Attention Based Models for Keyword Spotting

Human Data Analytics project report - 06/09/2021

Riccardo Mazzieri

Abstract—**Today, Keyword Spotting (KWS) systems can be found in almost any device, ranging from smartphones, smart home devices or modern cars. They require real-time interaction and a high accuracy in order to function smoothly, even in heavily resource-costrained devices: for this reason research in KWS tries to engineer systems able to provide a good balance in the performance/lightness tradeoff. In this work we explore a variety of deep neural newtork architectures, with particular focus on the attention mechanism, for KWS. Indeed, in recent years attention based models have proven to be successful in a wide variety of domains, as well as being inherently interpretable. Motivated by this trend, we propose a variety of attention based architectures, taking the Att-RNN model, introduced by De Andreade et al. [1], as a baseline. We find that models based on multi head attention layers perform slightly better, even if this comes at the cost of an increased memory footprint.**

Index Terms—**Keyword Spotting, Convolutional Neural Networks, Recurrent Neural Networks, Attention Mechanism, Transformers.**

I. Introduction

The Keyword Spotting (KWS) task consists in the detection of a certain predetermined set of keywords from a stream of user utterances. In recent years, this problem has become increasingly popular and, with the rapid development of mobile devices, it is now playing an important role in human-computer interaction, as well as encouraging the adoption of hands-free interfaces for a wide variety of use cases, that can range from "smart home" devices like Amazon Alexa, to virtual assistants such as Apple's Siri or Google Assistant. KWS systems are typically required to continuously listen to user inputs on mobile devices, which are highly constrained in their memory and compute capabilities: this restriction has encouraged the development of systems able to achieve highly accurate results but still maintaining a small memory and computational footprint.

At present, Deep Learning (DL) techniques represent the state of the art approach for the KWS problem and have proved to give good results on the tradeoff between model lightness and model performance [2][3][4]. In the literature, a lot of DL models have been presented, ranging from simple fully connected feed forward networks [2], to both shallow and very deep Convolutional Neural Networks (CNN) [5][3] [6][7]. Especially in recent years, mostly thanks to the work by Vaswani et al. [8], interest towards attention-based architectures has grown dramatically. Indeed, recent works in machine learning have shown that models incorporating attention are able to provide groundbreaking results in a wide variety of domains [9][10][11][12][13]. Another charming feature of attention-based systems is their inherent interpretability: by visualizing the attention weights, one can directly see to which part of a specific input the model was paying attention when performing inference tasks. Indeed, Explainable AI (XAI) is quickly becoming an hot topic in modern machine learning research, and the development of models capable of giving some sort of explanation for their predictions will in time become more and more desirable, if not required [14].

Motivated by those increasing trends, in this work we first give a review of the existing applications of the attention mechanism for the KWS task; then, we propose different variations of an hybrid architecture based on attention, first introduced by de Andreade et al. [1]. This is done mostly to harness the importance of each block in the original model, and to explore new model architectures with relatively small memory footprint, in order to obtain better performances. We refer to this baseline as Att-RNN. We summarize the proposed contributions as follows:

- We explore a variation of the Attention layer: specifically, more query vectors[1] are used instead of a single one. We call this model SQAtt-RNN.
- We replace the Attention layer with a multi head attention layer [8], and explore how model performance changes by varying number of heads.
- We explore the role of the convolutional part of Att-RNN, by both removing it and enhancing it with a more complex architecture based on residual layers;
- We compare the performances of each proposed model, also in terms of their number of parameters.

Figure 1 shows a representation of Att-RNN and one of the proposed models, SQAtt-RNN. The other architectures will be described in detail later, but can be easily obtained by modifying single blocks starting from those two.

II. Related Work

A. Foundations and state of the art

In recent years, machine learning techniques have proven to be the de-facto standard for approaching the KWS problem. Such models typically perform segmentation of the audio sequence on the time domain and extract log-mel scale spectrograms or mel-frequency cepstral coefficients (MFCC) [15] from each frame. Those are then used as input feature vectors for the models. One of the first works exploring deep neural networks for the KWS task is from Chen et al [2]: the authors explore small footprint fully connected architectures and show how those improve performances with

[1] In this work, we use the *query, key, values* terminology introduced in [8].

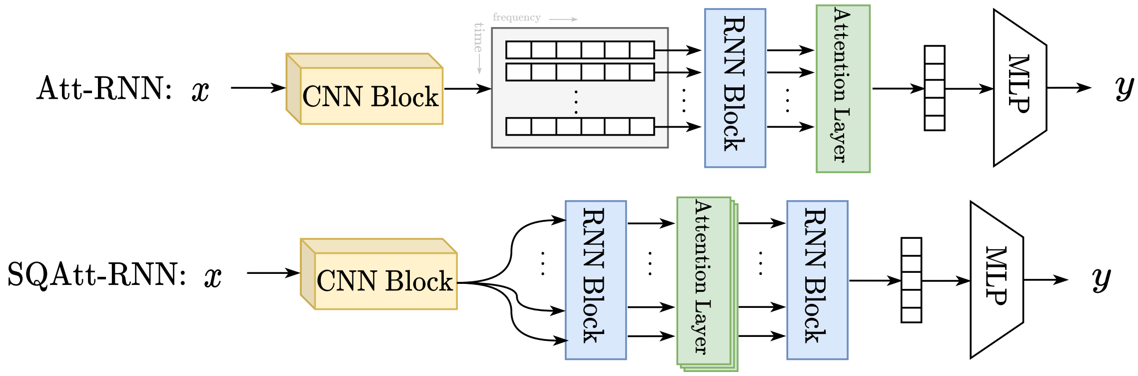


Fig. 1: Overview of Att-RNN model (baseline) and SQAtt-RNN. The input x is a matrix, where the i-th row is the vector of MFCCs computed for the i-th time frame. In both models, the CNN block outputs an image where the number of timesteps is preserved: feature vectors at each timestep are used as the input sequence for the RNN block. In the Att-RNN model, the attention layer returns a single context vector which is used for classification. In SQAtt-RNN, we use the whole sequence coming from the RNN block as query vectors: this results in a new sequence which is supposed to be a representation of the previous one. One last Bidirectional RNN layer scans the sequence and returns a single output vector, which is used for classification.

respect to the baseline HMM models. This kind of architecture accounted for the sequentiality of audio data by stacking feature vectors of adjacent audio frames: while this gives good results compared to the baseline, it is a very simplistic way to model sequential data. Sainath and Parada [3] approached the problem by using CNNs, a more fitting class of models which are able, by design, to capture several essential features of speech data (like input topology or translational invariance) and at the same time having a much smaller memory footprint due to parameter sharing. Indeed, CNNs have proven to be very successful for KWS: the current state of the art has recently been achieved in [16], where the authors introduce Broadcasted Residual Networks (BC-ResNets), a particular class of ResNets that are able to capture both 1D and 2D features thanks to the introduction of a new kind of residual block.

B. *Models based on Attention*

As a premise, it is useful to note that in the following applications of attention, the query, key and value vectors are all represented by the same vectors from the input sequence; therefore when referring to the attention mechanism, the reader should implicitly read self-attention [17]. Specifically, dot product self-attention is adopted [18]. In [1], De Andreade et al. propose an attention based convolutional recurrent neural network (Att-RNN), which takes as input a mel-scale spectrogram and extracts features in the frequency dimension with two convolutional layers. Then, such features are given as input to two bidirectional LSTM [19] layers, in order to extract long range dependencies from the inherently sequential audio data. The last[2] output feature from the LSTM layer is then transformed with a linear dense layer and used as a query vector for the attention mechanism. Finally, the sum of the LSTM outputs, weighted with respect to the attention scores, are given as input to a final dense MLP, which performs the classification. An immediate drawback of this approach is its impossibility to function in a streaming fashion (at least without introducing delay), since bidirectional recurrent layers require the whole sequence of data to work. Furthermore, a single representative for the whole sequence is used as a query vector: this could act as an information bottleneck. With SQAtt-RNN we seek to provide a richer representation of the input sequence, by returning another sequence from the Attention layer instead of a single compressed vector representation. In [4] the authors propose a more modern variant of Att-RNN (referred as MHAtt-RNN): LSTM is substitued by GRU [20], and the attention mechanism is replaced by a multi-head attention layer using 4 heads. Despite increasing accuracy with respect to Att-RNN, this kind of models can be very heavy in terms of memory footprint, especially if using a high encoding dimension for the multi-head attention layer. In the case of [4], the reported number of parameters is 743000, which is more than four times the number of parameters of Att-RNN.

The latest contribution in attention based models for KWS comes from [21], where the authors, inspired by the success of the newly introduced Vision Transformer (ViT) [9], propose an adaptation of such architecture for keyword spotting, called the Keyword Transformer (KWT). The results are suprising:

[2]In the paper, the authors report to use the middle output, but looking at their implementation, they use the last one. Either way, as they point out, any output of the LSTM layer should work well as a query vector, since the bidirectional LSTM layer should be able to summarize the whole sequence in any of its outputs.

despite ViT proved to be competitive only when supported by pre-training on large datasets, KWT outperformed more complex models based on mixes between CNN, RNN and attention, even when trained on relatively small datasets like the Google Speech Commands dataset [22]. Even in this case, despite good performances, it might not be feasible to integrate the KWT model on small mobile devices: indeed, the best performing variation of the KWT that was proposed is relying on a very deep architecture (12 Transformer encoder layers), which counts more than 5 million parameters. By contrast, our aim is to experiment with relatively small footprint models. In the next section, we describe the experiments and results obtained with the proposed architectures.

III. Experimental Setup

A. Signals and Features

Each model was trained on the Google Speech Commands dataset V2 [22], which consists on a total of 105829 user utterances of a total of 35 keywords. Each utterance is stored as a one second long[3] `WAVE` file, sampled at a 16KHz rate. From each signal, 40 MFCCs are computed, using a window size of 25*ms* (400 samples) and a hop size of 10*ms* (160 samples). With this approach, the input for the network consists in 3D tensors with one outer channel, where width represents the time domain and height represents the frequency domain. Following Google's suggestions from [22], the dataset is split in 80% for training set, 10% for validation set and 10% for test set, in such a way that the same speakers are never present in two different splits. Following the approach from past literature, the models are trained for two different tasks, which are the following:

- **12kws** task: the model must discriminate among 12 different keywords: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", unknown or silence. The unknown keywords are randomly chosen from the set of remaining keywords and the silence samples consist in one second long crops, randomly extracted from the noise files provided by the dataset[4]. Since the total amount of words belonging to the "unknown" class was much higher than the number of representatives for each of the other keywords, we decided to randomly extract them to create a balanced dataset. In this way, we have 36921 samples for training, 4443 for validation and 4888 for testing.
- **35kws** task: the model must discriminate among all the 35 keywords present in the dataset. No unknown class or silence class is introduced, therefore, for this task, the entire dataset is used. The training set consists in 84843 samples, the validation set in 9981 samples and the test set in 11005 samples.

[3] Some clips are a bit shorter than one second: in those cases, the clips are zero padded towards the end.

[4] Those consist in 6 files containing noisy background sounds, both artificially generated and recorded from real environments.

For each task, the training set is augmented, following existing approaches from the literature. Specifically, the augmentation process follows this order:

1) Each signal is randomly shifted left or right (zero padding the remaining portion), by x *ms*, where x is drawn from a uniform distirbution $U(0, 100)$;
2) Samples belonging to the "silence" class do not remain the same across epochs, but are randomly generated each time;
3) With a probability of 0.8, each signal is mixed with a randomly generated background noise, which is multiplied by a factor drawn from $U(0, 0.2)$;
4) After the conversion in MFCC, the features are augmented using SpecAugment [23], a simple data augmentation method which consists in adding randomly sized frequency and time masks to the feature matrix: this is done in order to render the model more robust to partial loss of frequency information or of small sections of speech. We set the maximum size for the time and frequency mask equal to 20 and 10 respectively.

The image resulting from the last step constitutes the actual input for the model. Specifically, each image is a 98×40 matrix, where the i-th row represents the MFCCs of the i-th time frame.

All the project was carried out using an NVIDIA GeForce GTX 1060 6GB GPU and an Intel i5-64000 CPU, on a Linux machine. All models were built and trained using TensorFlow [24]. All source code is available on Github[5].

B. Input Pipeline

Since data augmentation is applied, storing the entire dataset in memory would not be possible: this would result in the same data being reused across epochs. For this reason, a core part of this work was to build an efficient input pipeline that could handle data augmentation on the fly, during training. In this section we explain how the data generation pipeline works. The augmentation takes place in two different moments during training:

1) Phase 1: here, for each sample beloging to the *silence* class, random noise clips are extracted; also, each sample is randomly shifted. This is performed by the CPU, while the GPU is performing training.
2) Phase 2: this phase is performed inside the model by the GPU, with a series of preprocessing layers. Specifically, several custom preprocessing layers were built:
   a) `RandomNoiseAugment`: takes care of randomly adding noise to each waveform;
   b) `MFCC`: converts each waveform to its respective MFCC matrix;
   c) `SpecAugment`: performs the data augmentation following the SpecAugment policy.

Note that, in Phase 1, each operation is performed one sample at a time, while in Phase 2 each operation is performed

[5] https://github.com/rmazzier/HDA-Project-Key-Word-Spotting

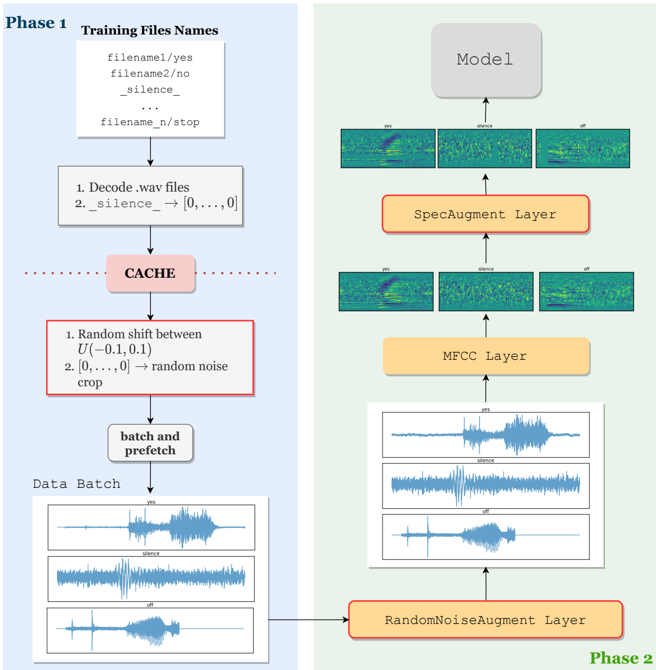


Fig. 2: Detailed description of the input pipeline for the training data. In the cache block, the data that was produced until that moment is saved to file. Each step which is performed before the caching operation happens only one time, during the first iteration of the dataset; on all the successive iterations, the data is read from the cached file. Boxes with the red outline denote data augmentation steps

in parallel on an entire batch of samples: this results in a much more optimized implementation. Given the hardware setup with which the project was carried on, this framework was a necessity: just computing the MFCCs for each sample on CPU caused a performance bottleneck, wasting the computational capabilities of the GPU. In Figure 2 we report a detailed diagram, showing all the steps in the input pipeline for the generation of the training set. For the validation and test sets, the augmentation is not applied. Furthermore, to ensure complete separation between training and validation/test sets, the noise samples were generated from different portions of the noise files, based on whether or not they were used for training.

C. Learning Framework

We now describe each proposed architecture, mainly using summary tables. Note that each RNN layer is bidirectional, each CNN layer uses stride $= (1, 1)$, padding = `same`, and uses batch normalization before passing through the activation function. Also, the first layers of each model are the preprocessing layers discussed in the previous section; here we don't report them as they are not part of the actual model architecture. In the tables we report with m the number of output classes, which differ depending on the type of task. Finally, we denote with "Attention" the attention layer used by Att-RNN, while with "QAttention" we mean the attention layer introduced for SQAtt-RNN. For all multi-head attention layers, the projection dimension is set to 64.

*1) **SimpleAtt**:* We propose this model as a baseline light weight model to see how its performances compare to more complex architectures. It's architecture is shown in Table 1.

TABLE 1: Simple Attention RNN architecture

layer type	nRNN	nMLP	nF	Fw	Fh	Act.
RNN (GRU)	64					tanh
Attention						
Dense		128				ReLu
Dense		64				ReLu
Dense		m				Softmax

*2) **Att-RNN**:* The baseline model, from [1]. Architecture is shown in Table 2.

TABLE 2: Att-RNN architecture

layer type	nRNN	nMLP	nF	Fw	Fh	Act.
Conv			10	5	1	ReLu
Conv			1	5	1	ReLu
RNN (LSTM)	128					tanh
RNN(LSTM)	128					tanh
Attention						
Dense		64				ReLu
Dense		m				Softmax

*3) **SQAtt-RNN**:* Simple variation of Att-RNN, where the attention layer is modified in order to return a new sequence. The idea is that the new returned sequence will constitute a richer representation of the input sequence. Architecture is shown in Table 3.

TABLE 3: SQAtt-RNN architecture

layer type	nRNN	nMLP	nF	Fw	Fh	Act.
Conv			10	5	1	ReLu
Conv			1	5	1	ReLu
RNN (GRU)	128					tanh
RNN (GRU)	128					tanh
QAttention						
RNN (GRU)	64					tanh
Dense		64				ReLu
Dense		m				Softmax

*4) **SQ-noCNN**:* Simple variation of SQAtt-RNN, where we completely omit the CNN block. This is done to investigate the impact of the initial feature extraction phase performed by the Convolutional layers. In this way, the input to the RNN block is the raw feature matrix.

*5) **MHAtt-RNN**:* Variation of Att-RNN, proposed in [4]. We train different versions of this architecture, varying the number of heads. Architecture is shown in Table 4.

*6) **SQMHAtt-RNN**:* Variation of MHAtt-RNN, where we use the whole sequence as query vectors in the same way it is done in SQAtt-RNN.

TABLE 4: MHAtt-RNNh architecture, where h is the number of heads for the MH Attention layer.

layer type	nRNN	nMLP	nF	Fw	Fh	Act.
Conv			10	5	1	ReLu
Conv			1	5	1	ReLu
RNN (GRU)	128					tanh
RNN (GRU)	128					tanh
MHAttention						
Dense		64				ReLu
Dense		m				Softmax

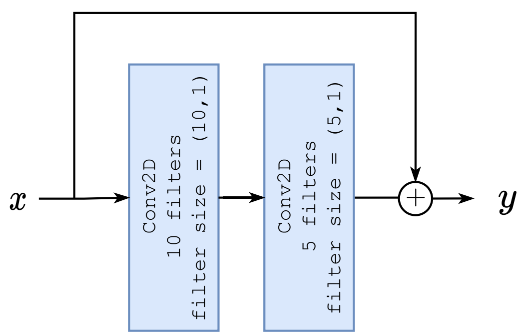


Fig. 3: Architecture for the residual block used in the Res-Att model.

7) ***Res-Att:*** Variation of Att-RNN where we use a deeper CNN block, constituted of residual layers. We train different versions of this architecture, varying the number of resdual blocks used. The residual block is shown in Figure 3. We report the whole architecture in Table 5.

TABLE 5: Res-Att$k$ architecture, where k is the number of residual blocks.

layer type	nRNN	nMLP	nF	Fw	Fh	Act.
Conv			5	5	1	ReLu
ResBlock ($\times k$)						
Conv			1	5	1	ReLu
AvgPool $(2, 1)$						
RNN (GRU)	128					tanh
RNN (GRU)	128					tanh
Attention						
Dense		64				ReLu
Dense		m				Softmax

Each model was trained for 30 epochs, using the Adam optimizer with a starting learning rate of 0.001. To learn as much as possible, in the case where after 2 epochs the validation loss didn't improve, the learning rate was reduced of a factor of 0.1.

IV. Results

Following the main approach in the KWS literature, we evaluate model performance based on the top-1 accuracy score on the test set. In Table 6 we report all the results for each model and for each task. Additionally, in Figure 6 we provide a plot of each model accuracy against its number of parameters. The SQAtt-RNN provides a comparable accuracy for the 12kws task, while giving slightly worse results for the 35kws task. This suggests that providing a single query vector representative for the whole sequence is more than sufficient when dealing with relatively short sequences. Indeed, the query vector comes from a bidirectional GRU layer, which proved to be really effective at capturing the relations between the sequence elements. The models based on multi head attention didn't provide consistent results: specifically, contrary to expectations, increasing the number of heads didn't necessarily increase the performances of the model. Nevertheless, the best scoring models were SQMHAtt-RNN5 for the 12kws task, and SQMHAtt-RNN2 for the 35kws task. While it is true that multi head attention models are heavier in terms of memory footprint with respect to Att-RNN, the ones proposed here still present less parameters than the one proposed in [4], which reported to have around 700K parameters.

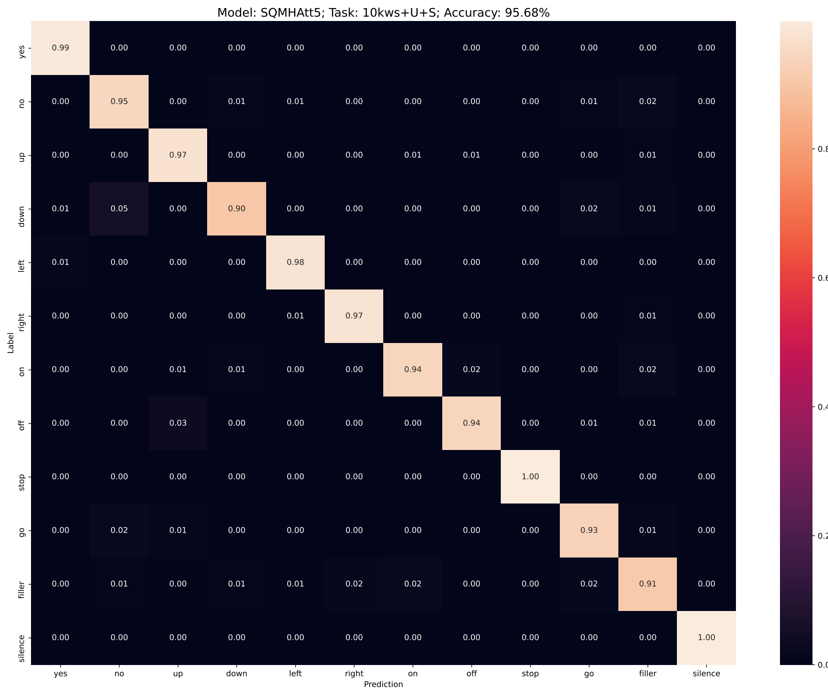
TABLE 6: Results in term of top-1 accuracy for both tasks.

Model	Param.	Acc. 12kw	Acc. 35kw
SimpleAtt	84 K	92.9%	92.8%
Att-RNN	180 K	94.7%	95.2%
SQAtt-RNN	169 K	94.9%	94.6%
SQAtt-noCNN	169 K	94.8%	94.8%
MHAtt-RNN2	208 K	95.1%	94.8%
MHAtt-RNN3	241 K	95.2%	95.3%
MHAtt-RNN4	274 K	94.4%	95.3%
MHAtt-RNN5	307 K	95.5%	95.1%
SQMHAtt-RNN2	235 K	95.3%	**95.4**%
SQMHAtt-RNN3	268 K	95.4%	94.4%
SQMHAtt-RNN4	301 K	94.7%	94.3%
SQMHAtt-RNN5	334 K	**95.7**%	95.0%
Res-AttRNN3	182 K	95.1%	94.6%
Res-AttRNN4	183 K	94.4%	94.7%
Res-AttRNN5	184 K	94.0%	94.8%

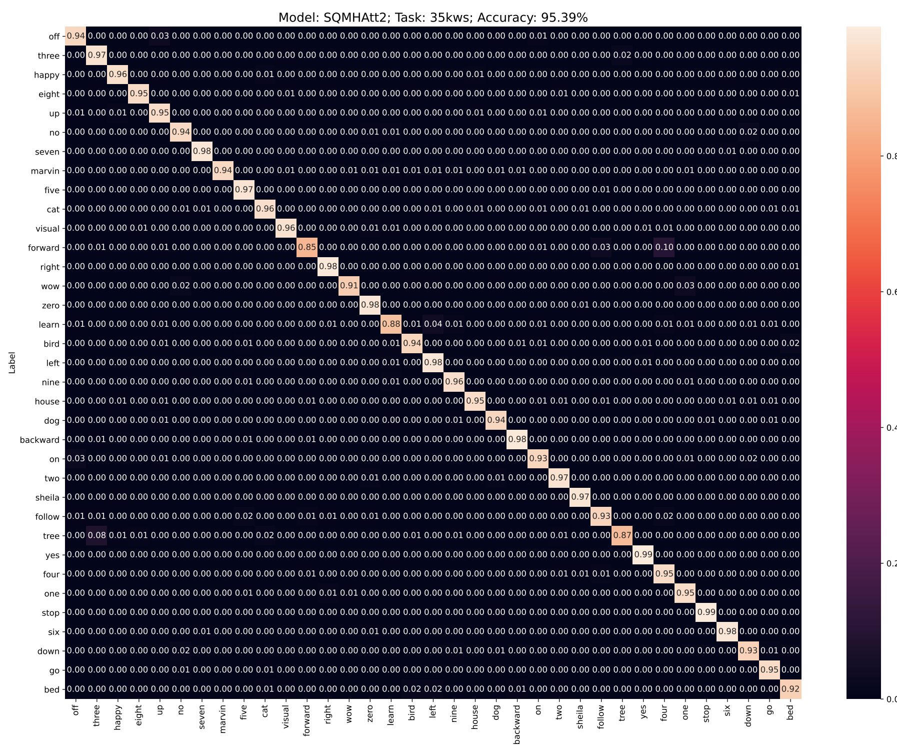
In the same way, also the ResAtt models didn't provide consistent results, and in general seemed to perform worse than the simpler CNN block from Att-RNN. Maybe results in this sense can be improved employing more sophisticated recurrent blocks. Also, by looking at the performances of the SQ-noCNN model, it is interesting to note that, for the 12kws task, accuracy is slightly higher with respect to Att-RNN, while for the 35kws task it is lower. This could suggest that the feature extraction task performed by the CNN block might be more important when in need to recognize an higher amount of key-words. In Figure 4, we also provide a confusion matrix for the best performing models, for both tasks.

A. Attention Plots

As already mentioned, a really convenient feature of models based on attention is that it is easy to interpret them. Specifically, we can plot the attention scores of our models to



(a) Confusion matrix of the best performing model for the 12kws task. The words where the model struggle the most are the unknown ones: specifically, known words are often recognized as unknown. Note that this is a desirable situation for a KWS system since we prefer false negatives instead of false positives. Another aspect which should be improved is in the recognition of the word *down*, which is often recognized as *no*.



(b) Confusion matrix of the best performing model for the 35kws task. In this case, two common mistakes can be distinguished. First, the word *forward* is often confused with the word *four*. This is to be expected, since both present the phoneme /fɔr/ at the beginning. A similar common error occurs with the word *tree* which is often confused with *three*. Apart from the obvious similarity between the words, another possible reason for this confusion, as pointed by the authors in [1], might be due to the inability of non native english speakers to correctly pronounce the phoneme /θ/ in *three*.

Fig. 4: Confusion Matrices

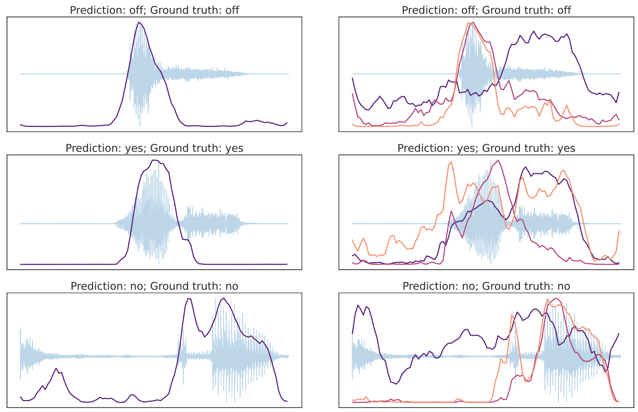


Fig. 5: Comparison between attention scores from Att-RNN model (left) and MHAtt-RNN3 (right), on the words *off*, *yes* and *no*.

see which portions of the audio files were more important for the model in order to perform inference. In Figure 5 we visualize log attention weights for Att-RNN and MHAtt-RNN3 models. We can see that Att-RNN has only one head, so one set of attention scores per prediction is computed. MHAtt-RNN instead computes one set of attention weights per head: here we visualize the attention scores for each head. In these examples, we can see how each head learns to pay attention to different phonemes of the same word. In the first example, Att-RNN pays attention only to the first phoneme /o/, while MHAtt-RNN has two heads paying attention to /o/ and one paying attention to /f/. In the second example, a similar thing happens: Att-RNN pays attention just at the phoneme /je/ while MHAtt-RNN has different heads concentrating both on /je/ and /s/. The third example presents a noise at the beginning which is not part of the spoken word: Att-RNN has its attention drawn a bit, while two of three heads from MHAtt-RNN learn to completely ignore it.

V. Concluding Remarks

In this work, we explored different variants of attention based models for keyword spotting. In particular, we found that models based on multi head attention provide slightly higher accuracy with respect to the baseline Att-RNN model, even if this came with the cost of a higher parameter count. Deeper convolutional blocks didn't prove to be beneficial, even if this could be due to a too simplistic design for the residual layers. Similarly to many works in the literature, we used the top-1 accuracy as the only evaluation metric, but this is not a complete way to evaluate a KWS model. For example, authors in [2][3][4] make audio streaming tests to evaluate the false reject and false alarm rate, as well as system latency, to have additional tools for evaluation. Also, for this project, the models were trained just one time due to the limited computational capabilites; this fact makes the reported results not too statistically relevant. To perform a more rigorous statistical analysis, one should train the models more times and use an avarage of the test set accuracies among the runs, computing confidence intervals for the final accuracy. Furthermore, additional experiments regarding attention mechanisms could be done starting from the Keyword Transformer architecture; even if the KWT has an extremely heavy memory footprint, it could be worth experimenting lighter variations of it.

In conclusion, this was a very instructive project to work on: I had the opportunity to study a lot of modern machine learning literature and to understand more complex architectures based on the attention mechanism. I also think that this report LaTeXtemplate was extremely useful and well done, and it will surely be a very useful tool for the future. Besides the part involving the study of the literature, the most difficult part of this work, in my experience, was to build a working and efficient input pipeline, both for technical reasons (due to the limitations of my hardware) and for the difficulty to find information online. While it is true that the laboratories were extremely useful and essential, especially on the part regarding the `tf.data.Dataset` API, while working on the project I often came across errors which were really hard to debug mostly because of my unawareness of how Tensorflow really works under the hood (see for example the difference between Graph execution and Eager execution). Besides this aspect, I think that the course gave me strong knowledge foundations, essential in order to complete the project.

References

[1] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *arXiv preprint arXiv:1808.08929*, 2018.

[2] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4087–4091, 2014.

[3] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *INTERSPEECH*, 2015.

[4] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," *Interspeech 2020*, 2020.

[5] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5484–5488, IEEE, 2018.

[6] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll, "Small-footprint keyword spotting on raw audio data with sinc-convolutions," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7454–7458, IEEE, 2020.

[7] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal convolution for real-time keyword spotting on mobile devices," *arXiv preprint arXiv:1904.03814*, 2019.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin,

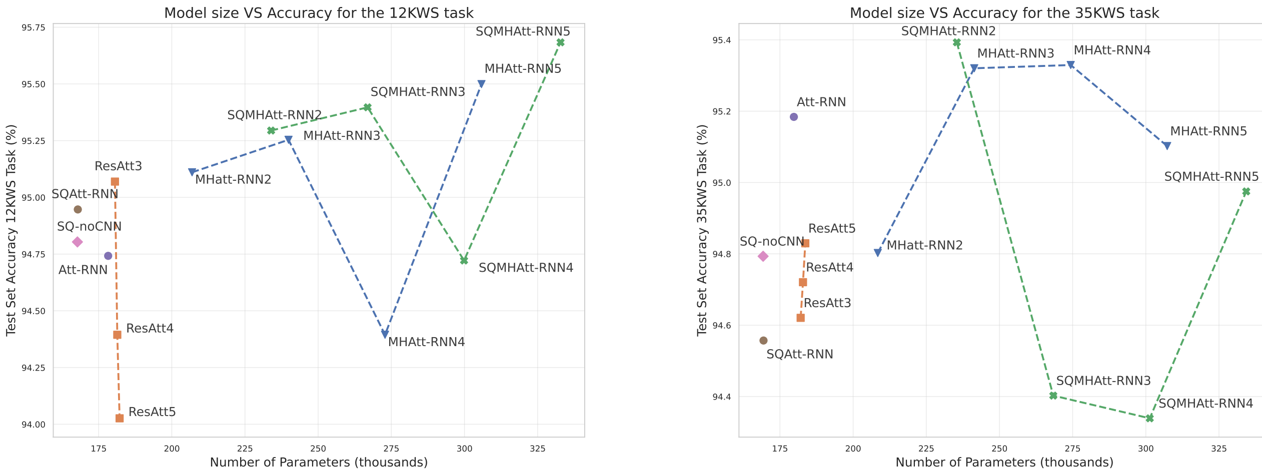


Fig. 6: Models size vs. test set accuracy, for the 12kws (left) and 35kws task (right).

"Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[10] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, pp. 10347–10357, PMLR, 2021.

[11] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[12] M. Kumar, D. Weissenborn, and N. Kalchbrenner, "Colorization transformer," *arXiv preprint arXiv:2102.04432*, 2021.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.

[14] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a 'right to explanation'," *AI Magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[15] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[16] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," *arXiv preprint arXiv:2106.04140*, 2021.

[17] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.

[18] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] K. Cho, B. V. Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[21] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword transformer: A self-attention model for keyword spotting," *arXiv preprint arXiv:2104.00769*, 2021.

[22] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[23] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[24] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.