Hand Gesture Recognition on Arduino Using Recurrent Neural Networks and Ambient Light

Matthew Lipski
Supervisors: Mingkun Yang , Ran Zhu
EEMCS, Delft University of Technology, The Netherlands
m.s.lipski@student.tudelft.nl, m.yang-3@tudelft.nl, r.zhu-1@tudelft.nl

Abstract

1 Introduction

1.1 Research Overview

Traditionally, physical buttons have been by far the most common way for users to interact with electronic devices in public settings, whether these are coffee machines, elevator panels, or train ticket machines. However, the concern of disease transmission has become increasingly prevalent in recent years due to the COVID-19 pandemic, making it enticing to develop an alternative solution which does not require touch. One such solution is the use of hand gestures to interact with public devices instead. By performing hand motions such as swiping and tapping, users can effectively and intuitively interact with electronic devices with minimum risk of disease transmission.

However, there are a few key challenges to this approach which stand in the way of it replacing physical buttons in real-world applications.

1. Additional hardware is needed to detect the positions of a user's hand while performing a gesture. The data output by this hardware is likely to be low in resolution since system costs should be minimized.
2. Additional software must be implemented to recognize gestures based on hand position over time. While buttons are just simple digital inputs, one gesture can also be performed differently between different users, yet the system must be able to accurately classify it regardless.
3. Gesture recognition must be done in real-time. Since the process of classifying gestures can be quite complex, the latency introduced by this may be significant. However, the user should not perceive any lag while using the system for a positive experience.

1.2 Research Question

Given the challenges in developing a neural network for gesture recognition, the goal of this paper can be summarized with the following research question:

'Which neural network architecture is most appropriate for recognizing hand gestures on an Arduino Nano 33 BLE, using 3D-formatted data from OPT101 photodiodes?'

This can then be segmented into the following sub-questions:

1. Which neural network architectures produce the highest accuracy for hand gesture recognition?
2. Which neural network architectures produce the lowest inference latency for hand gesture recognition?
3. What is the minimum acceptable accuracy for recognizing hand gestures on an Arduino Nano 33 BLE?
4. What is the maximum acceptable inference latency for recognizing hand gestures on an Arduino Nano 33 BLE?
5. How can 3D-formatting data be exploited for better gesture recognition performance?

1.3 Contributions

This research overcomes these challenges by using data from OPT101 photodiodes, which is fed into a neural network to recognize gestures with high accuracy and low latency on an Arduino Nano 33 BLE microcontroller. It is also part of a larger project which integrates this neural network into a full gesture recognition system, which is elaborated on in section 3.2.

Similar research which involves using photodiodes and machine learning to recognize hand gestures has already been conducted, but this paper improves on existing solutions in a number of ways:

1. A high level of accuracy is maintained with fewer photodiodes, and therefore fewer model input features, than

existing solutions.

2. The data from photodiodes is 3D-formatted, which better preserves temporal information and improves recognition accuracy. An explanation for what 3D-formatting involves can be found in section 2.4.
3. The neural network used to classify hand gestures is smaller in size than competing solutions, leading to reduced system latency.

A discussion regarding existing research in this field is found in section 8, which provides more context to these improvements.

2 Background

2.1 Machine Learning & Artificial Neural Networks

Machine learning is a sub-field of artificial intelligence which "provides learning capability to computers without being explicitly programmed" [1]. More specifically, machine learning allows computers to "learn" patterns and trends from existing data in order to make accurate predictions on new data, without any human input.

Artificial neural networks (ANNs) are a type of machine learning model which draw inspiration from the human brain [13]. These types of models feature "neurons" organized into densely connected layers. The first layer, which takes in the input data, is known as the input layer while the final layer, which yields the processed output data, is called the output layer. In between these is at least one hidden layer and in general, having more hidden layers and more neurons per hidden layer allows a neural network to approximate increasingly complex functions.

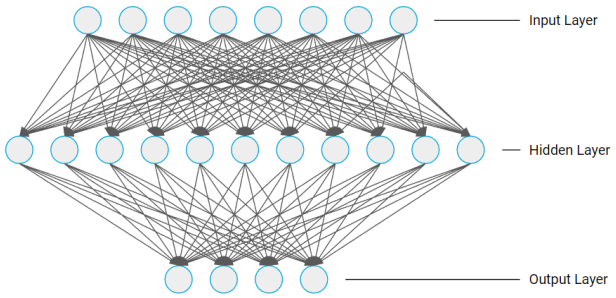

Figure 1: Visualization of a generic ANN with 8 input neurons and 4 output neurons, as well as a single hidden layer with 11 neurons.

Each connection between two neurons has an associated weight and bias, which determine how much the output from the source neuron affects the output of the destination neuron. These values change as the model "learns" from existing data, causing the network's performance to gradually improve. In addition to this, each neuron passes the combined input of all neurons in the previous layer through a non-linear activation function, which is what allows neural networks to effectively model any function possible [13].

Although ANNs are no longer considered state-of-the-art, understanding the purposes of neurons, neuron connections, layers, and activation functions remains useful, as these still form the building blocks of any neural network architecture.

2.2 Neural Networks on Embedded Hardware

Machine learning, and specifically neural networks, have traditionally been restricted to the realms of high-performance and in turn, high power devices [2]. Unfortunately, this means that it has been previously impractical to use these technologies with embedded hardware such as microcontrollers, as they lack the memory & performance to run inference on neural networks locally, while using a remote processor for inference is not always feasible.

However, recent advances into machine learning model compression and optimization have changed this, allowing deep neural networks with multiple hidden layers to be run on devices even powered by coin batteries. The most prominent development in this field has been TensorFlow Lite for Microcontrollers, which is a Python framework specifically made for optimizing and running neural networks on microcontrollers [4]. The original Tensorflow has been an extremely popular machine learning framework for over a decade, but the extension to make it work effectively on microcontrollers has only been developed in recent years.

2.3 Hand Gesture Data

In this research, neural networks are used to detect gestures, but to do this, data from some sensor(s) must be fed into the network.

There are a variety of sensors which could be used to record hand gestures and provide this data. One of these is an accelerometer attached to the user's wrist (typically from a smartwatch) to track the direction and acceleration of the their hand movements [10]. Another option is using a depth/range camera to record the user's hand, which provides a huge amount of information but in turn requires a computationally intensive video processing pipeline to make sense of the data [9]. Photodiodes can also be used, which are sensors that output a signal which increases with the amount of light that hits them. This means they can track the shadows cast by the user's hand under ambient light, therefore making it possible to recognize which gesture is being performed [5]. This project uses photodiodes for their much lower monetary and computational cost compared to cameras as well as the fact that they don't require the user to wear a device on their wrists, unlike accelerometers.

2.4 3D-Formatted Data

The term ”3D-formatted data” is specific to this research, and must be explained to understand the choice of neural network architectures tested. This is best done by comparing it to ”2D-formatted data”.

2D-formatted data can be represented as an image, with some horizontal resolution x and vertical resolution y. In this research, each photodiode outputs values at a predetermined sampling rate over the course of a gesture. This means that data can be formatted as a 2D image in which x is the number of photodiodes used and y is the number of total samples received from any of the photodiodes. Therefore, the value of each ”pixel” in the image represents a reading from a single photodiode at a single point in time.

3D-formatted data can meanwhile be thought of as a video, which splits this 2D-image into a sequence of n frames, as shown in figure 2. 3D-formatting is generally more appropriate when the data is sensitive to time, i.e. when data points should be considered in a specific sequence. Therefore, by 3D-formatting the photo diode data, lower error rates should be achievable as temporal information from the photo diodes isn't lost.

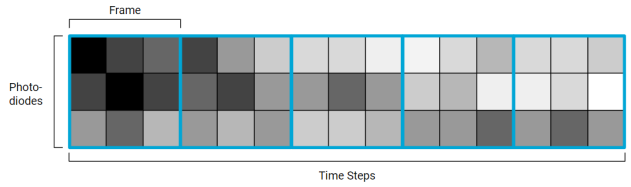

Figure 2: Visualization of 2D photodiode data after being 3D-formatted into 5 frames.

3 System Overview

3.1 Full Gesture Recognition System

This research focuses on finding an appropriate neural network architecture to perform gesture recognition on a microcontroller, but it is only part of a larger project to create an entire gesture recognition system/pipeline. The creation of this pipeline is composed of the following tasks:

1. Optimizing the number and placement of OPT101 photo diodes.
2. Pre-processing data from photo diodes.
3. Creating an appropriate dataset for training a neural network to recognize gestures.
4. Finding an appropriate neural network architecture on the created dataset and ensuring gestures can be recognized in real-time on an Arduino Nano 33 BLE.

The research presented in this paper aims to complete task 4. Although tasks 1–3 were completed by other project group members and are beyond the scope of this research, they are worth mentioning to provide some context regarding the rest of the gesture recognition system. Due to the findings from these tasks, the final system uses 3 photodiodes and can recognize 10 different gestures, while each gesture is composed of 100 samples from each photodiode. This is relevant for task 4, as it means that whatever neural network is implemented must use a 2D array of size 3 by 100 (split into n frames after 3D-formatting) as an input feature and be able to distinguish between 10 output classes, as illustrated in figure 4.

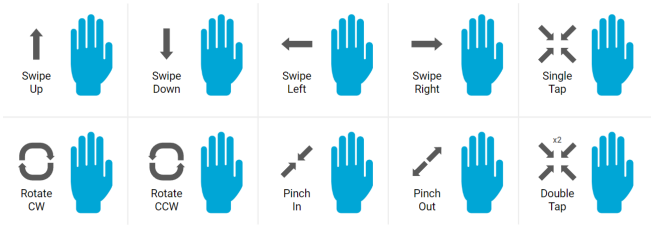

Figure 3: Illustrations of the 10 different gestures that the system can recognize.

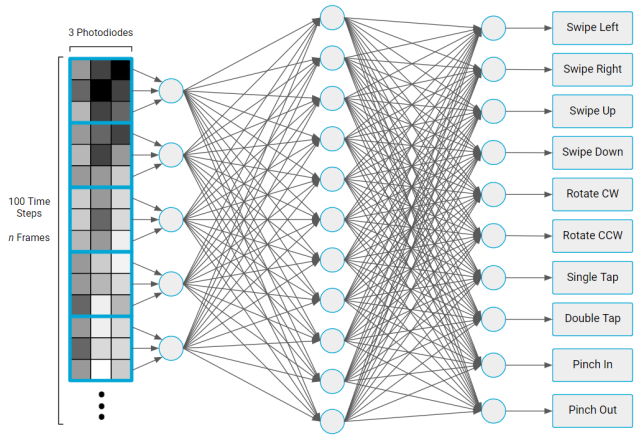

Figure 4: Visualization of the input features & output classes using a generic artificial neural network as an example.

3.2 System Caveats

The overarching goal of the project that this research contributes to, is the creation of a full gesture recognition pipeline, which presents some issues when considering the fact that each part of this pipeline was developed in parallel. In reality, it would make much more sense to complete each step of the pipeline sequentially, as the performance of later parts of the pipeline relies on the performance of previous parts. To put this in the context of the gesture recognition system, it is impossible to train a neural network to recognize gestures without first having a dataset to train it on. However, the creation of that dataset relies on photodiode count and

placement, as well as sampling rate and pre-processing, being finalized. If they change after the dataset is completed, it will not be representative of real-world data, leading to poor gesture classification performance from the neural network trained on it.

3.3 Dataset

Having a varied, expansive, and representative dataset is crucial for training a machine learning with high real-world accuracy. Fortunately, a dataset for recognizing gestures using photodiode data was done by another member of the project group, as mentioned in section 3.1.

Unfortunately, because the allotted time for the project meant that all group members had to work in parallel, as stated in section 3.2, the neural networks evaluated in this paper had to be trained on a dataset that is not final. Specifically, the dataset used in this research is not passed through the data pre-processing stage of the system. This means that the neural networks presented in this paper were trained on raw data from the photodiodes, whereas real-world data on the final system would include this pre-processing step. This leads to accuracy measurements being lower than they could be, as there is noise and other artifacts in the raw photodiode data.

The dataset used contains 5 repetitions of each of the 10 gestures per hand across 48 participants, leading to 4800 total data instances. Each instance is made up of a 5 second window during which a gesture is performed over the photodiodes, with a sampling rate of 20Hz for a total of 100 samples per photodiode. Although the dataset contains instances from a variety of environments and lighting setups, these are mostly indoor locations as this is the planned use case for the system. The dataset was also mostly recorded on the TU Delft campus, meaning the demographic of participants is somewhat skewed. Most notably, the dataset contains substantially more instances of males compared to females, and more right-handed participants than left-handed. However, this is not expected to have a large impact on the performance on any neural networks trained on this dataset.

4 Model Design

4.1 Neural Network Architectures Tested

Recognizing hand gestures based on photodiode data can be thought of as a time-series classification task, which is a type of problem that recurrent neural networks (RNNs) are especially well suited for [7]. RNNs differ from conventional neural networks as they do not process the input in its entirety, but instead process each time step or sample from the input sequentially. Each time step is used to update a "hidden state", which is fed back into the RNN along with the next time step. This gives RNNs the ability to exploit the time-sensitive nature of time-series data, as each time step also has temporal information associated with it. Thanks to this desirable property, RNNs were the first type of neural network tested.

Although RNNs are highly suitable for time-series classification, they suffer from the "vanishing gradient problem", which has since been overcome by long short-term memory cells (LSTMs) and gated recurrent units (GRUs) [6]. To briefly explain this problem, the weight of the hidden state of a time step in an RNN tends to exponentially decrease for future time steps. This means that in practice, RNNs tend to ignore if earlier time steps are out of order, which limits performance for long data sequences. LSTMs and GRUs solve this issue using so-called "gates", which determine which contents of the previous hidden state should be kept and which contents of the current hidden state should be propagated to the next time step. This largely mitigates the vanishing gradient problem as only relevant information is kept, therefore greatly reducing the rate at which hidden state weights decay. Given that LSTMs and GRUs are should yield better classification performance than RNNs due to this advantage, these were also investigated.

One issue of using LSTMs and GRUs, however, is that they only accept a single data sequence with multiple features as input, in which each time step contains a single value from each feature, i.e. a 1D array. This is relevant as with 3D-formatted data, each time step is a 2D frame - not a 1D array. Therefore, each frame has to be flattened first, which causes a loss of spacial information. This issue can be solved by using a convolutional neural network (CNN) to remove noise from the input data and extract features, which are then fed into the LSTM [8]. CNNs accept 2D images as input, which means that frames do not have to be flattened. By using convolutional and pooling layers, the resolution of this image can be reduced to just a single value for each neuron in the final layer of the CNN. Therefore, the final CNN layer effectively outputs a 1D array with a size equal to the number of neurons in it, which can then be used as input to the LSTM with no loss in spacial information. Due to this, the CNN-LSTM architecture was also investigated in this study.

Finally, transformer encoders were also tested for this project as a novel alternative to RNN based architectures. Transformers use a concept called self-attention which identifies which elements in a sequence carry most semantic value based on the other elements of the sequence [12] This is fundamentally different to the hidden state mechanism used by RNNs and their derivatives. They are typically used in language translation and are made up of two parts, an encoder and a decoder. The encoder converts the input into an internal representation, which can then be manipulated and decoded into a more useful format, while the decoder converts this internal representation back into a more useful data type. In a study by Yüksel *et al.* [15], a transformer encoder was used to classify topics from sequences of Turkish text taken from Twitter. Although this is clearly different to the goal presented in this paper, i.e. classifying gestures based on photodiode readings, both are classification tasks using input data in which order is important. Given the positive results found in this study,

it seemed reasonable to test transformer encoders for gesture recognition as well.

4.2 Parameter Tuning

When training a machine learning model, the model parameters can have a substantial impact on final performance. For the neural networks tested in this paper, some of these parameters affect almost all model types, while some are specific to individual architectures. These are outlined below:

All Architectures

- Frame size
- Number of layers
- Number of neurons per layer

CNN+LSTM

- Number of convolutional layers
- Number of neurons per convolutional layer
- Filter kernel sizes

Transformer Encoder

- Number of attention heads

In reality, there are more parameters that could be tuned, but the ones listed above are likely to have the largest impact on final model performance. This paper does not go into detail regarding the testing of different parameters to determine which combinations yield optimal performance, but the final parameter values are stated in section 7.

5 Model Implementation

5.1 Training Code

The implementation and testing of each neural network architecture was done using the TensorFlow Python library and high-level Keras API (version 2.9). While the dataset is split into training and validation sets when evaluating architecture performance, the final models to be deployed on Arduino are trained on all available data.

5.2 Inference Code

To run trained Keras/TensorFlow models on the Arduino Nano 33 BLE, they were first converted to TensorFlow Lite models with integer quantized parameters and arguments to optimize space and runtime performance. The TensorFlow Lite models were then converted into C++ files so that they can be loaded and ran using TensorFlow Lite for Microcontrollers. This was done using the method detailed in chapter 4.5 of "TinyML Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers" [14].

6 Results

6.1 Inference Latency Testing

Methodology

Unfortunately, it is not currently possible to test the inference latency performance of most neural networks architectures tested directly on the Arduino Nano 33 BLE. As mentioned in section 5, the TensorFlow library is used to train the neural networks, while TensorFlow Lite for Microcontrollers is used run inference on them. However, the Arduino implementation of TensorFlow Lite for Microcontrollers does not currently have support for recurrent neural networks, which are also needed for all other architectures tested excluding transformer encoders. While it is possible to use RNNs with the most recent version of TensorFlow Lite for Microcontrollers (version 2.9), the library's Arduino implementation is a few versions behind (currently version 2.4), and is outdated enough that it is missing certain tensor operations necessary for running RNNs and their derivatives.

Therefore, a different method was used to estimate the inference latency of RNNs on Arduino. Since regular artificial neural networks are compatible with TensorFlow Lite for Microcontrollers version 2.4, several configurations of these with varying neuron and densely connected layer counts were trained and deployed on the Arduino Nano 33 BLE. These configurations are listed in table 1, formatted as (number of densely connected layers layers/number of neurons per layer). Each configuration also includes a final densely connected layer of 10 neurons for output. Since ANNs require 1D input data, a dummy data instance was taken from the dataset and flattened into a single array of 300 values (3 photodiodes $\times$ 100 samples). This dummy data instance was then used as input to the networks so that their latencies could be measured. The inference times of these generic networks on the Arduino Nano 33 BLEs were compared to their TensorFlow Lite file (.tflite extension) sizes, to see if file size could be used to predict inference latency.

Results

Figure 5 illustrates that a neural network's inference latency can indeed be estimated using the function $y = 1.89x + 23.6$, where x is the model file size in kilobytes and y is the model's expected inference latency in milliseconds.

However, it is worth noting that the largest model for which latency was tested, with a 160KB file size, only produced a latency of 333 milliseconds despite using 97% of the available RAM on the Arduino Nano 33 BLE (this memory usage

Configuration	File Size (MB)	Latency (ms)
(1/128)	42	103.9
(1/256)	81	179.6
(1/512)	160	333.0
(2/64)	27	74.7
(2/128)	59	134.5
(2/256)	147	293.7
(4/32)	17	55.4
(4/64)	59	92.3
(4/128)	93	195.0

Table 1: Table comparing the file sizes and inference latencies of various TensorFlow Lite artificial neural networks.

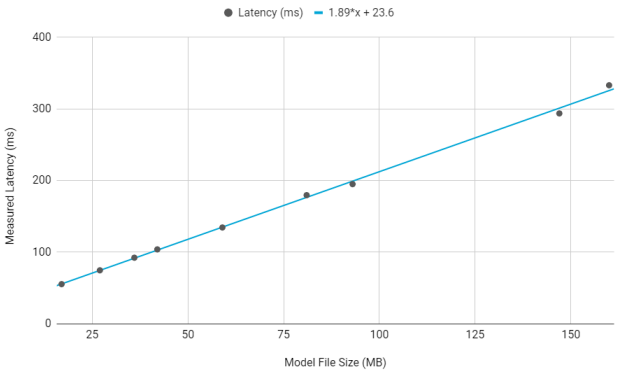

Figure 5: Graph showing the relationship between file size and inference latency for various TensorFlow Lite artificial neural networks.

includes the boilerplate code needed for loading and running the model). A latency of 333ms is similar to human reaction time (CITATION NEEDED) and can therefore be considered real-time for the purpose of gesture recognition. Therefore, inference latency is not much of a concern, though it is important that all models tested in this study do not exceed a file size of ĩ60KB, so that they could be run on the Arduino in the future.

6.2 Accuracy Testing

Methodology

To ensure that the measured validation accuracy was representative across all neural network architectures tested, four distinct measures were put in place:

1. A dropout layer with $p = 0.5$ was added before the output layer for each architecture to reduce overfitting [11].
2. K-fold cross-validation [3] was used with $k = 5$, as validation accuracy can be skewed based on how the test and validation sets are split.
3. Each neural network was trained until the value of the validation function no longer improved for 100 consecutive epochs, mitigating the variation in training time caused by randomizing initial neuron weights as well as differences in architecture.
4. Data instances for each participant in the dataset were shuffled randomly, as ordered data can cause slow training times and poor performance. However, the order of participants was not shuffled to ensure that gestures in the training data were recorded by different participants to gestures in the validation data, making validation more representative of a real-world scenario.

7 Accuracy Testing

RNN

LSTM

GRU

CNN+LSTM

Transformer Encoder

8 Related Work

9 Responsible Research

All code used for this research is open source can be found at https://github.com/matthewlipski/research_project. Due to the inherent randomness present when training neural networks, the results presented in this paper may not be completely reproducible. However, using the same code and hardware, it should be possible for anyone to reach the same conclusions based on their own findings.

10 Conclusion

References

[1] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series*, 1142:012012, nov 2018.

[2] Liliana Andrade, Adrien Prost-Boucle, and Frédéric Pétrot. Overview of the state of the art in embedded machine learning. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1033–1038, 2018.

[3] Daniel Berrar. *Cross-Validation*. 01 2018.

[4] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. Tensorflow lite micro: Embedded machine learning for tinyml systems. In A. Smola,

A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 800–811, 2021.

[5] Haihan Duan, Miao Huang, Yanbing Yang, Jie Hao, and Liangyin Chen. Ambient light based hand gesture recognition enabled by recurrent neural network. *IEEE Access*, 8:7303–7312, 2020.

[6] Yuhuang Hu, Adrian E. G. Huber, Jithendar Anumula, and Shih-Chii Liu. Overcoming the vanishing gradient problem in plain recurrent networks. *CoRR*, abs/1801.06105, 2018.

[7] Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

[8] Tae-Young Kim and Sung-Bae Cho. Predicting residential energy consumption using cnn-lstm neural networks. *Energy*, 182:72–81, 2019.

[9] Herve Lahamy and Derek Lichti. Real-time hand gesture recognition using range cameras. 05 2012.

[10] Jiayang Liu, Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. In *2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–9, 2009.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[13] Sun-Chong Wang. *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA, 2003.

[14] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly, 2020.

[15] Atıf Emre Yüksel, Yaşar Alim Türkmen, Arzucan Özgür, and Berna Altınel. Turkish tweet classification with transformer encoder. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1380–1387, Varna, Bulgaria, September 2019. INCOMA Ltd.