

# Lab Report of Exercise 3

## Approach

### 1. Read file

Read files into dataframes.

### 2. Clean the data

Remove URLs, numbers, punctuations, '@sb.', '#topics' and emoji.

### 3. Preprocess

Tokenize the corpus, create a dictionary of characters with characters as keys and indices as values, encode the tweets with indices of characters; one-hot encoding to vectorize labels.

### 4. Initiate CNN

CNN architecture: input -> embedding layer -> convolutional layer (with activation function Relu) -> pooling layer -> drop out -> fully connected layer -> output layer.

### 5. Hyperparameter search by RandomizedSearchCV

Find the best number of filters and kernel size.

### 6. Train an anchor CNN model

Split training set into training and validation sets; train an anchor CNN with the best number of filters and kernel size found in the hyperparameter search above; use early stop to track the validation loss; predict test labels by the model.

### 7. Ablation Study

Change the pooling strategy, dropout, batch size and strides one by one, predict test labels by the models.

## Technical Description

Table 1. Functions and Their Descriptions

Function	Description
----------	-------------

load_dataset	Load files into dataframes
clean_text	<ul style="list-style-type: none"> <li>● remove url</li> <li>● remove "#topics", "@sb"</li> <li>● remove numbers</li> <li>● remove punctuations</li> <li>● remove emoji</li> </ul>
preprocess	<ul style="list-style-type: none"> <li>● Use one-hot encoding to vectorize labels</li> <li>● Tokenize corpus and use character indices to encode tweets</li> <li>● Return preprocessed dataset</li> </ul>
create_model	<ul style="list-style-type: none"> <li>● Define CNN model</li> <li>● Use it in Randomized Search CV</li> </ul>
get_model	<ul style="list-style-type: none"> <li>● Initiate CNN</li> <li>● Split training data into training and validation sets</li> <li>● Use early stop to track the validation loss</li> <li>● Return trained model</li> </ul>
prediction	<ul style="list-style-type: none"> <li>● Predict probabilities by trained model</li> <li>● Get the predicted string labels</li> <li>● Print classification report</li> </ul>

## Experiments and Results

### 1. Hyperparameter Search

I used the RadomizedSearchCV with CV fold=4 to search the hyperparameter space regarding number of filters and kernel size. In order to reduce the training time, I only defined a small hyperparameter space, which is:

```
param_grid = dict(num_filters=[64, 128], kernel_size=[3, 5])
```

The best hyperparameter regarding number of filters and kernel size I find is:

**Best combination: {'num\_filters': 128, 'kernel\_size': 3}**

## 2. Ablation Experiments

I set some hyperparameters as follows, then change pooling strategy, drop out, batch size, strides and optimizer to perform the ablation study. The results are in Table 2.

Set: Embedding size = 100, max length = 100; number of filters = 128, kernel size = 3; learning rate = default.

Table 2. Results of Ablation Experiments

Experiments	Pooling Strategy	Dropout	Batch Size	Strides	Optimizer	Test Accuracy
<b>Anchor</b>	<b>global max</b>	<b>0.2</b>	<b>32</b>	<b>1</b>	<b>adam</b>	0.9291
<b>Change pooling</b>	<b>global average</b>	<b>0.2</b>	<b>32</b>	<b>1</b>	<b>adam</b>	<b>0.9306</b>
Change dropout	global max	<b>None</b>	32	1	adam	0.9187
Change batch size	global max	0.2	<b>16</b>	1	adam	0.9284
Change strides	global max	0.2	32	<b>5</b>	adam	0.8629
Change optimizer	global max	0.2	32	1	<b>sgd</b>	0.9175

I defined an anchor model, then changed hyperparameters including pooling strategy, dropout, batch size, strides and optimizer one by one to test the effect of these hyperparameters on test accuracy. As for the learning rate, I used the default one, as the default learning rate is experimented as the best one for most cases. The results are listed in the Table above.

**The best CNN model with hyperparameter combination:**

**Embedding size=100, max length=100, number of filters=128, kernel size=3, global average pooling, dropout(0.2), batch size=32, strides=1, optimizer=adam, learning\_rate=0.01.**

## Discussion

### 1. Hyperparameters and Their Effect on Performance

#### Number of filters

Filters are used to learn multiple features in parallel for the input. When increasing the number of filters, the number of parallel feature learning increases. Here, 128 filters is better than 64 filters.

### Kernel size

Kernel size is the width of the convolution filter. Here the kernel size 3 is better than kernel size 5, when combined with 128 filters.

### Pooling strategy

Training model with global max pooling is faster than with global average pooling, while the performance is not as good as global average pooling. This may be due to the loss of feature information when using global max pooling.

### Dropout

Drop out can help to avoid overfitting and reduce the training time. In the CNN model, when removing the drop out, the test accuracy decreased, which may be due to the overfitting on training.

### Batch size

Batch size can help to avoid overfitting and reduce the training time as well. I compared batch size 32 with batch size 16, the test accuracies are almost the same.

### Strides

I tested strides 1 and strides 5, the test accuracy of strides 1 is much better than that of strides 5. That's because when using large strides, much feature information is lost.

### Optimizer

Optimizer adam is much faster than sgd to get to the early stop epoch, and the performance is better than sgd here.

## 2. CNN vs. MLP

The best performing model from Exercise 1 is MLP. However, in exercise 1, I use CountVectorizer to encode each word and the token is word, here I encode each character and the token is character, so the data preprocess is different here from in E01.

I trained the MLP model again with the same preprocess data for CNN. The performance comparison for CNN and MLPs is as follow:

Table 3. CNN vs. MLP

Model	Macro Precision	Macro Recall	Test Accuracy
Best CNN	0.4347	0.3677	0.9306
MLP (with same preprocessed data as CNN)	0.1001	0.0595	0.5674

<b>MLP</b> (from Exercise 1, token is word)	0.3800	0.2403	0.8420
--	--------	--------	--------

CNN performs much better than MLP: The accuracy is higher, in the mean time, the average macro precision and average macro recall are much higher than MLP.

### 3. Confusion Matrix of the Best CNN Model

The best CNN model is the model with global average pooling shown in Table 2. As the confusion matrix is too large, here I only display part of the confusion matrix. **Please see the complete confusion matrix in the ipython notebook.**

Table 4. Part of the Confusion Matrix

	en	und	es	ja	id	pt	ar	th	fr	tr	ms	ru	hi-Latn	bs	xh	ko	it
en	4661	57	6	1	8	6	0	0	8	0	0	1	0	0	0	2	2
und	131	923	44	10	59	12	4	0	7	11	0	7	1	0	0	2	1
es	13	35	1400	0	2	17	0	0	3	1	0	0	0	0	0	0	3
ja	2	9	0	2458	1	1	0	3	1	0	0	0	0	0	0	2	0
id	26	52	3	1	719	4	0	0	2	4	0	0	0	0	0	0	0
pt	11	18	39	0	1	621	0	0	4	0	0	0	0	0	0	0	3
ar	0	0	0	2	0	0	525	2	0	0	0	0	0	0	0	0	0
th	0	0	0	0	0	0	0	97	0	0	0	1	0	0	0	0	0
fr	14	10	4	0	0	3	0	0	193	0	0	0	0	0	0	0	0
tr	3	4	0	0	3	0	0	0	0	164	0	0	0	0	0	0	0
ms	1	4	0	0	22	0	0	0	0	1	2	0	0	0	0	0	0
ru	0	0	0	3	0	0	0	1	0	0	0	239	0	0	0	0	0
hi-Latn	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
bs	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
xh	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ko	1	6	0	10	0	0	0	0	0	0	0	0	0	0	0	93	0
it	3	2	5	0	0	2	0	0	1	0	0	0	0	0	0	0	62

I listed the metrics below as examples, please see the complete table on ipython notebook for the precision and recall of all the classes.

Table 5. Metrics of Some Example Languages

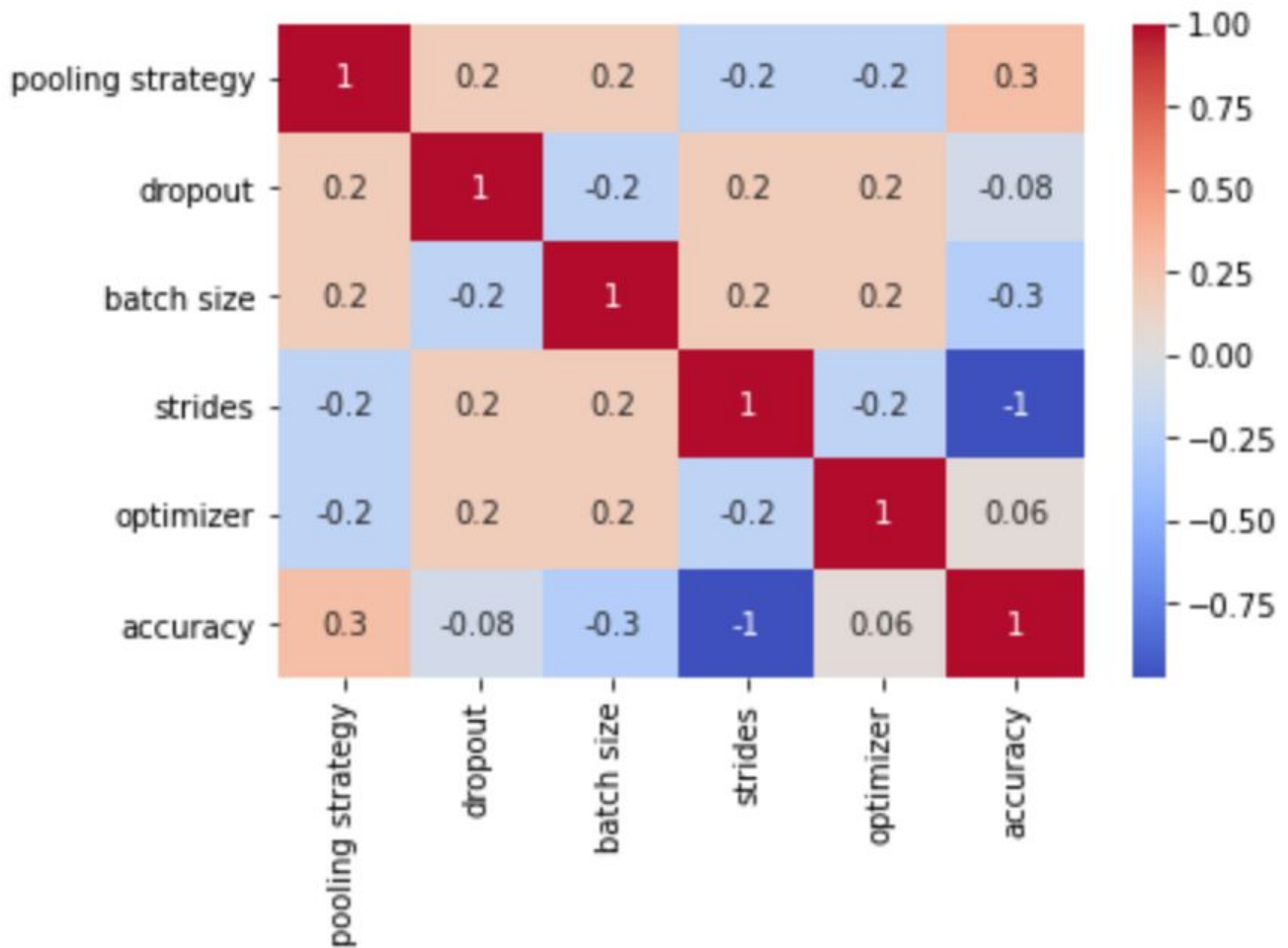
	Precision	Recall	Frequency in Training Set
<b>en</b>	0.952585	0.979613	18508
<b>ja</b>	0.985961	0.991929	10421
<b>ar</b>	0.988701	0.992439	2199
<b>ru</b>	0.937255	0.983539	978
<b>pl</b>	0.900000	0.692308	93
<b>vi</b>	1.000000	0.400000	16
<b>ja_LATN</b>	0	0	1
<b>fi</b>	0.750000	0.750000	15

- (1) Generally, for high frequency languages, such as ‘en’, ‘ja’, the precision, recall are better than those of low frequency languages such as ‘ja\_LATN’, ‘pl’, and ‘vi’ .
- (2) In addition, for languages with very unique characters, such as ‘fi’ and ‘ar’, although the frequency is low, the precision and recall are also good, because I used character indices to encode each tweet.

#### 4. Heatmap Correlation

I encode global max pooling as 0, global average pooling as 1; optimizer adam as 0 and optimizer sgd as 1, then plot the hyperparameter correlation heatmap (see below).

Figure 1. Hyperparameter Correlation Heatmap



From the heatmap, we can see:

- (1) Pooling strategy has positive correlation with drop out, batch size and accuracy; it has negative correlation with strides and optimizer. When the pooling strategy is the global average pooling instead of the global max pooling, the accuracy increases.
- (2) Drop out has positive correlation with pooling strategy, strides, optimizer; and has negative correlation with batch size and accuracy. When adding dropout instead of removing it, the test accuracy should decrease, but the coefficient is very small.
- (3) Batch size has positive correlation with strides and optimizer; negative correlation with accuracy.
- (4) Strides has negative correlation with optimizer and accuracy. When increase strides, the test accuracy decreases and the magnitude is relatively large.
- (5) Optimizer has positive correlation with accuracy, which means when optimizer is SGD, the test accuracy increases, but the magnitude is very small.