Digit recognition from 3D coordinate data

Ilmari Vahteristo, Pauli Anttonen & Emma Hirvonen

December 19, 2023

Contents

1 Introduction

1.1 Docs

Ilmari Vahteristo (0545212), Pauli Anttonen (0563870), Emma Hirvonen (0565946).

Our Code can be found at `https://github.com/ilmari99/DigitRecognition3D`.

To test our model you need MATLAB version >= 2023a.

Our function 'digit_classify.m' takes in a matrix X with size (222,3), and optionally a model loaded with "importNetworkFromTensorFlow('model_pb')", if the model is not provided, the model is loaded from 'model_pb'. Providing the model is faster.

If something does not work, you can run 'digit_classify_test.m' but you need to change the 'DATA_ROOT' argument to contain your test data.

As for group roles: Everyone did an equal amount of writing to the report, and everyone was part of planning and executing our modeling and preprocessing plan.

1.2 Problem description

This project focuses on the development of a pattern recognition system which task is to identify hand-drawn digits 0, 1, ..., 9, which are created through free-hand strokes in the air using the index finger. The 3-D location data of the index finger, is collected vith a LeapMotion sensor. Given dataset includes thousand observations/measurements of the hand drawn digits and related ground truth labels. Dataset has balanced distribution over classes, which helps to get reliable results and to reduce bias towards specific classes. For the recognition of the hand-drawn digits, we use a convolutional neural network (CNN). Selected model comprises a series of convolutional layers interspersed with pooling operations.

1.3 Data visualization

To visualize and get a grasp of the data, we plotted two random samples from the data in the original coordinate data (Figure 1). We also plotted the data in 2D, since characters are typically represented in 2 dimensions. The idea in projecting each coordinate in a sequence to two principal components is that we only need the coordinates in the 2 dimensions that maintain the greatest variance in the data, i.e. the character. These projections can be seen in Figure 2. Projecting the data to 2 PC's also reduces the amount of data in a padded coordinate sequence to $\frac{2}{3}$ of the original which speeds-up computations and possibly improves convergence.
[1]

[1] We did not do the modelling with 2D projections of the data, we only illustrate this.

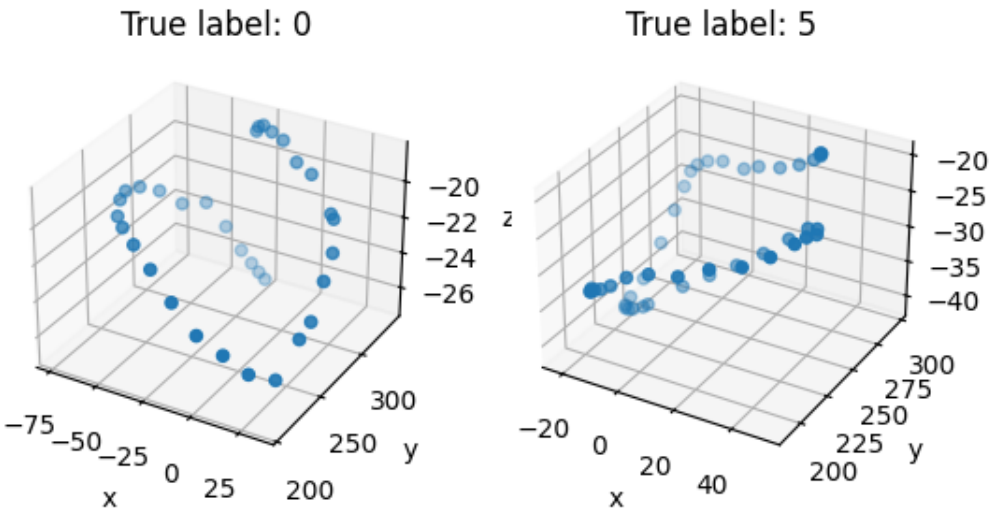

Figure 1. The data consist of sequences of 3D coordinates that describe where the finger of a person was moving through space. The data also shows the finger entering the drawing area, as well as the drawing of the digit.

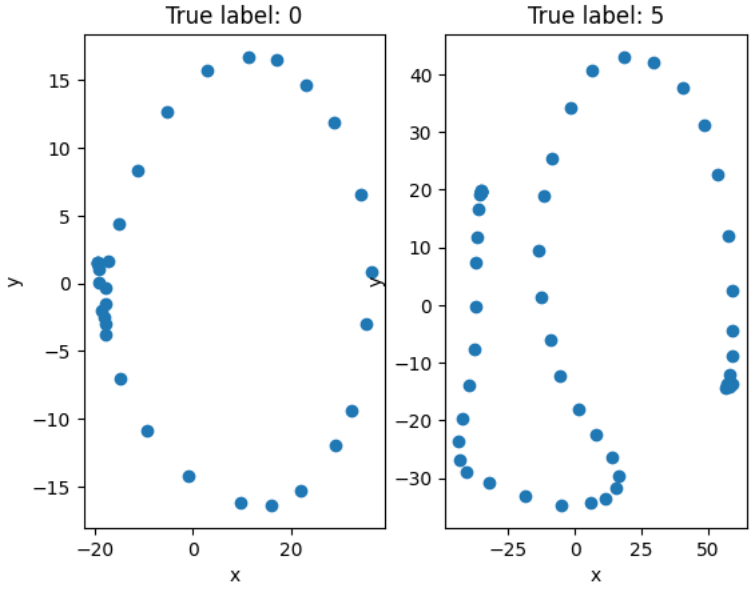

Figure 2. Two 3D coordinate sequences where each 3D coordinate is projected to two PC's calculated from the sequence.

2 Theory

2.1 Multilayer perceptron

Multilayer perceptron's (MLPs) are artificial neural network models, which can be used as universal function estimators. MLPs are composed with large number of simple computational units called neurons or perceptrons. General structure of perceptron is as follows, perceptron can receive n numerical inputs and each of these features are linked to weight. Input features are fed to input function u, which calculates so called net activation i.e. weighted sum of features, presented in formula 1

$$u_k = \sum_{i=1}^{n} w_{ik} x_i \tag{1}$$

where x_i is feature and w_i is the weight. Results of function u are passed to a nonlinear activation function 2, f for the neuron output.

$$y = f(u(x)) = \begin{cases} 1, & \text{if u(x) > } \theta \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

As can be seen the activation function is step function. [1] [2]

MLP's operates by combining several of these perceptrons, whit at least three layers. One of them is input layer which only distributes input features for the following hidden layer. MLP need to have at least one hidden layer, where the above mentioned perceptron calculations take place. MLP can also contain multiple hidden layers. And one output layer which get inputs from the last hidden layer perceptron outputs. [1] [2]

In multilayer networks as activation function is often used ReLU function 3 which gives as output a direct input if it is positive and zero if it is negative.

$$f_{relu} = max(0, x) \tag{3}$$

ReLU function is used because it helps to reduce gradient loss, which is a problem when using a linear step function like 2. For that reason ReLU helps effective learning in multilayer networks. [3]

Important part of learning is adjusting perceptrons' weights in such a way that training data error is minimized. This is usually done by using backpropagation algorithm. Backpropganation works by iteratively updating the weights starting from the output layer working backwards through each layer. Updating the weights is based on gradient decent optimization algorithm which uses gradient of the loss function to update weights in way that it reduces the error. Thus the backpropagation algorithm aims to minimise mean squared error in learning. [1] [2]

2.2 Convolutional layer

Expanding the the theory of MLP by using convolutional layers can help extract information from adjacent data points and further improve accuracy when dealing with continous curves. Convolutional layers use efficient convolutional calculations to transform data with trainable kernel wieghts. This structure is called convolutional neural network

(CNN) The definition of single one dimensional CNN output y is shown in Formulas 4 and 5. [4].

$$u_k = b_k + \sum_{i=1}^{n} \text{conv1D}(w_{ik}, x_i) \tag{4}$$

$$y_k = f(u_k) \tag{5}$$

With continuously increasing computational power and developed programming tools CNNs have become viable options for many applications. The capibilities of CNNs are used widely with signal, image and video inputs. Some of these applications include speech recognition, real-time electrocardiogram monitoring and vibration-based structural damage detection. [5]

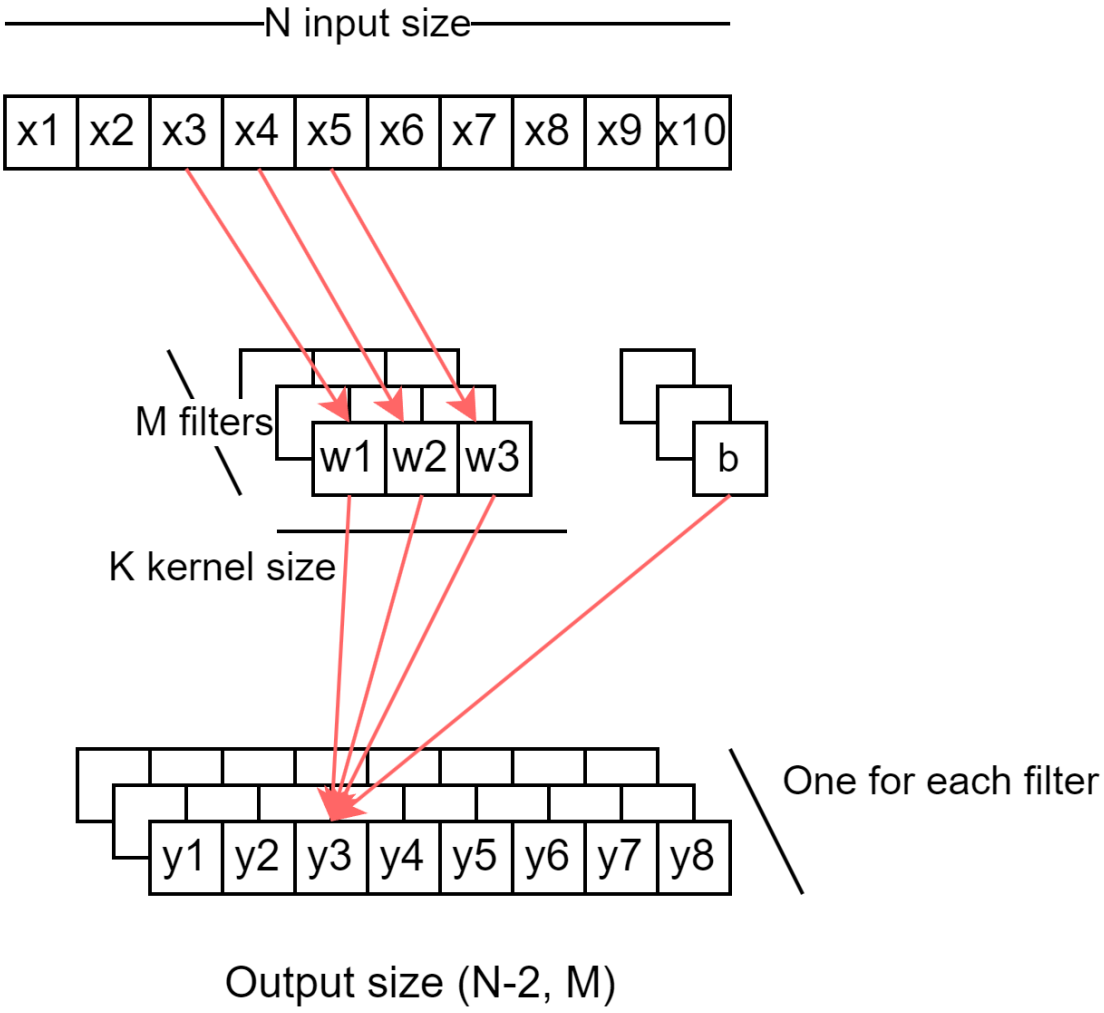

Figure 3. Basic idea of single layer CNN

2.3 Maximum pooling

Maximum pooling is a technique that is often used with convolutional neural networks to downsample the spatial dimensions of input data. The primary goal of maximum pooling is to reduce the amount of information while retaining the essential features necessary for the accuracy of predictions. Pooling is a kernel operation where a typically 2x2 kernel slides over the input data. In maximum pooling, within the kernel, the maximum value is selected to be saved to the pooled output. Hence, the advantage of maximum pooling is a faster convergence rate when selecting the best invariant features, leading to a better generalisation result. [6] [7]

2.4 Global average pooling

Average pooling is kernel operation, where only the average of the kernel is taken and saved for the output. In global average pooling the aim is to replace fully connected layers in classic convolutional neural network. This is done by creating one feature map for each corresponding classification task class in the last mlpconv layer. At the point, where fully connected layers would be added on top of the feature maps, in average pooling only the average value of every feature map is taken, and the result vector is directly given as input for the SoftMax layer. [8] [9]

One advantage of using global average pooling, compared to fully connected layers, is that average pooling better aligns with the convolutional structure, therefore enforcing correspondences between classes. Second advantage is that in average pooling, there are no parameters to optimize, which is why overfitting can be avoided for this layer. In addition, global average pooling summarises spatial information, which makes it more stable in relation to spatial changes. [8]

2.5 Regularization

In machine learning regularization refers to different methods that aim to balance the bias-variance tradeoff - to avoid overfitting. Typical regularization methods are for example l1 and l2, which add a term to the loss function proportional to the weights in layers kernel. Another common implicit regularization method is a dropout layer, which randomly sets some 'connections' between layers to zero, so the model learns new connections because it can't rely on always using the same connection.

Since we do hyperparameter optimization where we choose models based on the lowest validation loss, we do not want to use l1 or l2 regularization as a tunable hyperparameter, because it increases validation loss due to the penalty term.

We also do not want to use dropout layers, because we have a fully convolutional neural network, and dropout between convolutional layers doesn't have the desired effect [10].

A quite typical way of regularizing CNN's is batch normalization between convolutional layers. Batch normalization calculates the z-score of each neurons output for each batch, and forwards that instead of the true output. This can improve generalization and stabilize and speed the training process since the gradient is more stable. At inference the z-score is based on the training data [11].

2.6 Hyperband algorithm

Hyperband algorithm is a method that can be used to optimize hyperparameters of a model. These hyperparameters can include number of layers, number of trainable parameters per layer or choice of activation function. Hyperband algorithm uses successive halving method to optimise parameters in combination with other efficient resource allocation methods to find parameters that produce minimal validation error error. Successive halving uses finite number of parameter options and starts training the model with all possible combinations of them. After given time the number of parameter combinations is halved and resources are dedicated to train the better half. These steps continue until there is one model remaining. [12]

3 Methods

3.1 Preprocessing

The data is preprocessed to have a a constant length which is required by a neural network. We do this by finding the longest sequence of coordinates, and then padding every other sequence to the length of the longest sequence by appending coordinates (0,0,0) as long as is required.

3.2 Architecture

We started with an approximation of an architecture by using standard methods for similar problems. In our architecture, we did not want to optimize everything and we chose to keep certain hyperparameters fixed based on our intuition of the architecture.

We did not try different activation functions for any layers, since we had no reason to believe other activation functions than rectified linear (Relu) (equation 6) and softmax (equation 7) would produce better results.

$$ReLU(x) = max(x, 0) \tag{6}$$

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{7}$$

where z is the vector of raw scores (logits) and K is the number of classes.

The problem is a classification problem with 10 classes, and so our last layer had a softmax activation, which essentially turns the outputs of the last layer to a probability distribution over the classes (digits). We then calculate the difference between the predicted, and the true distribution (one hot encoded) using log loss (equation **??**).

$$H(q, p) = -\Sigma_i q_i * log(p_i) \tag{8}$$

where q_i is the true probability, so 0 or 1 in our case, and p_i is the predicted probability for class i.

Another design choice we kept fixed was to double the number of filters in each convolutional layer the deeper we went to the model. There is a plethora of reasons for doing this, but perhaps the most intuitive one is that the first layers capture more of the 'big picture' such as curves, and the later layers capture more nuanced patterns of which there are more. Having this intuition also reduces hyper parameter optimization time, since we only optimized the starting number of filters for both convolutional blocks. This is demonstrated in Figures 4 and 5.

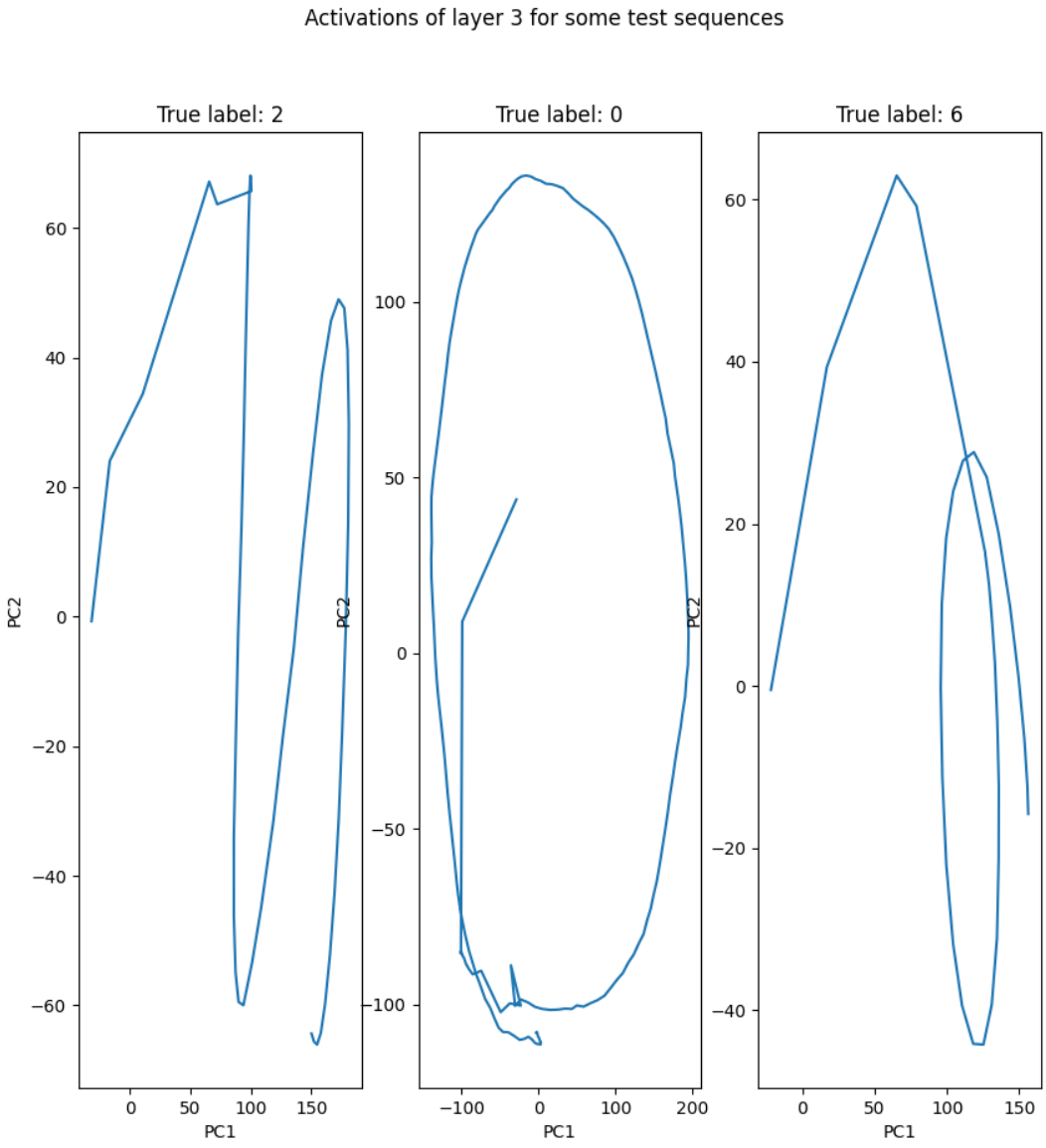

Figure 4. The full output of the 3rd convolutional layer for some inputs projected to two channels for visualization. It is clear that there is still clear correspondence to the inputs.

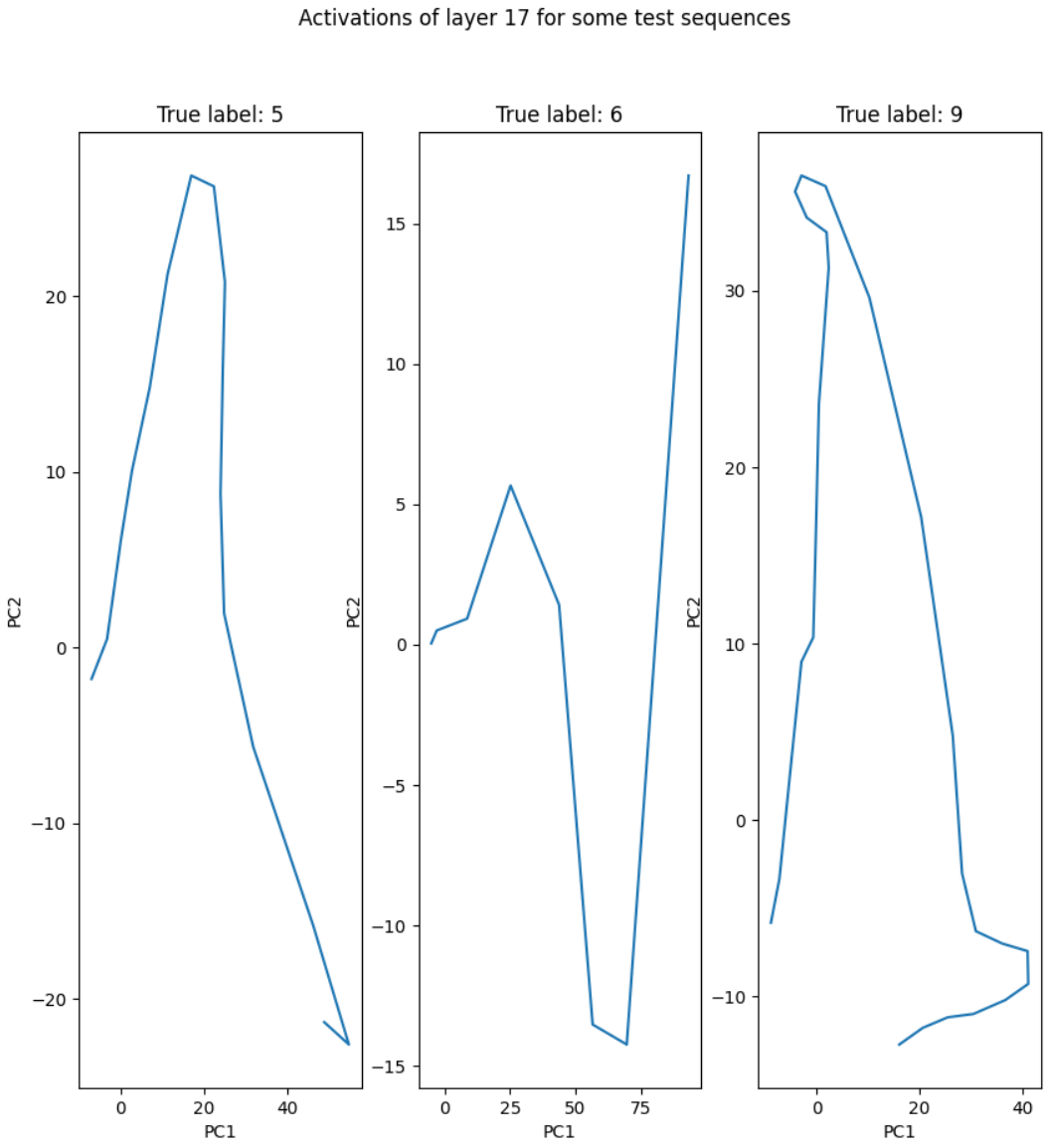

Figure 5. The full output of the 17th convolutional layer for some inputs projected to two channels for visualization. The features are more nuanced and not clearly discernible for humans.

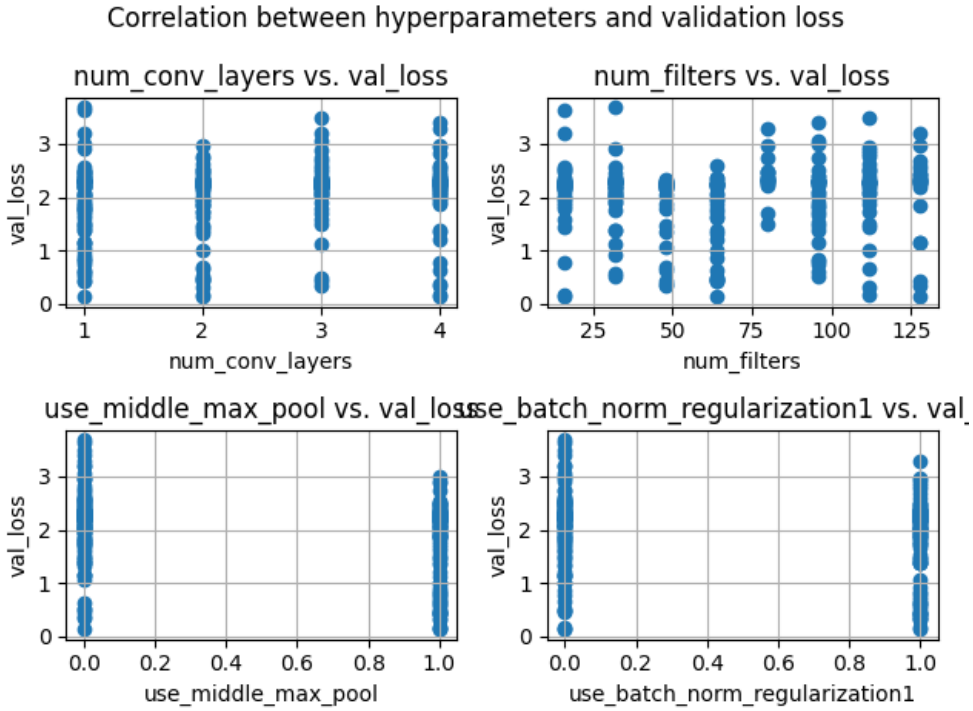

Figure 6. Scatter plot showcasing the effect of different hyperparameters on the validation loss.

4 Results

4.1 Hyperparameter optimization

As our final model, we reached a big model with 2.2 million parameters. In total we have 6 convolutional layers, where each is followed by a batch normalization, and a ReLU activation. In the middle of the network we use a max pooling layer, and the second to last layer is a global average pooling layer.

For hyperparameter optimization we used 60% of the data, and chose a model based on validation loss on 20% of the total data.

After hyperparameter optimization we re-trained the best model architecture from scratch but this time we used the training and validation data, so 80% of the data in total. We used 20% of the data as validation data (for the early stopping mechanism) for this later training process.

Figure 6 shows a scatter plot between some of the hyperparameters (x-axis) and validation loss (y-axis).

4.2 Training process

Training the model was a bit noisy, and a lower learning rate and larger batch size would be recommended.

During training we use an early stopping strategy, that ends training if the validation loss has not decreased in 20 epochs, and restores the best weights of the model. This was done to avoid overfitting.

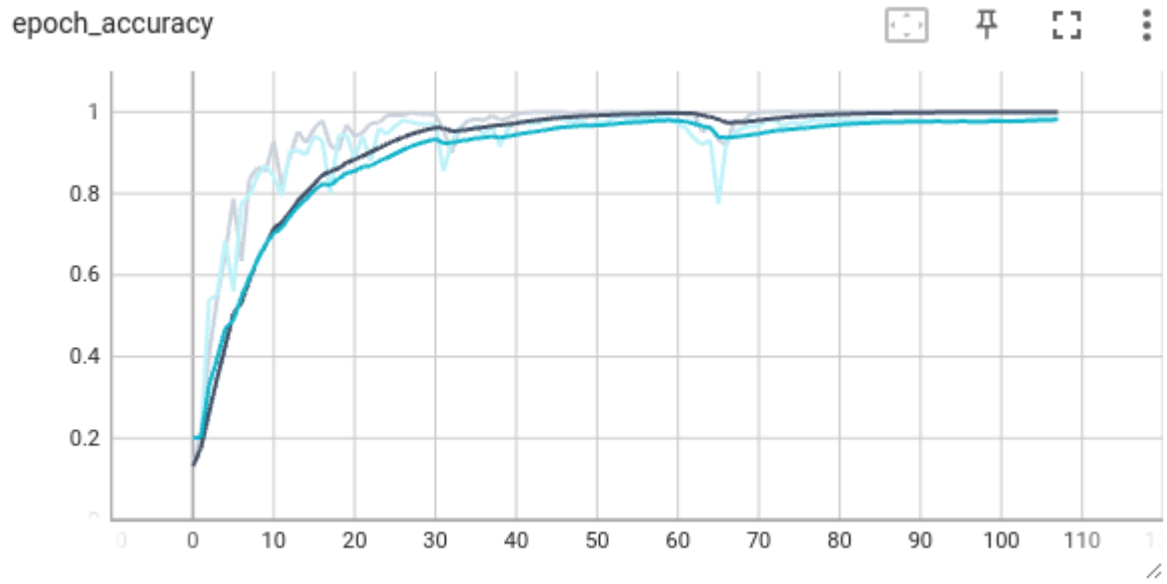

Figure 7. The accuracy on training and validation data during the second phase of training. Validation loss is the light blue line.

The convergence on the training process is shown in Figures 8 and 7. In the end, we reach a 99.5% accuracy on the test data, with a log loss of 0.06.

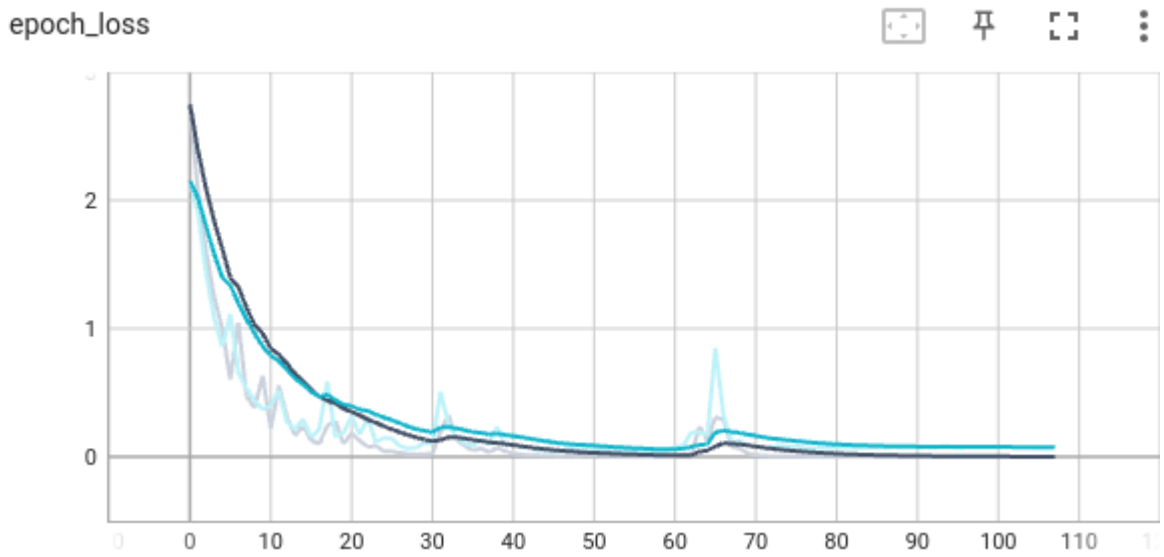

Figure 8. The log loss on training and validation data during the second phase of training. Validation loss is the light blue line.

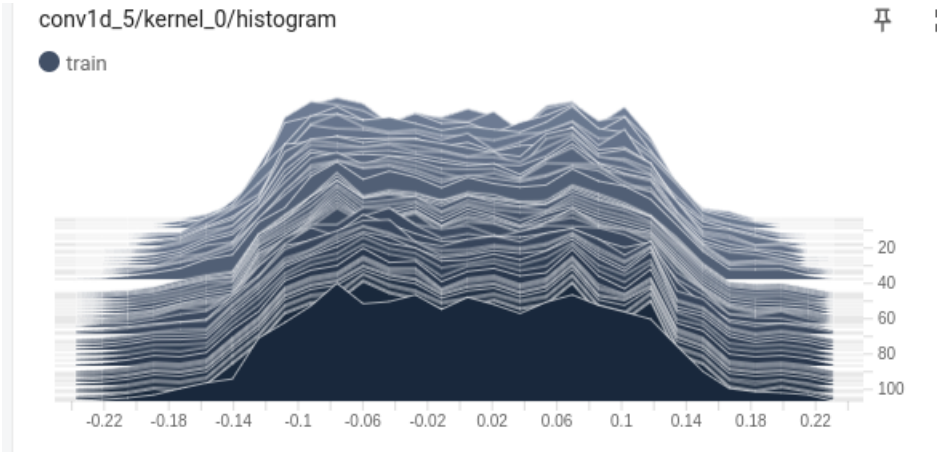

Figure 9. The weight distribution of the first convolution layers kernel as it progresses through training.

5 Conclusion

We selected convolutional neural networks to model the data, because of it's ability to capture spatial relationships and shapes which are very prevalent in this type of modeling. Regular fully connected layers would essentially draw statistical connections between each coordinate, and model the phenomena that way. But with a CNN, we instead draw statistical correlations from patterns of coordinates which is much more natural.

Our approach works well and matches the accuracy of state-of-the-art CNN's on a similar task of 2D digit recognition with the MNIST dataset.

References

[1] Rogers Mark. Menzies, Tim. Sharing data and models in software engineering. pages 324–326, 2015.

[2] Lasse Lensu. Pattern recognition and machine learning: Lecture artificial neural networks. pages 1–18, 2023.

[3] Shui-Hua Wang, Preetha Phillips, Yuxiu Sui, Bin Liu, Ming Yang, and Hong Cheng. Classification of alzheimer's disease based on eight-layer convolutional neural network with leaky rectified linear unit and max pooling. *Journal of medical systems*, 42(5):1–11, 05 2018. Copyright - Journal of Medical Systems is a copyright of Springer, (2018). All Rights Reserved; Last updated - 2023-11-27.

[4] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

[5] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021.

[6] Madhu Jain Dinesh K. Sharma. Data analytics and artificial intelligence for inventory and supply chain management. pages 139–140, 2022.

[7] Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE international conference on signal and image processing applications (ICSIPA)*, pages 342–347. IEEE, 2011.

[8] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[9] Muhamad Yani et al. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201 012052, 2019.

[10] Jacob Reinhold. Dropout on convolutional layers is weird, 2022.

[11] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018.

[12] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The journal of machine learning research*, 18(1):6765–6816, 2017.