DAT565 Assignment 7 – Group 99

Fabian Kaneby - (7 hrs)
Johanna Norell - (6 hrs)

March 1, 2024

Preprocessing

1.1 Explain the data pre-processing highlighted in the notebook

The data is converted from an 8-bit integer (able to take values between 0 and 255) to an float to make sure it can handle fraction values as the data in the next step is normalized to be a number between 0 and 1. Normalizing the data is done to make the training of the model easier to speed up the process and achieve better performance as the learning algorithm performs better on small values.

The last preprocessing step that is done transforms the data from an array to the encoded format, here one-hot encoded where the data is represented by a binary matrix.

Network model, training, and changing hyper-parameters

Model Parameters

How many layers does the network in the notebook have?

The model has the following layers:
1. Flatten()
2. Dense(64, activation='relu')
3. Dense(64, activation='relu')
4. Dense(num classes, activation='softmax')
So, it has a total of 4 layers.

How many neurons does each layer have?

The Flatten() layer reshapes the input without having any trainable parameters, so in that sense it does not have any neurons in the traditional sense.

The second and third layers are specified to have 64 neurons each.

The fourth layer is specified to have as many neurons as we want the output to have, so in this case 10 neurons representing the numbers 0-9.

What activation functions and why are these appropriate for this application?

So the first layer does not have any activation function since it does not have any neurons. The next two layers are specified to use the *Rectified Linear Unit* as activation function, This function is used to diminish issues related to the gradient and is normally used in deep learning to make it computational efficient. The last layer uses *Softmax* as its activation function which is suitable for classifying the output into multiple classes. It

The first two Dense layers use the ReLU (Rectified Linear Unit) activation function. ReLU is widely used in deep learning because it helps with the vanishing gradient problem and is computationally efficient. The vanishing gradient problem can cause early layers in the model to learn slowly, which ReLu thus helps to mitigate. The final Dense layer uses the Softmax activation function, which outputs a probability distribution over multiple classes. It's thus suitable for multi-class classification tasks.

What is the total number of parameters for the network?

From model.summary() we get the following:
Total params: 55050
Trainable params: 55050
Non-trainable params: 0

Why do the input and output layers have the dimensions they have?

The Flatten() layer does not really have any dimensions, but is more about transforming the input to a suitable type for the following layer to use. In this case it transforms something from a 2D shape (picutre) to something one dimensional (array).

The output layer, as previously mentioned, does have 10 neurons since we want to classify the input to one of the classes 0-9.

Loss function

What loss function is used to train the network?

The loss function used is `categorical_crossentropy` *(also referred to as log-loss)* which is normally preferred at classification problems and used if dealing with probabilistic outputs. It pushes the probabilities to the true labels by heavily penalizing predictions that are confidently wrong which is wishful in classification tasks.

What is the functional form (a mathematical expression) of the loss function? and how should we interpret it?

$$L(y, p) = -\sum_{i=1}^{C} y_i \log(p_i)$$

Where:

- y_i is the true label for class i (1 or 0)

- p_i is the predicted probability for class i.

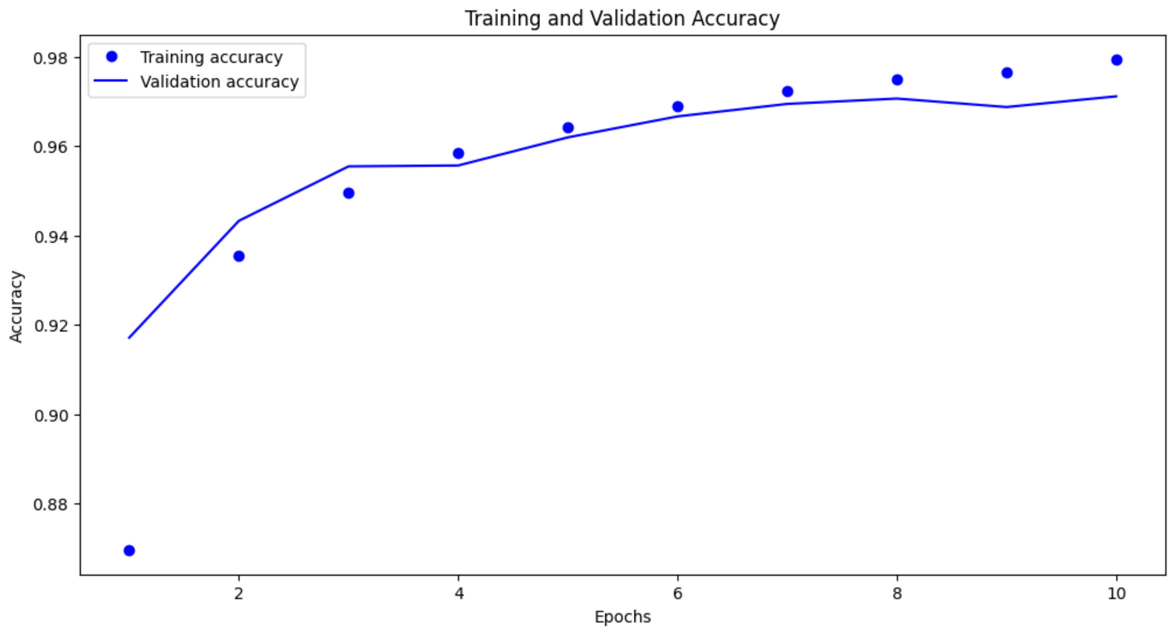
- C is the number of classes.

y_i will thus be 1 only for the correct class and 0 for all other classes, and the term $\log(p_i)$ will matter to the total loss when we have the correct class. The idea is then to relate the loss to the confidence of the prediction p_i by adding the logarithmic factor. If p_i small, that is the probability of predicting the right class is low, then we receive a big loss. In contrary, if the probability for predicting the right class is high, that is p_i close to 1, we receive a small loss. In this way we want to reward the model for making correct predictions, and penalize it for making incorrect ones.

Why is it appropriate for the problem at hand?

The greatest reason would be the nature of the problem, since it is a multi-class classification. Additionally, this function encourages the model to make accurate predictions especially when faced with ambiguous samples, which is common in the terms of classifying handwritten numbers. Generally, it also concludes stable and efficient convergence during training, due to the gradients being neither to small or too extensive which makes them more informative in comparison to other loss functions. Lastly, it has also showed great empirical success for this specific problem.

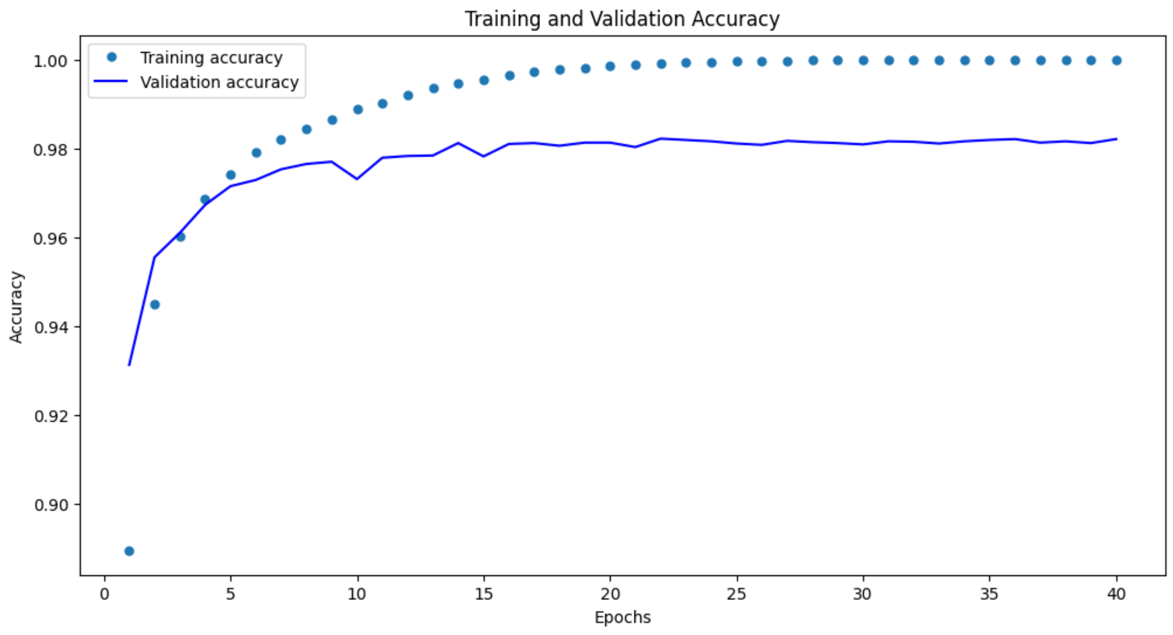
Training the model



At then end of the 10:th epoch, the model reaches a training accuracy of 97.9% and a validation accuracy of 97.0%.

Tuning hyperparameters

Update the model to implement a three-layer neural network where the hidden layers have 500 and 300 hidden units respectively. Train for 40 epochs. What is the best validation accuracy you can achieve?



In this case, the model reaches a training accuracy of 100% and a validation accuracy of 98.2% at the end of the 40th epoch.

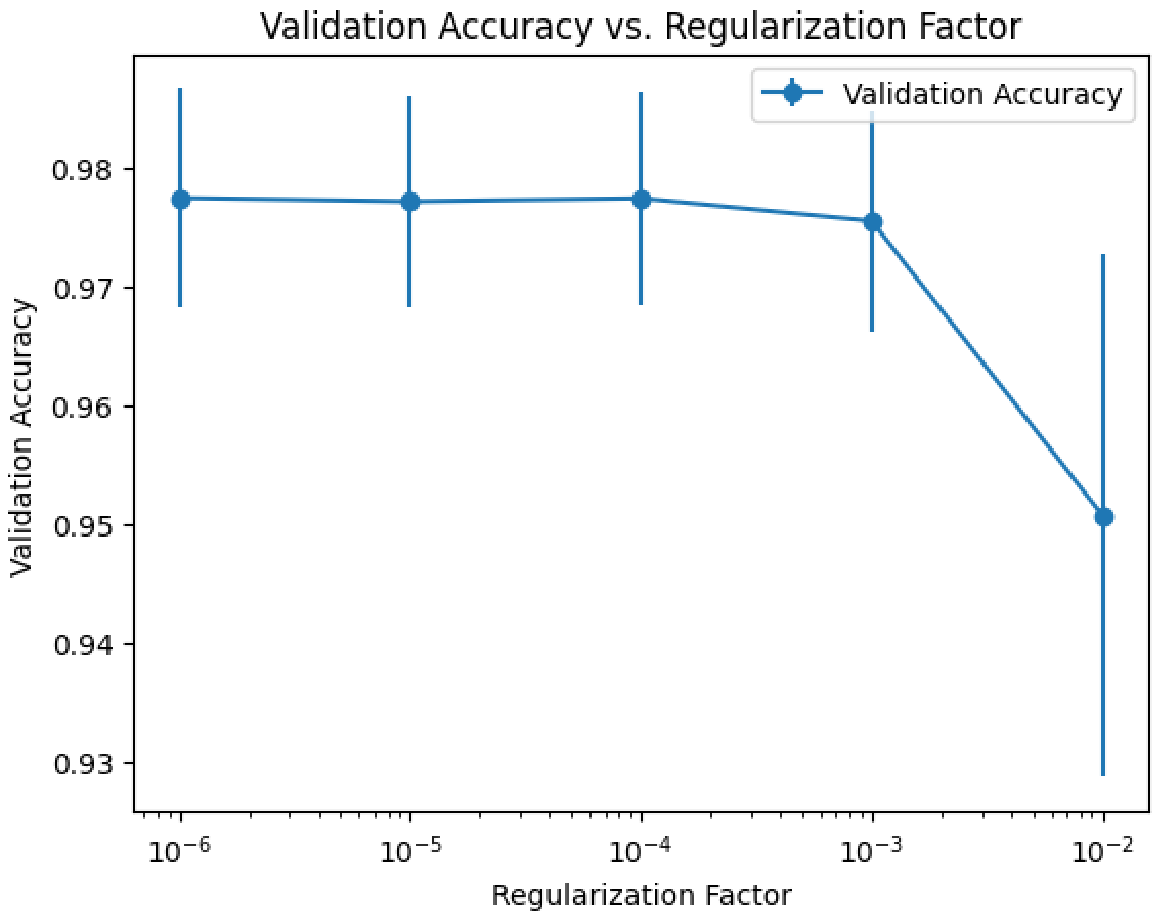
The increase in capcity from the additional neurons and the increased iterations improved the training accuracy with 2% and the validation accuracy with 1%. A 100% training accuracy could suggest overfitting the data, but since validation accuracy did also improve this should not be the case.

Implement weight decay on hidden units and train and select 5 regularization factors from 0.000001 to 0.001. Train 3 replicates networks for each regularization factor. Plot the final validation accuracy with standard deviation (computed from the replicates) as a function of the regularization factor. How close do you get to Hintons result?

The validation accuracies is displayed below for, we do not achive what Hinton was able too.

Validation Accuracy	Regularization Factor
0.9775	0.000001
0.9772	0.00001
0.9774	0.0001
0.9755	0.001
0.9507	0.01

Table 1: Validation Accuracy vs. Regularization Factor



If you do not get the same results, what factors may influence this?

Hyperparameters: Settings like learning rate, batch size, and regularization strength greatly affect training. Differences in these might be the cause of not reaching his results.

Initialization: Initial weights can significantly impact convergence and performance. Our inital weights might not have been identical to his.

Convolutional Neural Networks

Design a model that makes use of at least one convolutional layer – how performant a model can you get?

We implemented a model using Conv2D and MaxPooling2D, adding these twice to the model. From this we were able to reach a validation accuracy of 99.05%.

Conv2D, a key convolutional function for image processing, extracts essential features like edges and textures. It efficiently analyzes spatial patterns within images. MaxPooling2D complements Conv2D by downsizing feature maps, retaining vital information, and reducing computational load. This prevents overfitting and aids subsequent layers. Together, they significantly enhanced our model's performance, achieving an 99.05% validation accuracy in image recognition tasks

Discuss the differences and potential benefits of using convolutional layers over fully connected ones for the application?

In terms of connectivity, the neurons in fully connected layers takes the entire input image into consderation when processing the image, whilst convolutional layers focuses on a small local region of the picture. This enables the convolutional layer to learn specific patterns from parts of the image, making it less sensitive to if these regions shift slightly in the position of the image. It realtes to the hierarcical structure of the algorithm, which fully connected layers lack. This is particulary favorable when recognizing numbers, since we want it to be able to handle the cases where the position of the number is slightly different, but the number should not be interpereted differently. This is also related to translational invariance, where convolutional layers can recognize patterns it learned from one part of the image on other parts as well, which connected layers cannot.

Convolutional layers also have a benefit over fully connected layers in terms of the number of learnable parameters and followngly the efficiency of the computations when working with large scale image analysis. Fully connected layers have neurons connected to all neurons in the previous layer, causing a big number of neurons needed to train. Convolutional layers share parameters in a way that allows a small filter of weights to be fitted over the whole image allowing the user to specify the dimension of the filter. This enables adjustment to the wanted algorithm to prevent overfitting and complexity in the algorithm In this way the algorithm stays efficient while still being able to capture local features in the input images.