Speech Recognition

Gevel, N. J. (van de) 0890583   F.S. Straver 1362968   D. E. Matos Rodriguez 1352822   M.J.J. Nijssen 0961322

T.J. Schoonbeek 0904084

Eindhoven University of Technology

1. Introduction

For independent creators and entrepreneurs, building a simple speech detector with free data is difficult. This is due to the data pre-processing required before neural networks can be built on them. For this reason, Kaggle set up a competition on speech recognition [2]. Kaggle is an online community of data scientists and machine learners, owned by Google. Kaggle is mostly known for the competitions hosted on its website. One of these challenges is the TensorFlow Speech Recognition Challenge, hosted by Google one year after releasing Speech Commands Data-sets. This data set consists of 65,000 one-second long audio files of 30 short words, pronounced by thousands of different people. The challenge is to give labels to audio files in a test set. This set contains 10 word classes, namely: ‘yes’, ‘no’, ‘up’, ‘down’, ‘left’, ‘right’, ‘on’, ‘off’, ‘stop’, ‘go’. Next to these, there is an ‘unknown’ class, covering any word that is not any of the 10 aforementioned. Finally, there is a ‘silent’ class for ’no speech’ background audio files. This results in a multi-class classification problem with 12 classes.

The network architecture is covered in Section 2. Section 3 covers training procedure. The results of the complete network are discussed in Section 4, together with a conclusion.

2. Model

Tensorflow is used to create the model. Tensorflow uses low-level API that helps in understanding the network and the training process at a lower level. The downside is that writing out the code for training the network takes longer, but debugging is made easier as more details of the network can be controlled.

2.1. Pre-processing

Google Colab is used to create and train the model [1]. Before an audio file is fed through the network, it is transformed into a spectrogram. The spectrogram is a 128x128 image showing the time and frequency of the audio file, as can be seen in Fig. 1. The audio files were transformed into a spectrogram image, so in that way, it can be treated as a multi-class classification problem. Three data-sets are created. The train data-set contains 12120 .wav files, the same amount from every class (1000 .wav files), except the unknown class, which has 1120 .wav files. The test and validation data-sets contain 1200 files, with the same amount for every class (100 files). All the data-sets are randomly shuffled for better regularization.

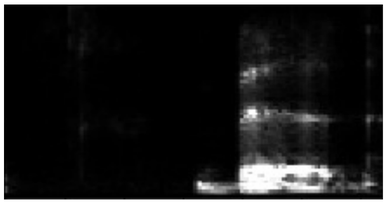
Figure 1. A spectrogram created from an audio clip of the data-set.

2.2. Input pipeline

Tensorflow Dataset Class is used to create the input pipeline. This Class grabs a text file with a list of all the audio files that will be analyzed. In addition, a method will batch as many audio files are needed from that given text file and convert this input audio files into Tensors. Once the spectrograms are batched up, the labels are extracted. The file names from the batch are fed through a function that encodes the labels. The last 17 characters of the file names are removed. The remaining characters are the class of the audio file. These classes are one hot encoded and batched with the same size as the spectrograms.

2.3. Network

The proposed neural network consists of four convolutional layers, with 3 x 3 kernel size. After every convolutional layer, there is a max pooling layer (kernel size 2 x2, which reduces the input by a factor of 2), except for the last pooling layer. The last pooling layer has two straight

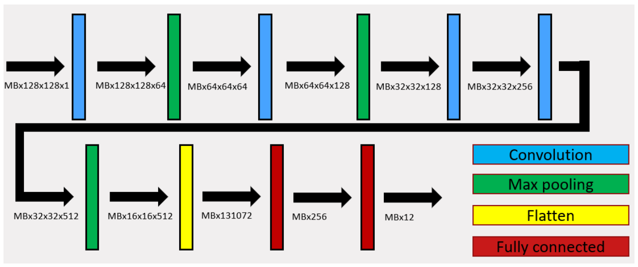

Figure 2. Architecture of our model [4].

convolutional layers. In order to get the proper shape for the classification problem, two fully connected layers are used, which makes the network have 12 outputs, each for one class tested. As a variation for our network, dropout is also used after the first fully connected layer. The number of parameters can be seen in Fig. 2.

The architecture of the model is shown in Fig. 2, including all tensor sizes. This network architecture was chosen empirically (see [source discussion thread]) to be of sufficient complexity, to prevent under- and over-fitting and provide good generalization.

3. Training

3.1. Loss function

The goal of the speech recognition challenge is to predict the word that corresponds to the input audio signal. The network outputs twelve values, where the highest value of the twelve is the predicted label. Only the highest of the twelve outputs will be selected by the model. For example, the audio file can be ’bed’ or ’no’, but not both. To get this typical output, the softmax cross entropy loss is used. This loss function measures the performance of the classification, which is a probability between 0 and 1 due to the softmax. The cross-entropy is the predicted probability from the actual label. So if the actual label is 1 and the prediction is 0.95, the model has a small loss. Finally, the argmax is taken over the network output to get the predicted, one hot encoded label.

3.2. Optimizer

The model uses the Adam optimizer is used. Adam is an improvement to the stochastic gradient descent method, where all the weights are updated according to one learning rate. Adam combines the advantages of AdaGrad and RMSProp [**?**]. Adam uses a learning rate for individual parameters based on L2 norm of previous gradients, such as in AdaGrad, but also uses a learning rate for individual parameters based on exponentially weights moving average of the previous gradients, just as in RMSprop. In summary, Adam calculates both the exponential moving average of gradients and the squared gradients and uses a bias update rule to control the decay of these moving averages. Fig. 3 shows the three different optimizers tested on the created model for Kaggle speech recognition challenge. It can be concluded that as mentioned in [**?**], the Adam optimizer shows the best performance.

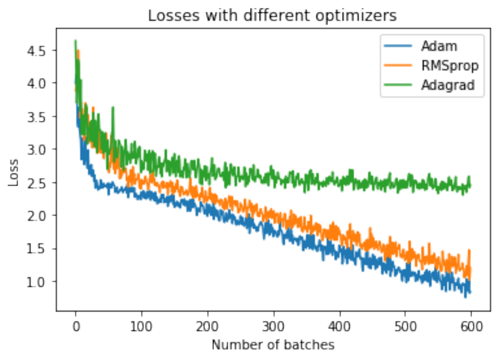

Figure 3. Loss as a function of number of batches for different optimizers.

3.3. Dropout

Dropout randomly sets some nodes in the network to zero during train time, resulting in a slightly smaller network. This prevents the network from overfitting on certain features since these features might be deactivated, which results in a model that is more generalized.

3.4. Composition of training, validating and testing

When training, three subsets are made. A train set, a validation set and a test set. The train set was used for training, such that the weights could be changed due to the back-flowing gradient coupled to the loss and optimization function. The validation set was used to tune the hyper-parameters for training. Hyper-parameters like the batch size and learning rate, were changed multiple times to receive the best result. The test set is an unseen set that is used to retrieve the actual accuracy of the model, that can be compared to getting an actual human speaking a word into the model.

4. Results

The following steps were executed to check the performance of the network:

1. Train on 0.1% of the training data to check if the network is able to over-fit on its training data.

2. Train for different experiments to determine best learning rate, batch size, epochs and dropout. The network is evaluated on the validation data-set during this step.

3. Train on the best settings and evaluate the performance on the test data-set.

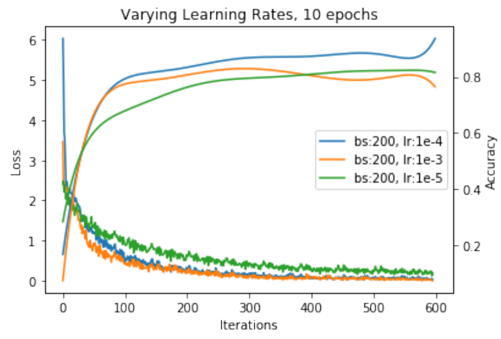

Figure 4. Loss and validation accuracy for different learning rates.

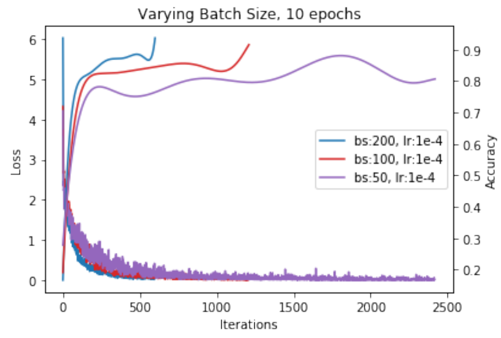

Figure 5. Loss and validation accuracy for different batch sizes.

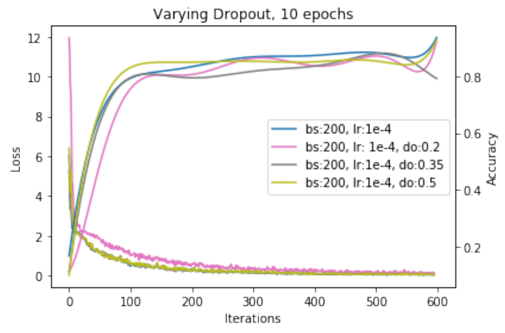

Figure 6. Loss and validation accuracy for different dropout rates.

4.1. Over-fit on training data

The network is trained on a sub-set of the training data to validate if it is capable of learning. An experiment is done where the loss converges to 0 and the accuracy to 100%, in other words, the network is able to over-fit and learn the training data.

4.2. Hyper-parameters experiments

The experiments that are performed can be seen in Table 1. These are run for 10 epochs. Experiment 1, 2 and 3 are done to find the best learning rate. As can be seen in Fig. 4, the best validation accuracy is found with a learning rate of $1e^{-4}$. Experiment 1, 4 and 5 are done to find a suiting batch size. From Fig. 5 it can be seen that a batch size of 200 results in the highest accuracy. Experiment 6, 7 and 8 are done to investigate if dropout has an impact of the performance. From Fig. 6 it can be seen that dropout does not have much impact so it is not used. Running the optimal hyper-parameters gives a test accuracy of 84%.

Experiment	Learning rate	Batch size	Dropout
1	$1e^{-4}$	200	-
2	$1e^{-3}$	200	-
3	$1e^{-5}$	200	-
4	$1e^{-4}$	100	-
5	$1e^{-4}$	50	-
6	$1e^{-4}$	200	0.2
7	$1e^{-4}$	200	0.35
8	$1e^{-4}$	200	0.5
Optimal	$1e^{-4}$	200	-

Table 1. Experiments

5. Conclusion

The created model runs best with the Adam optimizer with the following hyper-parameters: learning rate = 0.0001, batch size = 200 and no dropout. The final test accuracy is 84%, with only a training time of 20 minutes and 12120 training images.

6. Discussion

Instead of linearly sampling the hyper-parameter space (as described in Section 4.2), random sampling of the hyper-parameter space could have been done. This could result in finding better hyper-parameters since there is little to no relationship between the hyper-parameters due to the high dimensionality of the loss landscape. Due to time and hardware limitations, this was not done, however, it could have resulted in a more optimal set of hyper-parameters.

On the forum of the Kaggle competition, there is a file with bad training samples. The performance can be improved by removing these bad files. Training on the complete data-set of 65.000 files could improve the accuracy as well.

Different regularization techniques can be used to improve performance. Dropout did not have much influence. Adding dropout in the convolutional layers, instead of only in the fully-connected layer might help [3].

References

[1] Welcome To Colaboratory - Colaboratory. 1

[2] TensorFlow Speech Recognition Challenge — Kaggle, 2018. 1

[3] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. 11 2015. 3

[4] T. N. Sainath and C. Parada. Convolutional Neural Networks for Small-footprint Keyword Spotting. Technical report. 2