black or white border where the label can be overlaid without causing significant disturbance to the underlying patterns of the images. This means the model doesn't perform well on images with variable backgrounds, such as in the CIFAR-10 dataset. Hence, we propose to add a new dimension to the input tensor that contains the label encoding. This is then passed to the network for training.

# 2 Implementation details

This section describes the experiments in more details and explains the changes made to the architecture.

## 2.1 Data Pre-processing

**Dataset:** The MNIST dataset is a widely-used benchmark dataset for image classification tasks, particularly for handwritten digit recognition. It consists of a collection of grayscale images of size 28x28 pixels, each representing a handwritten digit from 0 to 9. The dataset has a total of 60,000 training images and 10,000 test images, with an equal distribution of 6,000 images per class in the training set and 1,000 images per class in the test set.

**Positive and Negative Images** The dataset provided to the model for training consists of two types of images: real and fake. Real images are positive examples, meaning that they are the original images with correct labels overlaid on them. In contrast, fake images are generated in the same way, but with incorrect labels overlaid on them.

During the positive pass, the model uses real data to update the weights of the hidden layers in order to improve the quality of the model. Specifically, the model tries to minimize the difference between its predicted labels and the correct labels. Conversely, during the negative pass, the model uses "negative data" (i.e., data with incorrect labels) to update the weights of the hidden layers to decrease the quality of the model. This has the effect of penalizing the model for making incorrect predictions, and helps it learn to

distinguish between correct and incorrect classifications.It is important to note that this training process is iterates for a few epochs per layer in every model epoch run.

**Overlaying Label on Image** There are two methods used for this subprocess which are explained with code below:

The overlay y on x method modifies the input image by overlaying it with a one-hot encoding of the label, which ensures that the maximum value in the image corresponds to the label. This technique can be useful for training neural networks on image classification tasks, as it encourages the network to focus on the most salient features of the image that are relevant to the label. The function takes a tuple data as input, which contains an image sample and its corresponding label y sample. The method returns the modified image X sample and the original label y sample.

We implemented another method "add one hot label" that modifies the input image by adding a one-hot encoded tensor of the label as an additional channel instead of overlaying on top of the image. This removes the limitation of poor training results on images with varying backgrounds.This can be useful for training neural networks on image classification tasks, as it provides the network with additional information about the label that can help it make more accurate predictions.

```python
def add_one_hot_label(self, data):
    # Define some constants
    num_classes = 10
    input_height = 28
    input_width = 28

    # Unpack the input data into X_sample and y_sample
    X_sample, y_sample = data

    # Convert the label to one-hot encoded tensor
    label = tf.one_hot(y_sample, num_classes)

    # Create the label layer tensor with the one-hot encoded label
    label_layer = tf.tensor_scatter_nd_update(tf.zeros((input_height, input_width)), [[0, i] for i in range(num_classes)],

    # Cast the X_sample tensor to float32 and reshape it to the desired size
    X_sample_value = tf.cast(X_sample, dtype=tf.float32)
    X_sample_value = tf.reshape(X_sample_value, (input_height, input_width, 1))

    # Concatenate the label layer tensor with the X_sample tensor
    final_array = tf.concat([X_sample_value, tf.expand_dims(label_layer, -1)], -1)

    # Reshape the final image variable into a flattened tensor and cast it to float64
    new_X_sample = tf.cast(tf.reshape(final_array, (-1,)), tf.float64)

    # Return the new flattened image tensor and y_sample
    return new_X_sample, y_sample
```

Figure 1: One hot label

## 2.2 Baseline Architecture

The baseline architecture used in this project is a simple shallow Multilayer Perceptron (MLP) composed of 2 linear layers followed by a sigmoid function, with 500 neurons

assigned to each layer. To optimize the model, we employed the Adam optimizer with a learning rate of 0.03. In order to train the model, it was run for a total of 250 epochs, with 50 internal epochs per layer.

To encode the label on top of the images, we passed the dataset through the "overlay y on x" function. In terms of the loss per layer, we calculated it as the sum of the squared neural activities and the negative sum of the squared activities.

The objective of this experiment is to determine whether a positive image yields a goodness score (which is the sum of the squared activities of neurons) above a specified threshold, while a negative image yields a goodness score below a given threshold. We set the threshold to 1.5 to ensure that the results are accurate and reliable.

## 2.3 Impact of Depth and Breadth

Increasing the width of the hidden layers by adding more units per layer can significantly improve the performance of the model. This is because wider layers promote more variability and non-linearity in the model, allowing it to capture richer hidden features. In contrast, increasing the depth of the network did not show any significant improvement in the model's performance. This suggests that the learned features did not generalize well through the layers.The four experiments conducted and their results are discussed in section 4.

## 2.4 Impact of Loss

The goal of this experiment is to investigate how the goodness function affects performance. As a reminder, the goodness function for a layer is a function applied to the sum of the squares of the activities of the rectified linear neurons in that layer. The aim of the learning process is to make the goodness be well above some threshold value for real data and well below that value for negative data.The equation used for positive data is:

$p(positive) = \sigma(\sum_j {y_j}^2 - \theta)$

where sigma is a logistic function and y is the activation of the j unit in the layer.

In this experiment, we tested the use of margin loss as an alternative to the sum of squared goodness function to see if it could improve performance. The use of margin loss did not improve the model's performance and instead resulted in a slight decrease in accuracy. It's possible that margin loss was not an effective replacement for the sum of squared goodness function in this task. The conclusion we can draw is that the sum of squared goodness function appears to be the most suitable choice for this model as it resulted in the highest accuracy in our experiments. However, further investigation may be necessary to explore alternative goodness functions that could potentially improve the model's performance.

## 2.5 Impact of label formatting

In this section, we aim to address one of the limitations of the original model, which is related to labels. The model was trained using supervised learning, achieved by embedding

labels within the images. This approach involves overlaying labels on the edge of the images, which was relevant for the MNIST dataset that contains images with a black background. However, this method requires a uniform background to work effectively, which limits the range of datasets on which the model can be trained. For example, the CIFAR-10 dataset includes images with highly variable backgrounds, making it difficult to place the labels on the border without interfering with other image features.

To overcome this limitation, we propose embedding the labels in a separate channel of the images as discussed in section 3.1.

The addition of a separate label layer did not significantly improve the model's performance on the MNIST dataset, but it did not result in a decrease in accuracy either. The lack of improvement could be due to the fact that the MNIST dataset already has a uniform background, making it easy to overlay labels on the edge of the images. Embedding labels in a separate layer could be a useful solution for datasets with variable backgrounds where overlaying labels on the edge of the images is not feasible. However, for datasets like MNIST with uniform backgrounds, this approach does not seem to provide a significant advantage over the original method.

# 3 Results and discussions

This section entails the perfomance of the model for every experiment with graphs and evaluation metrics. Furthermore it offers an explanation for the cause and future potential of the project.

## 3.1 Impact of Depth and Breadth

### 3.1.1 Performance of a deeper network that the baseline (4 hidden layers)

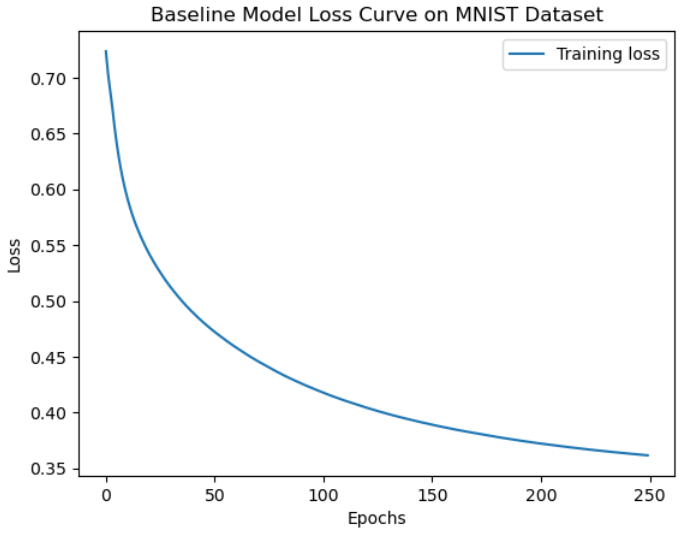


Figure 2: Training loss across epochs

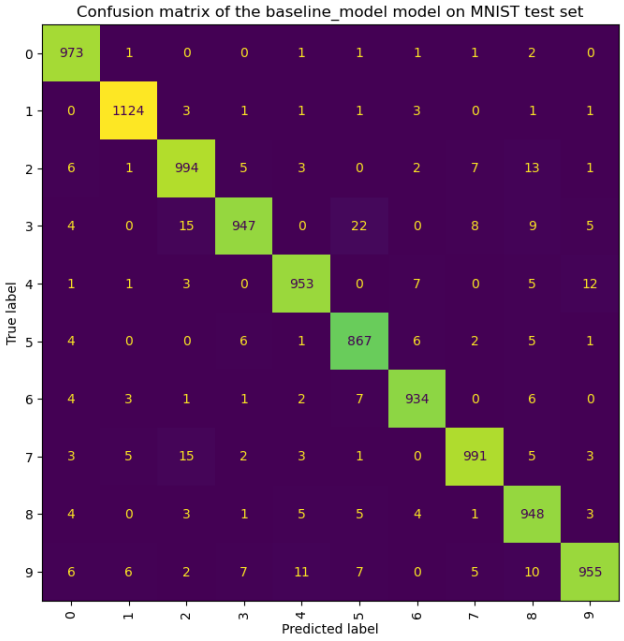


Figure 3: Confusion matrix of the model

**Analysis**: Overall, increasing the depth of the network did not improve the performance of the model, indicating that the learned features did not generalize well through the layers. However, increasing the width of the hidden layers by adding more units per layer can improve the performance of the model by promoting more variability/non-linearity, making the model capture richer hidden features. Regularization techniques such as dropout can also help the model generalize well on unseen data by controlling the chances of overfitting.

**Conclusions drawn**: In conclusion, we found that wider hidden layers and dropout can help improve the performance of the baseline model. However, increasing the depth of the network did not improve the performance of the model, indicating that the learned features did not generalize well through the layers.

### 3.1.2 Performance of a wider network that the baseline (1200 units in each layer), with dropout)

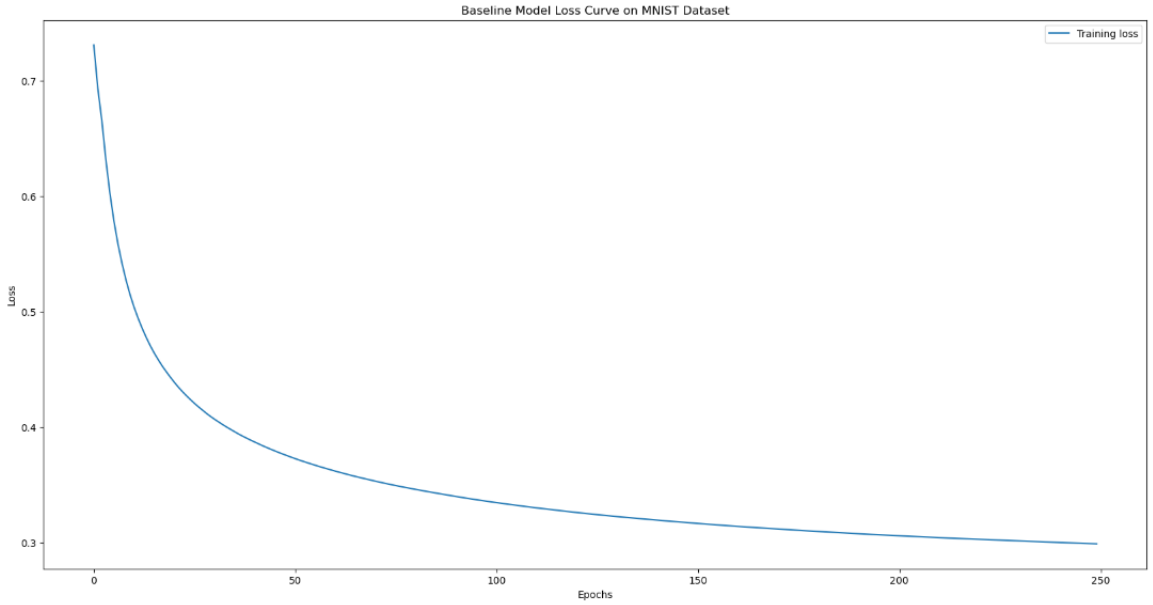


Figure 4: Training loss across epochs

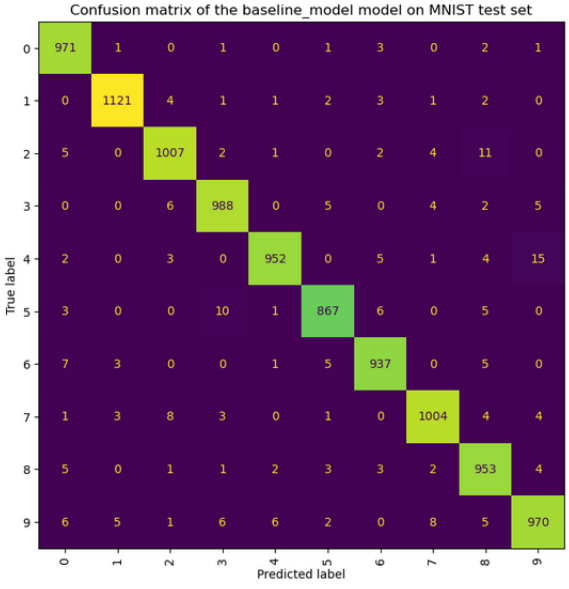


Figure 5: Confusion matrix of the model

### 3.1.3 Performance of a narrower network that the baseline (200 units in each layer)

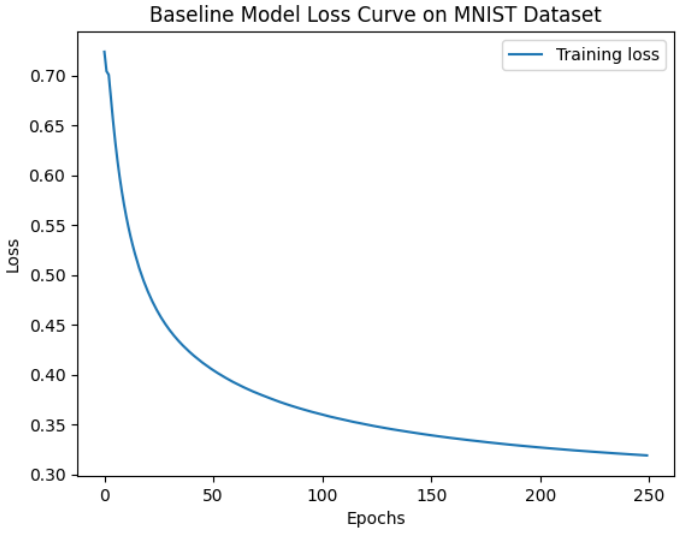


Figure 6: Training loss across epochs

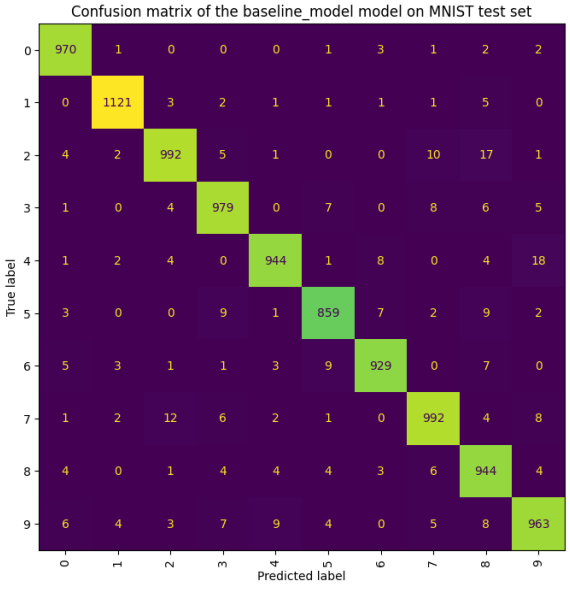


Figure 7: Confusion matrix of the model

## 3.2 Impact of Loss/goodness function on the performance

### 3.2.1 Implementation of the model with margin loss

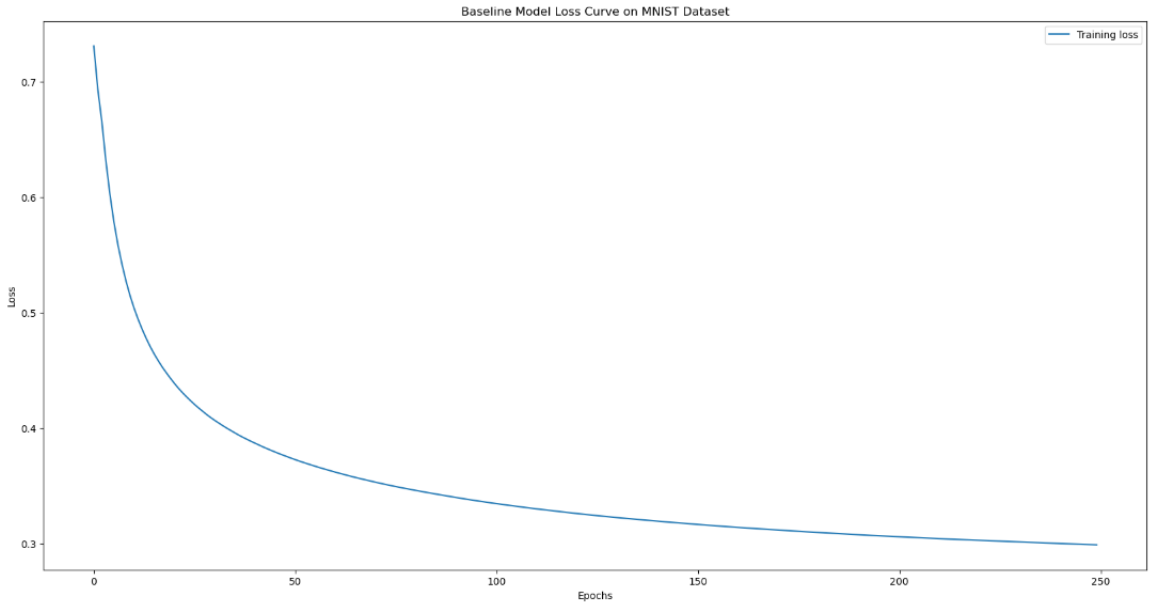


Figure 8: Training loss across epochs

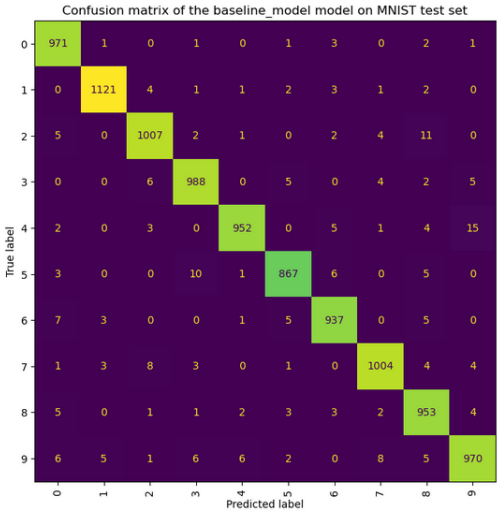


Figure 9: Confusion matrix of the model

**Analysis**: The use of the margin loss did not improve the model's performance, and instead resulted in a slight decrease in accuracy. It's possible that the margin loss was not an effective replacement for the sum of squared goodness function in this task.

**Conclusions drawn**: The sum of squared goodness function appears to be a suitable choice for this model, as it resulted in the highest accuracy in our experiments. However,

further investigation may be necessary to explore alternative goodness functions that could potentially improve the model's performance.

## 3.3 Impact of label formatting

### 3.3.1 Implementation of the model with the new way to encode labels

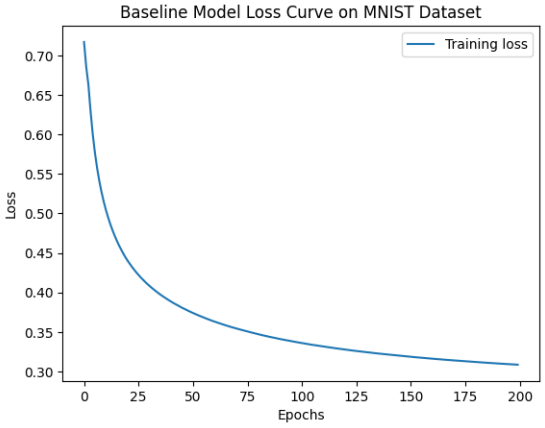


Figure 10: Training loss across epochs

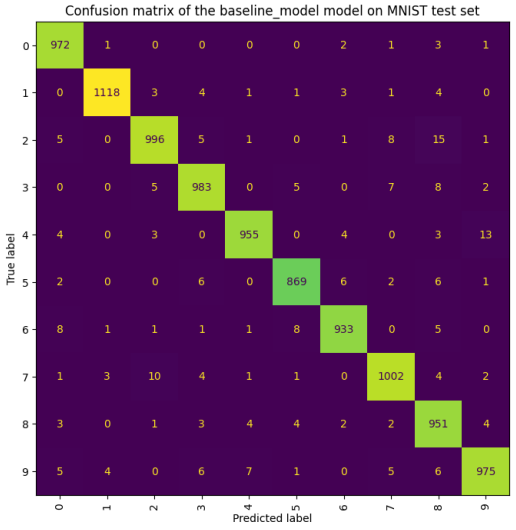


Figure 11: Confusion matrix of the model

**Analysis**: The addition of a separate label layer did not improve the model's performance on the MNIST dataset, but it did not result in a decrease in accuracy either. The lack of improvement could be due to the fact that the MNIST dataset already has a uniform background, making it easy to overlay labels on the edge of the images.

**Conclusions drawn**: The approach of embedding labels in a separate layer could be a useful solution for datasets with variable backgrounds where overlaying labels on the edge

of the images is not feasible. However, for datasets like MNIST where the backgrounds are uniform, this approach does not seem to provide a significant advantage over the original method.

## 3.4 Implementation of the concept of shared weights

### 3.4.1 Implementation of a convolution layer type

**Analysis**: The implementation of convolution layers was unsuccessful, as the use of shared weights did not prove to be feasible for this particular task.

**Conclusions drawn**: While convolution layers are commonly used in neural networks and have been successful in improving performance in many tasks, it appears that they may not be suitable for this particular model. It may be necessary to explore other types of layers or modifications to the existing layers to improve performance further.

# References

[1] Keras base implementation of the forward forward algorithm link.

[2] Link to the MNIST dataset page link.