Part I - Language identification with CNN

Preprocessing
As this is quite important for the model, we decided to quickly explain what we did to preprocess the tweets:

1. remove mentions
2. remove urls
3. remove emojis
4. lowercase the text
5. clean punctuation and numbers
6. normalize the tweets: will be discussed further down
7. Balance data set: will be discussed below

We think that none of the above mentioned special words can tell us anything about the language. Mentions and urls, for example, are part of any tweet no matter the language. One could argue that the name in the mentions or parts of the url might be different for each language but we think that this does not matter enough. Also, many usernames are actually English, even though the tweet might be written in any other language. Not filtering them out could worsen the performance of the system. Also punctuation symbols and numbers are very international as also, for example, Asian languages that have a different writing system use the same numbers as European languages.

We represented each tweet as a sequence of single characters and created a vocabulary that consists of characters mapped to a unique number. Doing this we could replace the characters and use the number from the vocabulary to represent each character in each tweet. Each tweet is therefore represented as a sequence of single numbers.

Normalizing tweets
As mentioned above, we normalized the tweets. After the preprocessing was done, we brought all the tweets to the same length. This is important as the cnn needs inputs of the same length. This means that we had a maximum length of 200 characters in each tweet. Tweets that were longer than that were simply shortened. But there are actually not many tweets that are longer than this (only about 100). Tweets that are shorter than this were simply filled with '0' until the maximum length was reached. The number '0' is in the vocabulary mapped to <UNK>. This means that it simply refers to any unknown character. As many tweets will have unknown characters (a lot of 0), the model will quickly learn that those are not important.

Balance dataset
Just as for the first exercise we balanced the dataset and did the following steps:

1. Reduction of the number of classes
Combine all classes with a number of samples below a threshold (in our case, all classes that had less than 100 samples) into one bigger class called 'other'.

2. Resampling
Resample the dataset to balance the number of samples in each class. After we did the first step, we resampled all of the classes to have the same amount of samples (in our case this is 6000)

We are not going to explain this in detail anymore, because we have already discussed this in class and in the tutorials for the first exercise and again for this one in the last tutorial.

Ablation study

We tried a lot more models than the below five because we wanted to understand what influences the models the most. We therefore do not show the best models below but the models for which it is clearest how they are influenced by the hyperparameters. Generally we noticed that our accuracies are not very good (at least on the test set, the other ones were very good). We were thinking that maybe our model is overfitting and therefore produces very good results on the train and validation set but not on the test set.

Note that those models in the table are not necessarily the ones that we last ran in the notebook. We included the most interesting ones here to study the hyperparameters.

# study	hyperparameters	performance (accuracy on test set)
1	a) Adam b) 0.01 c) none d) 32 / 128 e) 1 f) 2 / 3 g) maxPooling h) 50	accuracy: 0.0083
2	a) Adam b) 0.0001 c) none d) 32 / 128 e) 1 f) 2 / 3 g) maxPooling h) 50	accuracy: 0.8471
3	a) Adam b) 0.0001 c) none d) 32 / 128 e) 1 f) 2 / 3 g) none	accuracy: 0.8431

	h) 50	
4	a) Adam b) 0.0001 c) none d) 32 / 64 e) 1 f) 2 / 3 g) maxPooling h) 30	accuracy: 0.7754
5	a) RMSprop b) 0.0001 c) none d) 32 / 128 e) 1 f) 2 / 3 g) none h) 100	accuracy: 0.8602

Effect of different hyperparameters

Note that not everything that we mention below is visible above in the table, as we only included 5 models, but tested many more to arrive at our conclusions and understand the model.

a) **optimizer**
We tried two different optimizers: Adam and RMSprop. We read in various places that those were the two most useful optimizers for sentence classification tasks. However, there was not a big difference in the performance of the model when using the two. We think that this might be due to their similarity. It might have been better to choose two very different optimizers to see a clear difference. Our best model uses RMSprop but we did not generally notice that model performed better using RMSprop. We think that in the best model this might be due to other factors.

b) **learning rate**
We found out that a low learning rate increased the model's performance. We think that this is due to the fact that if the learning rate is too big, the model makes steps that are too big and misses the optimum. We can clearly see this observation in the first study which is really, really bad. The comparison to the second study is quite interesting as there the accuracy is so much better and the only thing that was changed is the learning rate.

c) **dropout**
We did not always include a dropout layer as we noticed that this did not improve the results and rather made them worse. We are not quite sure if dropout, despite being crucial for image processing, makes sense when we are talking about a language classifier. As far as we understood, the dropout in this case basically hides some of the characters in each sentence. We think that for a language classification task it does not make sense to use dropout as we need all of the characters to properly classify a sentence.

d) **# of filters**
Surprisingly, we did not notice a big influence of the filters on the results. We tried to use different ones for both convolutional layers. But generally we noticed that a lower number of filters in total produced worse results which can be seen in study 4 (note that the score in study 4 might also be low due to the changed batch size). If we think of the filters as something that detects features in the data, it makes sense that more filters produce better results, as more features detectors can find more and better features. But it is a bit surprising to us, that the differences are not bigger.

e) **different strides**
We tried two different strides 1 and 2. Using stride 2 worsened the models performance by just a bit. But to us it makes sense that stride 1 is better than stride 2, as stride 2 simply skips certain n-grams. E.g. for the given sentence 'the dog' with kernel size 3 (i.e. n-gram size 2), it produces the n-grams: ('th', 'e_ ', 'do', etc.), but not 'he'. However, this is not a very rare n-gram in English and it could therefore be misleading to exclude it.

f) **different kernel sizes**
As we represent the sentences as single characters, the kernel sizes are basically the n-gram sizes that are taken into consideration (if we understood everything correctly). We therefore figured that is does not make much sense to have kernel sizes (=n-gram sizes) that are very big as those are not very generally characteristic for a language. Interestingly enough this cannot be very clearly seen as in a study (that is not visible above) we chose kernel size 5 for both layers and the result is actually quite good. We think that this is because 5 is maybe still characteristic enough for each language. To prove our claim we must probably choose a bigger kernel size.

g) **different pooling strategies**
We decided to see what happens if we added no pooling layer at all. So we did that and if we had a pooling layer we chose to use a MaxPooling layer. If you compare study number 2 and 3, you can see that there is no great difference in the accuracy. Study 3 has no pooling layers at all, while study 2 has two MaxPooling layers after each convolutional layer. We think that maybe due to the data summarizing the features as the pooling layer does, does not help that much.

h) **batch sizes**
We noticed that a small batch size has an influence on the model. This can be clearly seen when looking study 4. We used a batch size of 30 for this study and the accuracy is quite bad. We think that this is because after each batch the model gets slightly changed. If there are many small batches, the batches do not cover the full diversity of the data. This means that after one batch it gets changed in this way and after the other it gets changed in a completely different way.

Best hyperparameter combination: accuracy = 0.8873

(we got this combination from talos, but the model is not in the notebook anymore as it we got the accuracy just as we got it for the other models)

a) optimizer: RMSprop
b) learning rate = 0.001
c) Dropout: none
d) filter size: 128 / 128 (first layer / second layer)
e) strides: 1

f) kernel sizes: 3 / 3
g) pooling strategy: MaxPooling
h) batch size: 100

3. Compare the outputs of the best CNN model to your best performing model from Exercise 1. Which classifier scores higher on the test set? Do you have an idea, why this might be?

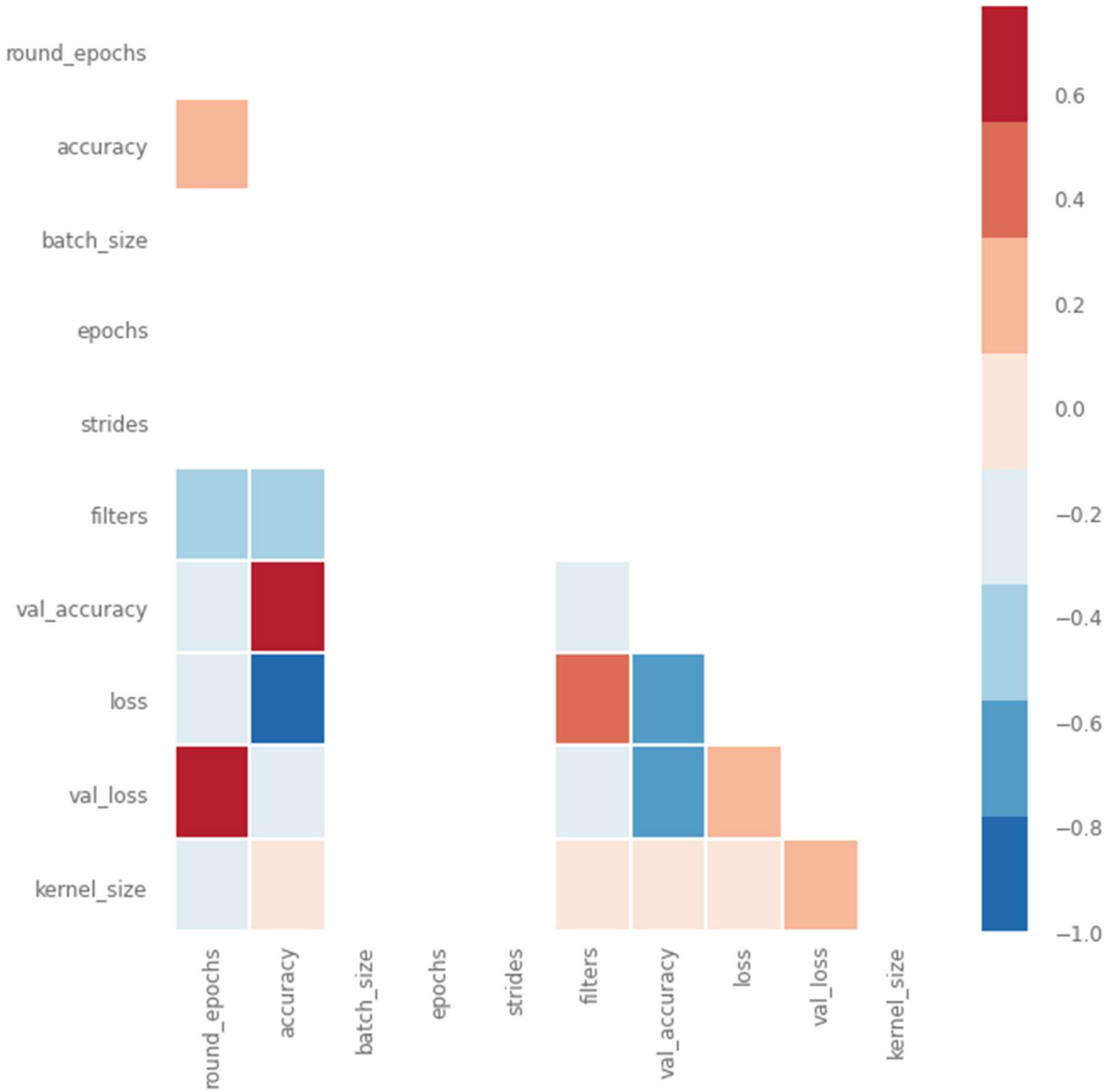
Our best model in exercise 1 achieved an accuracy of 87.94% and it was actually a linear model. So the best cnn model only achieves a similar accuracy as our best linear model. We think that this is maybe due to the data. We could not use the entire dataset in both cases because it was not very balanced. But this means that we did not have that much data as we could have had. And a neural model works well with a lot of data. So maybe if the data would have been better and more, the cnn would have achieved better results.
Another reason might be that cnn models have originally been developed for image processing, not language classification. Many of the unique features of a cnn (e. g. being able to process inputs in blocks and filtering out "interesting" areas) are not tailored for the task of language identification and thus we cannot really benefit from using a cnn to solve this task.

4. Use a confusion matrix to do your error analysis and summarise your answers in your report.

The confusion matrix is visible in the notebook and together with it we included a summary of which label got confused with which other label most. This result is quite interesting but not surprising: Malysian (ms) and Indonesian (id) got confused with each other quite a lot (over 60%). This is not surprising as those language are probably rather similar due to their geographical closeness (although we are no experts on those languages and we had to prove this better, but this is just our assumption). The confusion matrix or rather the list also shows that the unknown label got confused a lot with many languages as well. But the list also shows that not many labels got confused a lot with each other. But as we chose a model which has a rather good accuracy, this is not surprising. Additionally, we excluded very small classes and those would probably be the ones that get confused with other ones most often. Another language pair that got confused quite often (25%) was Italian and Spanish. But due to their similarity, this is not surprising as well.

5. Plot the Heatmap correlation between the hyperparameters above to understand the dependency of hyperparameters on each other and discuss.

We did not include all the hyperparameters in the talos search as we thought it better for our learning process to first test a few models manually and really understand what it does and then using talos on those hyperparameters that make the biggest difference. Further we noticed that talos uses a lot of RAM and we did not want to risk that the model stops running because it runs out of RAM. We could avoid the model to use so much RAM by telling it not to save all the models but as we want to retrieve the best model, we cannot do this. We can see in the heatmap that the different parameters do not correlate a lot. The colour is rather bluish and therefore the correlation low. But the filters and the kernel sizes do somewhat correlate. We can also see that the kernel size correlates a lot with the accuracy. We think that this might mean that it has a high influence on the accuracy in general.


round_epochs
accuracy
batch_size
epochs
strides
filters
val_accuracy
loss
val_loss
kernel_size
round_epochs
accuracy
batch_size
epochs
strides
filters
val_accuracy
loss
val_loss
kernel_size
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0