Deep learning for speech commands recognition

Author:
Thomas MARQUET

Supervisor:
Olivier MARCEAU

Contents

Section 1

Learning Task

1.1 Introduction

From our devices listening to you and waiting for you to call their names, to removing the language barrier between humans using automated translation, speech recognition is definitively one of the most growing technology. Being able to speak to a computer using the most natural communication vector of them all, will enhance many daily life applications. Many of us lives permanently with at least one computer more powerful than everything that we could think of 20 years ago, and this computer has access to almost infinite calculation power through the cloud with a low latency. Therefore the use of neural networks to enable robust human/computer communication through voice became possible on large scale and outclassed classical techniques.

1.2 Task description

This work is an introduction to speech recognition. We use a public dataset brought by Google called "Google speech commands v1". The task is the classification of 20 differents small words which could be commonly used to give commands to a computer. We'll compare the results of differents type and size of neural networks which are known to fit such tasks.

The words that we want to learn are the following : "yes" , "no" , "stop" , "go" , "down" , "up" , "left" , "right" , "on" , "off" , "zero" , "one" , "two" , "three" , "four", "five" , "six" , "seven" , "eight" , "nine".

We also want to recognize when there is an other word than those 20. To do so we'll use 10 additionnals words that will be labeled under "unknown". Those words are : "marvin", "sheila" , "tree" , "wow" , "bed" , "bird", "dog" , "cat" , "happy" , "house".

Finally, we also want to recognize when there is no word spoken. We'll add a silence label.

Section 2

Data analysis

2.1 Data's distribution

The dataset is composed of 70k .wav files which are already split (training set : 58252 , validation set : 6137 , test set : 6231). Those files are regrouped in 30 differents words that can be learned. You can also find 6 extra files of differents length which are just common noises.

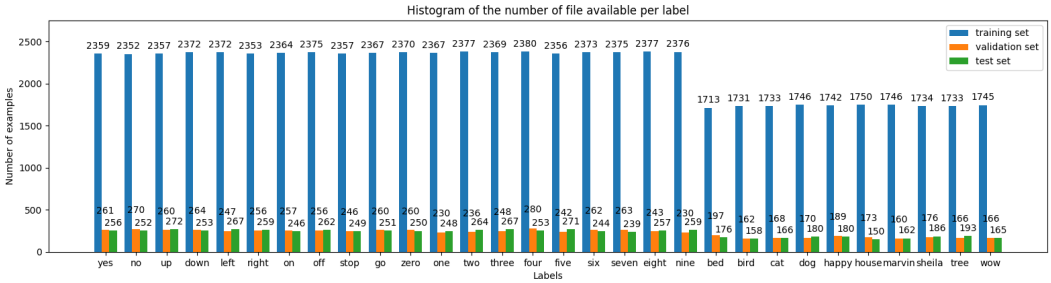


Figure 2.1: Files distributions

Each file is supposed to contain a one second clip recorded at a rate of 16kHz. Therefore we should expect 16000 points once the file is loaded, but plenty of them are actually not one second long. This will bring problems later on since we need the inputs to be always the same length. We can either expend those data to get the same length as the others (zero-padding, averaging, ...), we can cut the silences out of every file and take only the voices, or we can simply ignore them.

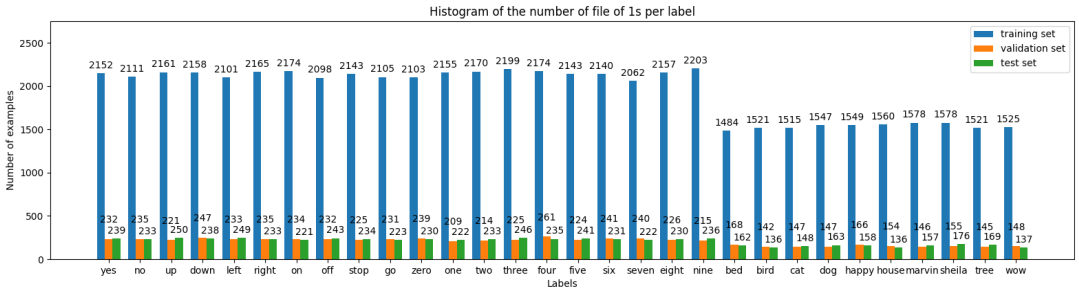


Figure 2.2: Files of 1s distributions

2.2 Raw data : Does it makes sense to use them ?

Even if audio raw files can be fed to a network and lead to a success while building really deep CNN structures, we're looking for lightweight networks and we have to highlights features to make the learning easier. Fortunately our data is relatively clean of any noise, and therefore we still can detect some patterns from the raw data.

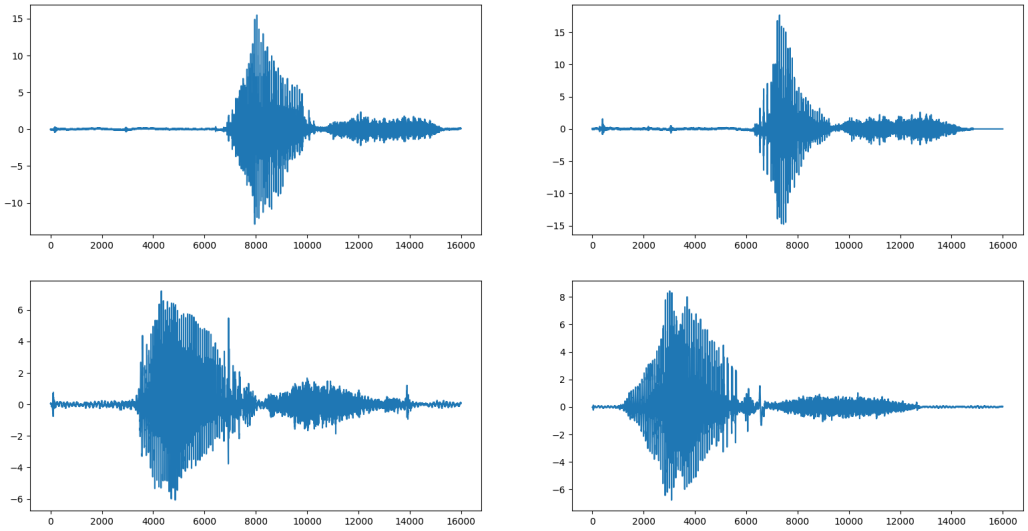


Figure 2.3: a few examples of raw data for the word yes

But as we can see below, amongst the others examples you will find some that are more or less out of the pattern.

2.3 Preprocessing data : Mel Frequency Cepstral Coefficients

The mfcc features are by far the most common way to preprocess audio data. The idea is to represent the power spectrum in a small subset of features over

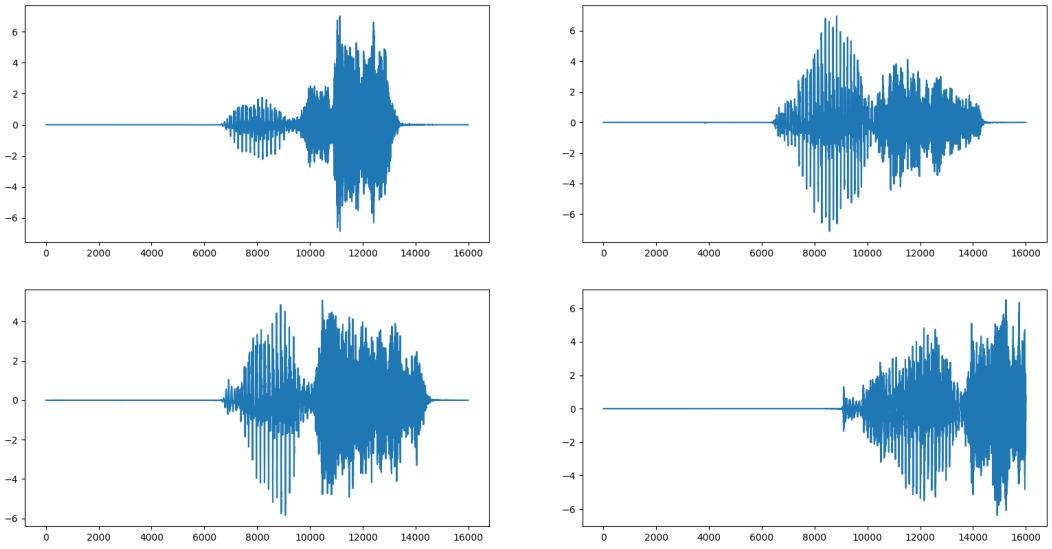


Figure 2.4: other examples of yes

time windows of the signal. The spectrum will be projected on a Mel-scale which is a way to represent the frequencies of a signal closer to how we hear sounds.

2.3.1 How do we calculate those coefficients ?

The method to calculate those coefficients has been taken from here.

- The first step is to make time windows of our signal. In our case we take 30ms time windows with a 10 ms step. Which makes us 98 frames.
- Then we apply on each frame the discrete fourier transform to obtain an estimate of the power spectrum
- Then we compute the Mel-scale filterbanks and apply it the power spectrum
- Then we take the log of all filters from the previous step
- Finally, apply the discrete cosine transform to those filters

This will yields an array of 98 x $n_{filters}$, $n_{filters}$ being an arbitrary chosen number between 10 and 40. Usually the number is 13.

2.3.2 Examples on the word yes

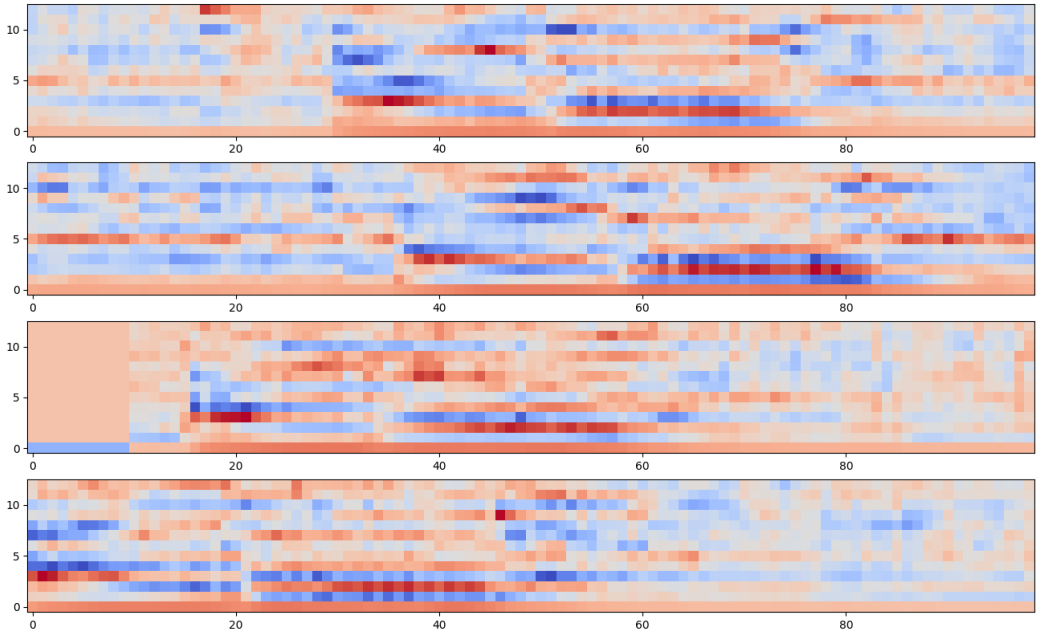


Figure 2.5: examples of MFCC features on the word yes

2.4 Preprocessing data : Spectral Subband centroids

To calculate those features, you've to divide the signal into n_sub subbands and take the centroid of each one. The centroid of a subband is the weighted mean of the frequencies in the signal weighted by their magnitude. The number of subbands is arbitrarly chosen (usually 26). You can see the full calculation here CITE SUBBAND.

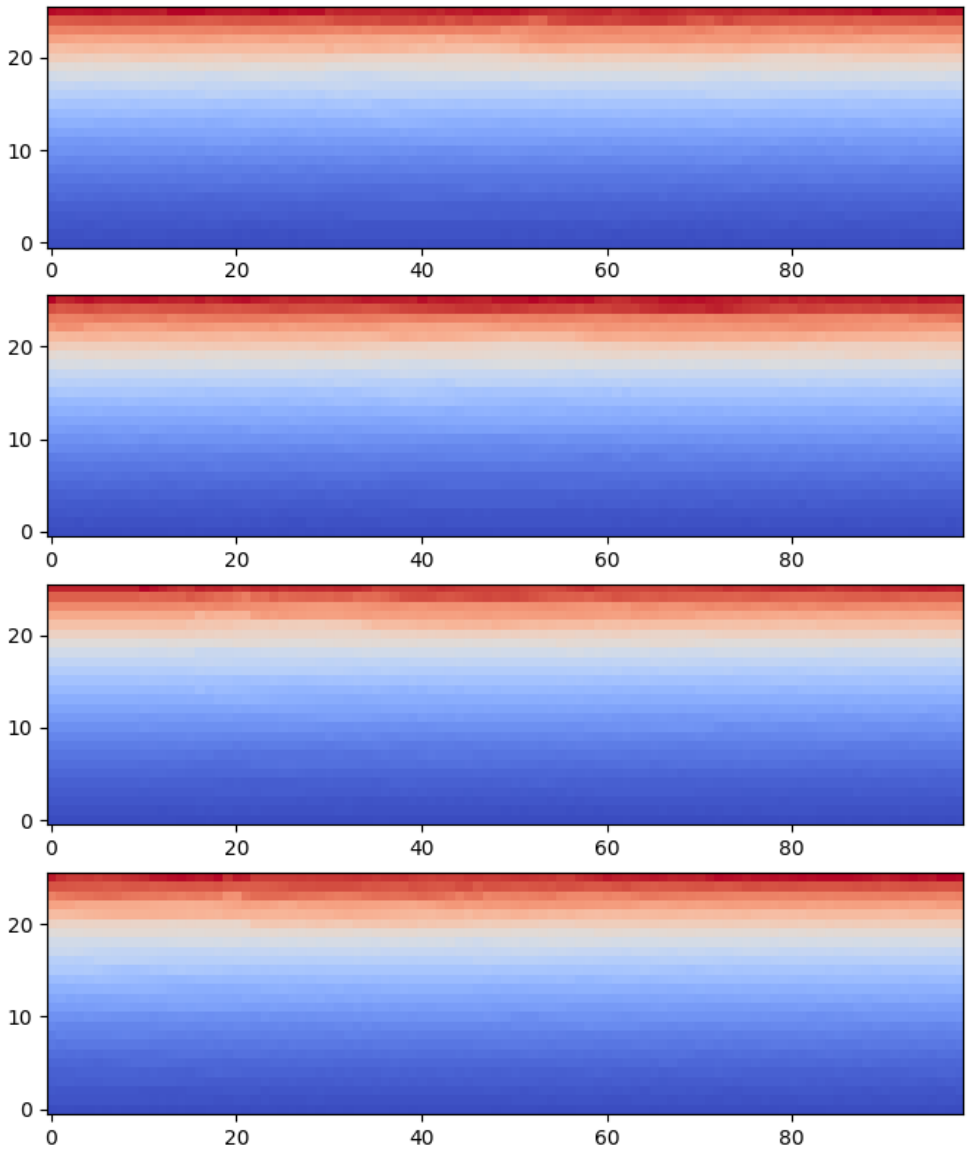


Figure 2.6: examples of SSC features on the word yes

Those features are extremely different from the mfcc and have been proven to be complementary CITE SUBBANDCOMPL.

Section 3

Related work

There are many types of neural networks and each of them perform differently according to the task. Considering the low vocabulary of our task and the large amount of data, we'll be looking to search for patterns through the identification of templates with neural networks. Therefore, CNN comes immediately to mind. But many works suggests the use of recurrent neural networks. Therefore we'll focus on those two types of networks that seems to be the most promising. Finally, MLPs are usually mentionned but not for their performances, mostly because they are easy to implement and it's always a nice comparison.

Keyword spotting task are extremely fit to CNNs according to many research CITE GOOGLE. A DNN cannot take advantage of the topology of the signal, however CNNs are extremely good at acoustic modeling which permits to give a good representation of a speech spectrum. The differences of frequencies between each speaker and also the timing differences between each examples are known to be easily handled by CNNs.

RNNs and especially LSTMs are also fit to this task because of their ability to take into account time sequences through feedback connections. They also benefits from the combination of lstm layers with deep or convolutionnal layers CITEALEX GRAVES. But one of the main problem encountered with RNNs, is that they are slow for many reasons (no parrellelization, slow learning, ...). Therefore the trend is to move away from recurrent structures.

Section 4

Implementation

4.1 Preprocessing

The preprocessing part is in the file 'preprocessing.py'. In this file you'll find all functions used to create sub - training, validation and testing lists and the functions that are used to process the data. Since it's better to have a uniform distribution of examples, we decided that we would take a fixed number of example for each label : 2060 examples per label for training, and 206 example per label for the validation. We then create 2266 "silences" for the training and validation. Those silences are created with the background noises given in the dataset.

4.1.1 Data

We use different preprocessing of the data in order to compare or combine them.

- Mel frequency cepstrum coefficients (MFCC)
- Spectral subband centroid (SSC)

Both mfcc and ssc features are obtained through a python library called python speech features. Then every kind of data is normalized to zero mean and unit variance. The data is then stored into a dictionnary which maps the data (value) to it's kind (key).

4.1.2 Labels

Labels are encoded as a list of length 22 with a one at the index of the right element and zeros otherwise.

4.2 Types of neural networks

To train our networks and implement them we use the tensorflow/keras library.

4.2.1 MLP

```
def create_model_mlp(data_type,layer_nb = 5,input_shape = (98,13), learning_rate = 0.00001,n_label = len(labels),dense_units=256):
    model = tf.keras.Sequential(name='mlp_'+ data_type)

    if data_type == 'mfcc':
        input_shape = (98,13)
    if data_type == 'ssc' :
        input_shape = (98,26)

    model.add(Dense(dense_units, name='fc1',input_shape = input_shape))
    model.add(BatchNormalization(name='block1_batchnorm'))
    model.add(tf.keras.layers.Activation('relu'))

    for i in range(2,layer_nb):
        model.add(Dense(dense_units, name='fc'+str(i)))
        model.add(BatchNormalization(name='block{}_batchnorm'.format(str(i))))
        model.add(tf.keras.layers.Activation('relu'))

    model.add(Flatten(name='flatten'))

    model.add(Dense(n_label, activation='softmax', name='predictions'))
    optimizer = RMSprop(lr=learning_rate)

    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    model.summary()
    return model
```

Figure 4.1: Mlp structure

4.2.2 CNN

Unlimited parameters CNN

This CNN is around 24M parameters and consists of 5 convolution block with batchnorm and maxpooling. Those 5 convolutions blocks uses 64, 128, 256, 512, 512 filters. Then it uses two denses layers of 4096 neurons with a dropout of 0.5 before each layer to prevent a large overfit.

```
# Convolution Blocks
# Block 1
model.add(Conv1D(64, 3, padding='same', name='block1_conv',input_shape = input_shape))
model.add(Lambda(lambda x: K.l2_normalize(x,axis=1)))
model.add(BatchNormalization(name='block1_batchnorm'))
model.add(tf.keras.layers.Activation('relu'))
model.add(MaxPooling1D(2, strides=2, name='block1_pool'))
```

Figure 4.2: Convolutions block

This makes a really heavy network that we used previously but usually able to learn whatever is asked of him. It's a good starting point. His size being around 200Mo is not extremely pratical for limited embedded devices and therefore we're looking for a smaller network.

Small CNN $<$ 250k parameters

The idea behind this network is to try out this trend that stacks convolutions with a lower amount of pooling in a network. To keep it small we uses small numbers of filters and only one dense layer with 200 neurons. No dropout needed because of it's small size, thus we don't waste learned information and we uses the batchnorm layers power to its fullest.

```
# End convolution

 model.add(Dropout(0.5,name = 'Dropout'))

 model.add(Flatten(name='flatten'))
 # Two Dense layers

 model.add(Dense(dense_units, name='fc1'))
 model.add(Lambda(lambda x: K.l2_normalize(x,axis=1)))
 model.add(BatchNormalization(name='block_batchnorm_dense1'))
 model.add(tf.keras.layers.Activation('relu'))

 model.add(Dropout(0.5))

 model.add(Dense(dense_units, name='fc2'))
 model.add(Lambda(lambda x: K.l2_normalize(x,axis=1)))
 model.add(BatchNormalization(name='block_batchnorm_dense2'))
 model.add(tf.keras.layers.Activation('relu'))

 model.add(Dense(n_label, activation='softmax', name='predictions'))

 optimizer = RMSprop(lr=learning_rate)
 model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
 model.summary()
 return model
```

Figure 4.3: Dense layers

```
def create_model_small_cnn(data_type,labels = len(labels) , learning_rate = 0.0001,dense_units = 200):
    if data_type == 'mfcc':
        input_shape = (98,13)
    if data_type == 'ssc' :
        input_shape = (98,26)

    model = tf.keras.Sequential(name='small_cnn_'+data_type)

    model.add(Input(input_shape))


    model.add(Conv1D(22, 3, padding='same', name='conv1'))
    model.add(BatchNormalization(name = 'batch_norm1'))
    model.add(tf.keras.layers.Activation('relu'))

    model.add(Conv1D(44, 3, padding='same', name='conv2'))
    model.add(BatchNormalization(name = 'batch_norm2'))
    model.add(tf.keras.layers.Activation('relu'))

    model.add(Conv1D(22, 3, padding='same', name='conv3'))
    model.add(BatchNormalization(name = 'batch_norm3'))
    model.add(tf.keras.layers.Activation('relu'))
    model.add(AveragePooling1D(2, strides=2, name='pooling'))


    model.add(Flatten(name = 'flatten'))


    model.add(Dense(dense_units, name='dense'))
    model.add(BatchNormalization(name='batch_norm_dense'))
    model.add(tf.keras.layers.Activation('relu'))

    model.add(Dense(labels, activation='softmax',name = 'output'))
    optimizer = RMSprop(lr=learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    model.summary()
    return model
```

Figure 4.4: Small CNN structure

4.2.3 LSTM

LSTM can be implemented with a cnn stacked on it therefore the following implementation yields two structure, one is a pure lstm and the other one is a lstm-cnn. The

```
def create_model_lstm(data_type,cnn =False,layer_nb  = 5,classes = len(labels),lstm_nodes=64 , learning_rate = 0.0001):


    if data_type == 'mfcc':
        input_shape = (98,13) if not cnn else (98,13,1)
    if data_type == 'ssc' :
        input_shape = (98,26) if not cnn else (98,26,1)

    model = Sequential(name = 'lstm_{}'.format('' if not cnn else 'cnn_')+ data_type)
    model.add(Input(shape = input_shape))

    if cnn:
        cnn = Sequential(name= 'cnn_entry_'+data_type)

        cnn.add(Conv1D(22, 3, padding='same', name='conv1'))
        cnn.add(BatchNormalization(name = 'batch_norm1'))
        cnn.add(tf.keras.layers.Activation('relu'))

        cnn.add(Conv1D(44, 3, padding='same', name='conv2'))
        cnn.add(BatchNormalization(name = 'batch_norm2'))
        cnn.add(tf.keras.layers.Activation('relu'))

        cnn.add(Conv1D(22, 3, padding='same', name='conv3'))
        cnn.add(BatchNormalization(name = 'batch_norm3'))
        cnn.add(tf.keras.layers.Activation('relu'))
        cnn.add(AveragePooling1D(2, strides=2, name='pooling'))

        model.add(TimeDistributed(cnn))
        model.add(Reshape((98,-1)))


    if layer_nb == 1:
        model.add(LSTM(lstm_nodes,name = 'lstm_entry'))

    else:
        model.add(LSTM(lstm_nodes, return_sequences=True,name = 'lstm_entry'))
        for i in range(2, layer_nb):

            model.add(LSTM(lstm_nodes, return_sequences=True,name = 'lstm_{}'.format(i)))
        model.add(LSTM(lstm_nodes,name = 'lstm_out'))

    model.add(Flatten())
    model.add(Dense(classes, activation='softmax', name='predictions'))

    optimizer = RMSprop(lr=learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    model.summary()
    return model
```

Figure 4.5: Lstm structure

Section 5

Results

Model	Accuracy %
DenseNet-121 best	85.52
ConvNet	85.4
Attention RNN best	94.5
Kaggle winner	91.06

Table 5.1: Public Accuracy on the Google Data dataset v1 20words

Those results have been achieve on differents subsets of the testing list that we have. The testing list that i have is the combination of the public test set for the kaggle challenge and the private test set. Therefore the accuracy achieved by the kaggle winner is only on the private test set. The others have been tested on the original test set which is the public and private test set of the kaggle challenge. But the label "silence" hasn't been added to the learning task. We have one more label than them.

5.1 Test definition and metrics

To do the test we use the original testing list provided by the author of the data set which is around 6000 files. We add also 200 silences that are obtained with various chunks of the background files, which are'nt taken into account in the tests sets of the figure 5.1.

In order to judge our networks, we use 4 metrics.

Let t_p, f_p, t_n, f_n be the number of true positives, false positives, true negatives and false negatives. We define our metrics the following way :

- Precision : $P = \frac{t_p}{t_p+f_p}$
- Recall : $R = \frac{t_p}{t_p+f_n}$
- False positive rate : $Fpr = \frac{f_p}{t_p+f_p+t_n+f_n}$

- Accuracy : $A = \frac{t_p+t_n}{t_p+f_p+t_n+f_n}$

5.2 General results

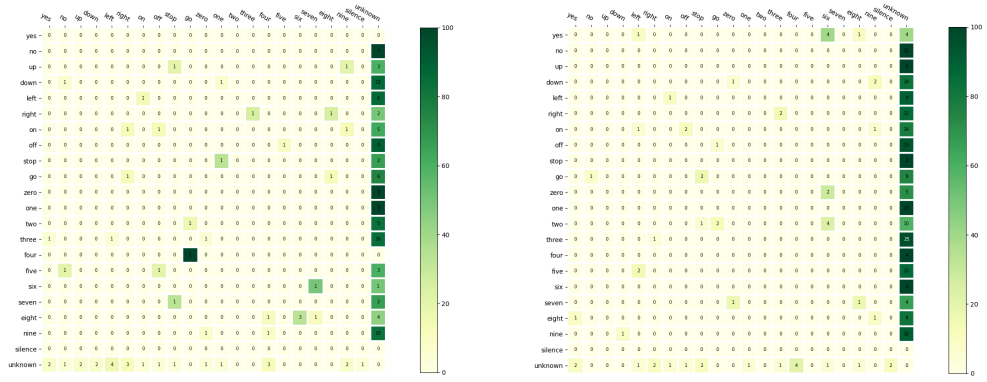
	MLP	CNN_{250k}	CNN	$LSTM_{cnn}$	$LSTM$
Success rate MFCC	97.58	98.54	99.33	97.57	97.37
Success rate SSC	96.19	92.75	97.17	96.33	28.20

Table 5.2: Success rates for all models and preprocessing

As expected the most complex network as the best accuracy in both preprocessing. Also MFCC preprocessing wins everytime and SSC preprocessing seems to not work at all on pure lstms layers.

5.3 MLP

As we can see in the table 1 and in the heatmap below, that most of the mistakes made by the MFCC preprocessing are coming from "down", "three", "eight", "nine" and "off". "down" and "three" are likely to be confused with "dog" and "tree" in the unknown set. "eight" is quite often confused with four. "nine" and "off" also with the unknown set but with no clear antagonist.



(a) MFCC preprocessing (b) SSC preprocessing

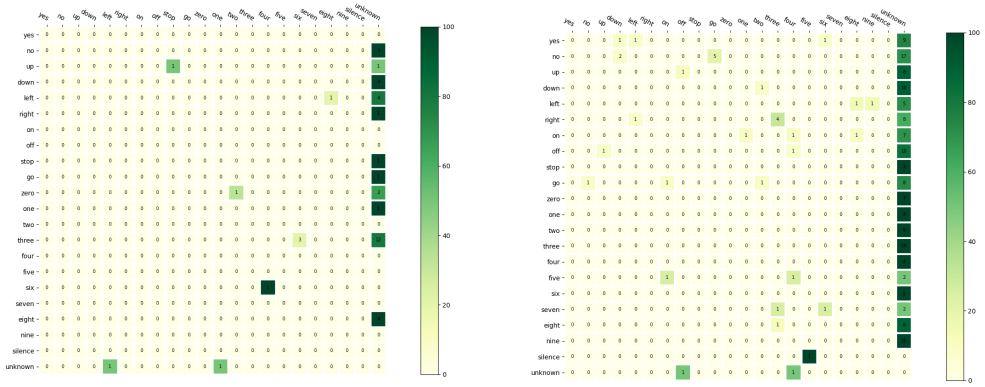
Figure 5.1: Confusion matrix for MLPs with different preprocessing

What we can see is that in general the SSC preprocessing has a classify a lot of out of the ordinary examples in the unknown label. So does the MFCC but with more success.

5.4 CNN

From this table 2 we can see that most of the labels are almost always recognized perfectly with the mfcc preprocessing, but again we see some problems for "three"

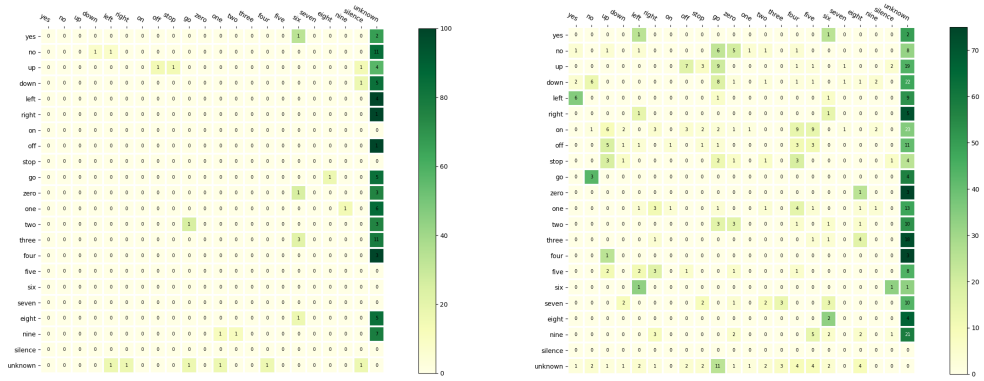
and "eight". It's even clearer when we look at the heatmap. There is a much higher confusion for "three" than for the others.



(a) MFCC preprocessing (b) SSC preprocessing

Figure 5.2: Confusion matrix for CNNs with different preprocessing

Results for the smaller models are in this table 3 .With a smaller model, we can see that "three" is still a problem, so is "nine" but we have a new problematic label emerging which is "no"



(a) MFCC preprocessing (b) SSC preprocessing

Figure 5.3: Confusion matrix for small CNNs with different preprocessing

5.5 LSTM

In lstm we can see that the ssc preprocessing is not giving any results. Looks like the structure is not able to learn anything. We didn't plot the heatmap for the ssc preprocessing because there was no point. It's just not accurate at all.

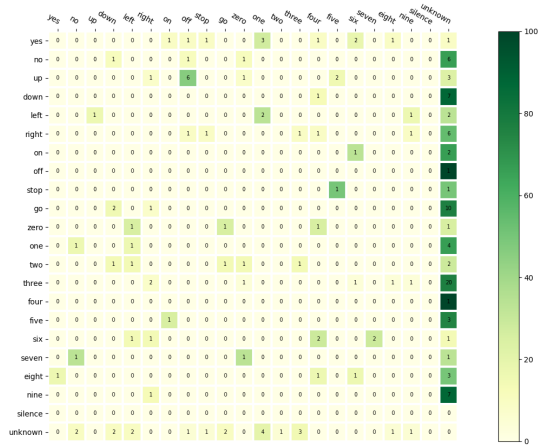
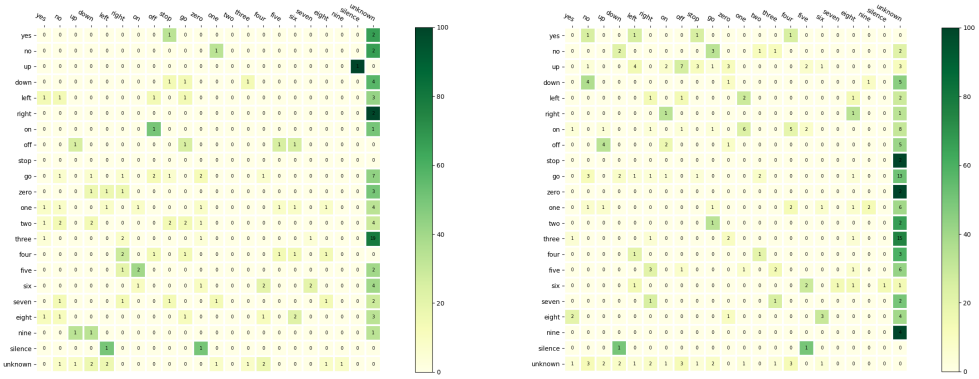


Figure 5.4: Confusions matrix for LSTM and MFCC

What we can see on the other hand is that the confusions looks like more distributed over the labels. Where usually the only confusions were giving a false unknown label, we have less false positives for the unknown label and more conufusion between regular labels. Still we have a few labels with really inaccurate results like "three".



(a) MFCC preprocessing (b) SSC preprocessing

Figure 5.5: Confusion matrix for LSTM CNN with different preprocessing

Same results than with the pure lstm in term of distributions of inaccuracies. But we solved the issue of the ssc preprocessing by adding a convolutionnal layer before hand.

5.6 Conclusion

To conclude, we have a few problematic labels, mostly "three". It makes sense since "tree" is present as an unknown word. The differenciation between those two words in the english language is mostly done with the context of the phrase. Here, there is no context. We think that a human would make a significant amount of mistakes if he tried to do the same. We can also see that CNNs and MLPs are classifying more often a word as an unkwown word where lstm are making more mistakes by classifying one known label with another. It would mean that for the CNNs and MLPs, the "templates" created when the model learns are more strict. Therefore when one example is out of the ordinary, it's is classified as an unknown word. While for LSTMs, those templates are more flexible, thus rejecting less words but making more mistakes. We believe that if we were to use one of this network in practice, we would choose a CNN. Because it's better to think that something is unknown than thinking something is true while it's not.

Section 6

Conclusion

In this work we tried different structures and preprocessing to achieve the classification of some one word commands. We saw that for our models a convolution is needed to achieve good performances. The best model is the most complex CNN but closely followed but the LSTM with a convolution and a smaller CNN, which are both networks with under 250k parameters. The MFCC preprocessing is clearly the winner as expected. We achieve a much better accuracy than the one that we've been able to find but we have an additional label that has 100% accuracy. Even without it though, the accuracy is better. During this work we've been through an idea called ensemble machine learning and even if we didn't demonstrated here we found some interesting results during live tests. Ensemble machine learning make more robust to input twists predictions if the networks that are combined do not learn the same thing. But since the idea here was to keep the footprint low, we decided not to use it.

Annexes

	FPR	R	A	P	FPR	R	A	P
	MFCC				SSC			
yes	0.00	0.99	0.99	1.00	0.04	0.99	0.97	0.96
no	0.03	0.99	0.98	0.97	0.05	1.00	0.97	0.95
up	0.02	0.99	0.99	0.98	0.02	1.00	0.99	0.98
down	0.06	0.99	0.97	0.94	0.07	1.00	0.96	0.93
left	0.03	0.98	0.98	0.97	0.04	0.98	0.97	0.96
right	0.02	0.98	0.98	0.98	0.06	0.99	0.96	0.94
on	0.04	0.99	0.98	0.96	0.08	0.99	0.96	0.92
off	0.04	0.99	0.97	0.96	0.06	0.99	0.96	0.94
stop	0.01	0.99	0.99	0.99	0.01	0.98	0.98	0.99
go	0.03	0.99	0.98	0.97	0.05	0.99	0.97	0.95
zero	0.01	0.99	0.99	0.99	0.03	0.99	0.98	0.97
one	0.01	0.99	0.99	0.99	0.04	1.00	0.98	0.96
two	0.03	1.00	0.99	0.97	0.07	1.00	0.96	0.93
three	0.07	1.00	0.96	0.93	0.10	0.99	0.94	0.90
four	0.00	0.98	0.99	1.00	0.02	0.98	0.98	0.98
five	0.02	1.00	0.99	0.98	0.05	1.00	0.97	0.95
six	0.01	0.99	0.99	0.99	0.02	0.95	0.97	0.98
seven	0.01	0.99	0.99	0.99	0.03	1.00	0.99	0.97
eight	0.04	0.99	0.98	0.96	0.05	0.99	0.97	0.95
nine	0.05	0.98	0.97	0.95	0.05	0.98	0.97	0.95
silence	0.00	0.99	1.00	1.00	0.00	0.99	0.99	1.00
unknown	0.02	0.94	0.96	0.98	0.01	0.88	0.93	0.99

Figure 1: Metrics for each label with MLP

	FPR	R	A	P	FPR	R	A	P
	MFCC				SSC			
yes	0.00	1.00	1.00	1.00	0.05	1.00	0.98	0.95
no	0.01	1.00	0.99	0.99	0.09	1.00	0.95	0.91
up	0.01	1.00	1.00	0.99	0.03	1.00	0.98	0.97
down	0.02	1.00	0.99	0.98	0.04	0.99	0.97	0.96
left	0.02	1.00	0.99	0.98	0.03	0.99	0.98	0.97
right	0.00	1.00	1.00	1.00	0.05	1.00	0.97	0.95
on	0.00	1.00	1.00	1.00	0.04	0.99	0.97	0.96
off	0.00	1.00	1.00	1.00	0.05	0.99	0.97	0.95
stop	0.00	1.00	1.00	1.00	0.01	1.00	0.99	0.99
go	0.00	1.00	1.00	1.00	0.05	0.98	0.96	0.95
zero	0.01	1.00	0.99	0.99	0.03	1.00	0.99	0.97
one	0.00	1.00	1.00	1.00	0.02	1.00	0.99	0.98
two	0.00	1.00	1.00	1.00	0.02	0.99	0.99	0.98
three	0.06	1.00	0.97	0.94	0.06	0.98	0.96	0.94
four	0.00	1.00	1.00	1.00	0.02	0.98	0.98	0.98
five	0.00	1.00	1.00	1.00	0.02	1.00	0.99	0.98
six	0.00	0.99	0.99	1.00	0.00	0.99	0.99	1.00
seven	0.00	1.00	1.00	1.00	0.02	1.00	0.99	0.98
eight	0.02	1.00	0.99	0.98	0.04	0.99	0.98	0.96
nine	0.00	1.00	1.00	1.00	0.06	1.00	0.97	0.94
silence	0.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00
unknown	0.00	0.98	0.99	1.00	0.00	0.90	0.95	1.00

Figure 2: Metrics for each label with CNN

	FPR	R	A	P	FPR	R	A	P
	MFCC				SSC			
yes	0.01	1.00	0.99	0.99	0.02	0.96	0.97	0.98
no	0.05	1.00	0.97	0.95	0.10	0.95	0.92	0.90
up	0.03	1.00	0.99	0.97	0.16	0.92	0.88	0.84
down	0.02	1.00	0.99	0.98	0.17	0.97	0.90	0.83
left	0.02	0.99	0.99	0.98	0.07	0.96	0.95	0.93
right	0.00	1.00	1.00	1.00	0.03	0.94	0.95	0.97
on	0.00	1.00	1.00	1.00	0.23	0.99	0.87	0.77
off	0.00	1.00	1.00	1.00	0.11	0.95	0.92	0.89
stop	0.00	1.00	1.00	1.00	0.07	0.96	0.95	0.93
go	0.03	0.99	0.98	0.97	0.04	0.80	0.88	0.96
zero	0.02	1.00	0.99	0.98	0.02	0.93	0.96	0.98
one	0.03	0.99	0.98	0.97	0.11	0.99	0.94	0.89
two	0.02	1.00	0.99	0.98	0.08	0.97	0.94	0.92
three	0.05	1.00	0.97	0.95	0.09	0.98	0.94	0.91
four	0.01	1.00	0.99	0.99	0.02	0.88	0.93	0.98
five	0.00	1.00	1.00	1.00	0.08	0.90	0.91	0.92
six	0.00	0.97	0.99	1.00	0.01	0.94	0.96	0.99
seven	0.00	1.00	1.00	1.00	0.10	0.99	0.94	0.90
eight	0.03	1.00	0.98	0.97	0.03	0.94	0.96	0.97
nine	0.04	1.00	0.98	0.96	0.13	0.98	0.92	0.87
silence	0.00	0.98	0.99	1.00	0.00	0.97	0.99	1.00
unknown	0.00	0.95	0.97	1.00	0.03	0.87	0.92	0.97

Figure 3: Metrics for each label with a small CNN

	FPR	R	A	P	FPR	R	A	P
	MFCC				SSC			
yes	0.04	1.00	0.98	0.96	0.76	0.24	0.24	0.24
no	0.04	0.98	0.97	0.96	0.79	0.10	0.14	0.21
up	0.05	1.00	0.97	0.95	0.86	0.05	0.07	0.14
down	0.03	0.97	0.97	0.97	0.74	0.13	0.18	0.26
left	0.02	0.98	0.98	0.98	0.84	0.02	0.03	0.16
right	0.05	0.97	0.96	0.95	0.82	0.16	0.17	0.18
on	0.01	0.99	0.99	0.99	0.82	0.15	0.16	0.18
off	0.00	0.96	0.98	1.00	0.67	0.16	0.21	0.33
stop	0.01	0.99	0.99	0.99	0.75	0.36	0.29	0.25
go	0.06	0.98	0.96	0.94	0.83	0.06	0.09	0.17
zero	0.02	0.98	0.98	0.98	0.77	0.50	0.32	0.23
one	0.03	0.96	0.97	0.97	0.80	0.09	0.12	0.20
two	0.03	1.00	0.98	0.97	0.74	0.15	0.19	0.26
three	0.10	0.98	0.94	0.90	0.76	0.17	0.20	0.24
four	0.00	0.97	0.98	1.00	0.80	0.13	0.15	0.20
five	0.02	0.99	0.99	0.98	0.85	0.20	0.17	0.15
six	0.03	0.98	0.97	0.97	0.67	0.36	0.34	0.33
seven	0.01	0.99	0.99	0.99	0.81	0.07	0.10	0.19
eight	0.03	0.99	0.98	0.97	0.73	0.09	0.13	0.27
nine	0.03	0.98	0.97	0.97	0.78	0.17	0.19	0.22
silence	0.00	1.00	1.00	1.00	0.29	0.95	0.81	0.71
unknown	0.01	0.95	0.97	0.99	0.69	0.55	0.40	0.31

Figure 4: Metrics for LSTM

	FPR	R	A	P	FPR	R	A	P
	MFCC				SSC			
yes	0.01	0.98	0.98	0.99	0.02	0.98	0.98	0.98
no	0.01	0.97	0.98	0.99	0.04	0.94	0.95	0.96
up	0.00	0.99	0.99	1.00	0.10	0.97	0.93	0.90
down	0.03	0.97	0.97	0.97	0.05	0.97	0.96	0.95
left	0.03	0.98	0.98	0.97	0.03	0.96	0.97	0.97
right	0.01	0.97	0.98	0.99	0.01	0.96	0.97	0.99
on	0.01	0.98	0.99	0.99	0.11	0.97	0.93	0.89
off	0.02	0.98	0.98	0.98	0.05	0.95	0.95	0.95
stop	0.00	0.97	0.99	1.00	0.01	0.97	0.98	0.99
go	0.07	0.97	0.95	0.93	0.10	0.96	0.93	0.90
zero	0.03	0.97	0.97	0.97	0.01	0.97	0.98	0.99
one	0.05	0.99	0.97	0.95	0.07	0.95	0.94	0.93
two	0.06	1.00	0.97	0.94	0.01	0.98	0.98	0.99
three	0.09	0.99	0.95	0.91	0.08	0.98	0.95	0.92
four	0.03	0.97	0.97	0.97	0.02	0.95	0.97	0.98
five	0.02	0.99	0.98	0.98	0.06	0.97	0.96	0.94
six	0.04	0.98	0.97	0.96	0.03	0.97	0.97	0.97
seven	0.03	0.99	0.98	0.97	0.02	1.00	0.99	0.98
eight	0.04	0.98	0.97	0.96	0.04	0.97	0.96	0.96
nine	0.01	1.00	0.99	0.99	0.02	0.99	0.99	0.98
silence	0.01	0.99	0.99	0.99	0.01	0.99	0.99	0.99
unknown	0.01	0.96	0.98	0.99	0.02	0.94	0.96	0.98

Figure 5: Metrics for LSTM + CNN