ML Unit 11 Final Assignment Transcript:

Hello and welcome to this presentation about neural network models for object recognition, where I will train and present a classifier for the CIFAR-10 dataset images.

I now start with presenting the structure of this presentation to you. First, in an introduction, I will give some reasoning why it makes sense to think about image and object classification. Then I will describe the data sets, prepare the data for the training phase of the model, define the neural network architecture I use, and essentially train the model and then in the evaluation phase watch how well it is suited to correctly classify the test data it has not been trained on. Then, in a conclusion, I will share some reflections about the process of building this neural network and training it and close with the list of references. And in order to allow you to focus on the slides in their entirety, I from now on turn off my camera.

The task of object recognition as seen in image one has gained more and more interest over time as it is necessary in many domains. Face recognition for the unlocking of phones or computers is only one example. Here, deep learning models and particularly convolutional neural networks, or CNNs, have significantly improved the accuracy and efficiency of classifiers. Applications of CNNs include, among others, autonomous driving, video surveillance, and medical imaging. In all these fields, deep learning algorithms like CNNs are used to detect and or classify objects in, for example, images. For example, in medical research, images of a patient can this way be classified as healthy or sick. CNNs are especially helpful for object detection tasks because they automatically extract hierarchical features from images, which enables the recognition of complex patterns without the need for manual feature engineering. In the following I will show you the training and evaluation of my own convolutional neural network for classifying the CIFAR 10 dataset.

Image two shows you a visualisation of the dataset used for this task, which is a CIFAR 10 dataset, which consists of 50,000 training images and 10,000 test images with a balanced distribution across ten classes, which are airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. This dataset is frequently used to test neural networks ability to classify images and is challenging because of its rather low resolution and small image size.

Preparing the data, first of all, I normalise the pixel values of the images from The range of 0 to 255 to the range of 0 to 1. This is done for the images in the training set as well as for the images in the test set, simply by dividing both of them by 255. This is done because it improves the efficiency of activation functions, as many activation functions operate optimally when inputs only comprise a small range. Furthermore, this data normalisation also accelerates the conversions during training because it ensures that gradients remain well conditioned. Then I also split the original training data set into two subsets. 80% of the images are in a training data set, and 20% are in a validation data set. This split-up is done randomly to prevent any bias and prevents overfitting because already during the testing, the model's performance is evaluated against data it was not trained on, showing overfitting already in the training period. This can then be used for hyperparameter tuning, for example, by setting a dynamic learning rate in dependence of the development, the validation precision, or loss over the epochs that the model is trained for.

I now show you how exactly I put together my CNN as classifier for the CIFAR 10 dataset. As you see, I start with initialising a sequential model in which layers are then added and stacked linearly. The first convolutional layer applies 64 filters with a 3x3 kernel to extract spatial features from the input image. The ReLU activation function is used to introduce nonlinearity and the L2 regulariser is used to reduce overfitting by constraining the weights to remain rather small and persistent, focusing on general patterns that are likely to generalise well to unseen data within the training phase. I then add batch normalisation, the next step, to normalize the activations to stabilize training and speed up convergence. Here, internal covariate shifts are reduced and the use of higher learning rates is enabled, which allows to train the model for less epochs until convergence, which speeds up the training process. Then, I also add MaxPooling, which downsamples the feature map by selecting the maximum value in a 2x2 region, which this way reduces spatial dimensions. And overall this reduces computational complexity and still captures dominant features.

Now, I add a second and third convolutional layer, in which I increase the depth to 128 filters to enable the extraction of more complex features. As the network deepens, it captures high-level features such as object parts and shapes. After these layers, I introduce a dropout to increase the generalisation of the model to data it has not been trained on. Dropout randomly deactivates a subset of neurons during training, preventing the model from becoming overly reliant on specific neurons and enabling the learning of robust distributed representations. This reduces overfitting by introducing noise to the network, this way making it more resilient to variations in unseen data. Here I have chosen to set the proportion of deactivated neurons to 40 percent. This is a trade-off between preventing overfitting and still retaining enough network capacity to learn complex patterns and presents a common best practice.

Next, a flattening layer converts the two dimensional feature map into a one dimensional vector as input into the dense layer. This step is necessary because dense layers require a one dimensional input vector. And the dense layer is used to learn high level global representations by fully connecting all neurons, which allows the model to capture complex relationships between the extracted features. This layer integrates the spatial features from the previous convolutional layers and outputs predictions tailored to the classification task, and then again adds a batch normalisation and a dropout. The dropout rate is now a bit smaller on deactivating 30% of the neurons. This is the case in order to prevent the risk of underfitting due to a too high proportion of dropped neurons. The final layer is also called output layer, and it is a dense layer with 10 neurons, one for each CIFAR-10 class. Using the softmax activation function ensures that outputs are interpreted as probabilities for multi class problems, and I will give some more detail about the choice of activation functions in a second.

In my neural network architecture as activation functions, I've used the rectified linear unit activation function, or short RELU, and the softmax activation function. Figure one shows you a visualisation of the RELU activation function, while figure two shows you a visualisation of the softmax activation function. The ReLU activation function is used in the convolutional layer and in the dense layer of the CNN to introduce non linearity, while at the same time it is computationally efficient and easy to optimize. It allows the network to model complex relationships by activating only neurons with a positive input value, this way improving sparsity in the model. At the same time, the simplicity of the ReLU activation function ensures effective gradient flow during backpropagation. The softmax activation function, on the other hand, is then applied in the output layer of the CNN, it converts the model's outputs to probabilities, ensuring that they sum to one across all classes, which allows the assignments of probabilistic confidence scores to each class. Now that I have defined the architecture of the neural network, I turn to the model compilation and training.

I now compile the model to prepare and configure it for training by specifying the optimizer, loss function, and evaluation metric. I choose the adaptive moment estimation optimizer, or short ADAM, for updating the model's weight during training based on the gradient computed from the loss function. It dynamically adjusts the training rate for each parameter using estimates of first and second moments of the gradients, this way improving conversion speed and stability. The sparse categorical cross entropy loss function calculates the error between the predicted class probabilities that are delivered as output of the softmax layer and the true class labels, which are, in case of this loss function, integers, making one hot encoding not necessary. This ability to avoid one hot encoding, a generally good fit for multi class classification problems with multiple mutually exclusive classes, as well as its close alignment with the softmax activations function I use in the output layer, are the reasons for choosing this loss function. As evaluation metric, I choose accuracy, which is the number of correct predictions divided by the total number of predictions. The evaluation metric is used to receive immediate feedback on how well the model is learning during training and on unseen validation data.

With data augmentation, I artificially increase the size of the training data set by generating transformed versions of the original images within that set. I allow the images to rotate up to plus or minus 5 degrees, shift a maximum of 5 percent horizontally as well as vertically, and also allow the images to flip horizontally. This way, by rotating, I create slightly different images that each still belong to the same class. How these slightly changed images can look like can be seen at image three, where a butterfly is slightly rotated, which results in four different images. I use data augmentation because it prevents overfitting by increasing the data diversity and this way ensuring that the trained model generalizes better to unseen data. This generalisation is improved via data augmentation because the model learns to recognize objects under varied conditions, which are closer to the real world scenarios that the model would face.

Early stopping is used to stop the training if the validation loss does not improve for three consecutive epochs, defined by the patience of three that I set. I also ensure that when training stops, the model's weights are set to their state from the epoch with the lowest validation loss. Additionally to early stopping, I reduce the learning rate whenever a plateau is reached. Whenever the validation loss does not improve for two consecutive epochs, the learning rate is reduced by a factor of 0.5, starting with a learning rate of 0.001. Finally, for training the CNN, I use datagen.flow, which introduces an additional data augmentation during each epoch, again, preventing overfitting by increasing the variety of training images. A batch size of 128 is chosen, keeping a balance between the performance and the speed of this trained model. I choose to train for 30 epochs to give the model enough time to use the dynamic learning rate reductions and early stopping while still ensuring that training does not take an excessive amount of time. As already explained before, the validation of the model's performance after each epoch is done using the validation data set, which the model has not seen before during training.

Here, table one shows to the learning weight reduction and early stopping in practice. We see that between three and five the validation loss does not improve, which leads to a learning rate reduction as set in the code by a factor of 0.5 and then we see that after several learning rate reductions between epochs 24 and 27 there are no improvements in the validation loss and thus the early stopping gets triggered and the training stops.

Now I evaluate the test results of the classification. On the left side you see the development of the training and validation accuracy over the epochs in figure 3 and figure 4 on the right side shows you the validation loss over the epochs. which essentially is the same curve just turned upside down. We

here see the effect of reducing the learning rate when a plateau is reached, as this smoothens out the development of accuracy and loss in later epochs, preventing too much volatility. We see that in the latest epoch, the training accuracy is at 84.1 percent, the validation accuracy is at 82.02 percent. The test accuracy lays at 81.42%, which signals only very slight overfitting, as it is slightly below the training accuracy. This test accuracy means that the classifier correctly predicts the class label for about 81 out of 100 images in the CIFAR-10 test set, which is quite a good result, signaling a performant classifier. However, an error rate of almost 20 percent remains that signals that the model also struggles with creating correct classifications for some classes or images and could be improved.

Here, I present more insights on which classes are classified good by the model and where it faces problems. Table 5 presents a confusion matrix which compares the predicted and true class for each test image and table 2 shows a listing of precision and recall values for each single class. We see that the classification worked especially good for automobiles and trucks, with the slight exception that automobiles are sometimes confused for trucks and vice versa. We also see that the model relatively often misclassified cats and dogs. Cats are mainly confused with dogs, frogs and birds, while dogs are mainly confused with cats, horses, and birds. Generally, the confusion matrix shows that classes are mainly confused with classes with which they share some features, like automobiles with trucks or cats with dogs, which makes the model create a wrong classification. The precision and recall values associate numbers to these findings, stressing that both for cats and dogs, the model has problems to identify these, which is associated with a relatively high false negative rate and a low recall value.

I now present my key takeaways from working on this assignment. It is especially interesting to see that with a relatively simple model and only 13 minutes of training time I can create quite a good classifier for the CIFAR-10 dataset with a test accuracy of above 80 percent. We saw that the model works good for automobiles and trucks and not that good for cats and dogs. If images are classified incorrectly, then most of the time they are falsely associated to classes by the model, which also for the human eye share some features with the real class of the image. Within the process of building this classifier, I faced some challenges. Avoiding both underfitting and overfitting turned out to be quite difficult and made a lot of adjustments of different parameters as well as a dynamically decreasing learning rate necessary. Also keeping the model simple enough to ensure a short training time and at the same time still ensuring good enough metrics proved challenging. Remaining weaknesses of the model's performance are caused by existing similarities between the features of different classes, which leads the model to make wrong classifications. Thus, there obviously is some room for improving the performance of the classifier. I could spend some more time tuning the hyperparameters of this existing architecture, for example, by changing the initial learning rate or the degree by which it is dynamically decreased. I could also use additional regularisation techniques like DropBlock, which selectively drops regions and feature maps to additionally improve the generalisation. Additionally, I could use a more complex neural network architecture. Models like EfficientNet can also be accessed as pre-trained versions and then fine-tuned for the given data set. They achieve test accuracies of up to 90%. Finally, I deliver some example use cases of CNNs in practice. In e-commerce, they are used to classify products in different categories by extracting features and uploading images. For autonomous vehicles, objects are detected and inform their automatic steering. And furthermore, for the Unlocking of mobile devices, CNNs are used for facial recognition. These are only a few examples and show the importance of CNNs for object recognition in the present and future of human interaction, this way justifying additional research and making additional improvements necessary.

Here you get a first look on my list of references.

You now see the final references. Please note that together with this video recording, I will also upload the slides and the code that I used. And with that, I thank you very much for your attention and come to an end of this presentation.