IntentBot: Building Machine Learning Systems For Automated Intent Detection

Robert Yang
bobyang9
Stanford University
bobyang9@cs.stanford.edu

Tai Vu
taivu
Stanford University
taivu@stanford.edu

Abstract

Intent detection is both an active area of research and a highly needed service spanning many industries including technology, logistics, and finance. In this project, we build a number of machine learning and deep learning models to classify user queries to correct intents. We find that the BERT model achieves the highest performance, with 99+% accuracy scores and $0.99+$ F1 scores on the ATIS and SNIPS datasets.

1 Introduction

Intent detection is an important research area in natural language understanding (NLU). This task involves identifying different types of actions from user commands and queries. This is a practical and applicable problem in the development of task-oriented dialog systems. For example, AI-powered chatbots can automate customer service by classifying requests from emails, social media, or texts into categories such as "Bug Report", "Question", and "Purchase", thus boosting business productivity.

However, intent detection is sometimes difficult and nuanced. There have been active research in leveraging AI systems to extract user intents [1]. Nevertheless, closing the gap between AI performance and human performance proves challenging. Therefore, in this study, we implement and evaluate various machine learning and deep learning algorithms for intent classification, including Naive Bayes, Softmax Regression, SVM, LSTM, and BERT. The input to our algorithm is a sentence indicating a user command. Our classifiers then output a predicted intent label.

2 Related Work

In recent years, the advent of deep learning breakthroughs has driven major advances in natural language processing. Researchers have invented sophisticated architectures for understanding human languages, such as CNNs, RNNs, LSTMs [16], and Transformers [15].

There have been successful attempts in applying deep learning to intent classfication. In particular, RNN models have been extensively used in designing intent detection systems [8, 9, 10]. Some other research groups have developed more complex LSTM [11, 12] and BERT architectures [13], which resulted in higher accuracy levels. The benefits of these neural networks is that our models learn deep vector encodings that capture the semantic meaning of the user commands, thereby classifying them into correct intent classes. However, these performance levels come at the cost of computational complexity, which sometimes hinders model deployment in dialog systems.

In this project, we build two simple LSTM and BERT neural networks. Additionally, we develop some machine learning algorithms like Naive Bayes, Softmax Regression, and SVMs, and compare their performance with the deep learning approaches.

3 Dataset and Features

We used the following two popular datasets for intent classification benchmark.

- **Kaggle ATIS (Air Travel Information System)** [2] is a simplified version of the original MS-ATIS dataset, and contains English air travel related queries such as `airfare`, `flight`, and `ground service`. There are 4977 training examples and 893 test examples, which belong to 8 intents.
- **SNIPS** [3] consists of English day-to-day user commands like `get weather`, `play music`, and `book restaurant`. The dataset has 13802 training samples and 699 test samples. There are 7 user intents, and each contains 2400 examples.

For data preprocessing, we wrote a Python script to convert raw data files into pairs of inputs and outputs. Then, we removed punctuation, turn the sentences into lowercase, and split into lists of words by spaces, etc. Since we did not have validation data for ATIS and SNIPS, we wrote a program to split each provided training dataset into 95% for training and 5% for validation.

Regrading feature extraction, we implemented a Bag of Words approach. Specifically, we created a dictionary of all the words which appear in at least 5 training examples. Subsequently, for each input x, we generated a feature vector $\phi(x)$, where each entry x_j in $\phi(x)$ is either the number of occurrences of the j-th word in x or a binary value indicating whether x contains the j-th word. This gave us two different sentence encodings that were appropriate for different machine learning models.

In addition, we implemented two neural word embeddings, including GloVe [6] and BERT [7]. These two encoders were pre-trained on large English corpora and are able to generate vector representations of input sentences in our datasets.

4 Methods

4.1 Machine Learning Algorithms

We developed several machine learing algorithms, including Naive Bayes, Softmax Regression, and Support Vector Machines.

Naive Bayes is a generative learning algorithm that models $p(x \mid y)$ and $p(y)$, and then derive $p(y \mid x)$ using Bayes rule. It makes an assumption that every feature x_j's are conditionally independent given y, which means $p(x_1, ..., x_d \mid y) = \prod_{j=1}^{d} p(x_j \mid y)$.

In the Naive Bayes algorithm with a Bernoulli event model, we have $x_j = 1$ if the example x contain the j-th word of the dictionary; otherwise, $x_j = 0$. The parameters are $\phi_{i|y=k} = p(x_j = 1 \mid y = k)$ and $\phi_k = p(y = k)$. Maximum likelihood estimates with Laplace smoothing give us $\phi_{i|y=k} = \frac{1+\sum_{i=1}^{n} 1\{x_j^{(i)}=1 \wedge y^{(i)}=k\}}{2+\sum_{i=1}^{n} 1\{y^{(i)}=k\}}$ and $\phi_k = \frac{\sum_{i=1}^{n} 1\{y^{(i)}=k\}}{n}$.

$$
\begin{aligned}
p(x_j = 1 \mid y = k) &= \frac{\sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = k\}}{\sum_{i=1}^{n} 1\{y^{(i)} = k\}} \\
p(y = k) &= \frac{\sum_{i=1}^{n} 1\{y^{(i)} = k\}}{n}
\end{aligned}
$$

In the Naive Bayes algorithm with a Multinomial event model, we let V denote the dictionary, $x_j \in \{1, 2, \ldots, |V|\}$ denote the identity of the j-th word in the input, and d_i denote the number of words in the i-th training example. The parameters are $\phi_{l|y=k} = p(x_j = l \mid y = k)$ and $\phi_k = p(y = k)$. Maximum likelihood estimates with Laplace smoothing give us $\phi_{l|y=k} = \frac{1+\sum_{i=1}^{n}\sum_{j=1}^{d_i} 1\{x_j^{(i)}=l \wedge y^{(i)}=k\}}{|V|+\sum_{i=1}^{n} 1\{y^{(i)}=k\}d_i}$ and $\phi_k = \frac{\sum_{i=1}^{n} 1\{y^{(i)}=k\}}{n}$.

$$
\begin{aligned}
p(x_j = l \mid y = k) &= \frac{1 + \sum_{i=1}^{n}\sum_{j=1}^{d_i} 1\{x_j^{(i)} = l \wedge y^{(i)} = k\}}{|V| + \sum_{i=1}^{n} 1\{y^{(i)} = k\}d_i} \\
p(y = k) &= \frac{\sum_{i=1}^{n} 1\{y^{(i)} = k\}}{n}
\end{aligned}
$$

Softmax Regression is a generalization of the Logistic Regression algorithm for multi-class datasets. It uses the softmax function $\frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$ for $i = 1, ..., k$. Let n be the number of training examples, d be the number of the features, and k be the number of classes. The algorithm estimates $p(y \mid x; \theta) = h_\theta(x)$ with the i-th element being $\frac{exp(W_i^T x + b_i)}{\sum_{j=1}^{k} exp(W_j^T x + b_j)}$, where $W \in \mathbb{R}^{k \times d}$ and $b \in \mathbb{R}^k$ are the weights, and $x \in \mathbb{R}^d$ is the input. During training, the weights are updated through the equation $W = W - \alpha(\hat{y} - y)X$, where $\alpha \in \mathbb{R}$ is a learning rate, $\hat{y} \in \mathbb{R}^{k \times n}$ is the predicted probabilities, $y \in \mathbb{R}^{k \times n}$ is the one-hot encoding of thelabels, and $X \in \mathbb{R}^{n \times d}$ is the input matrix. The biases are updated through the equation $b = b - \alpha \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})$.

Support Vector Machine (SVM) [22] is an algorithm that fits the data through maximizing the margin of separation between different classes. It conducts inference through the following formula: $h_{w,b}(x) = g(w^T x + b)$, where w are the weights, b are the biases, and $g(z)$ is a function that outputs 1 if $z \geq 0$ and outputs -1 otherwise. It conducts optimization through the following objective: $min_{\gamma,w,b} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i$ so that $y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i$ for $i = 1, ..., n$ and that $\xi_i \geq 0$ for $i = 1, ..., n$

For this project, we used a multi-class variant of SVM using the one-vs-one scheme, where a model is created to distinguish every possible pair of two classes.

4.2 Deep Learning Models

In addition to the above traditional machine learning techniques, we developed two neural networks for intent classification.

Long short-term memory (LSTM) is a model frequently used to process sequences of data. It is based on the recurrent neural network (RNN), where the hidden state at the previous timestep, along with the new input at the current timestep, can influence the current hidden state. LSTM has a cell memory with three gates (forget gate, input gate, and output gate), to preserve information over long sequences. We used Bidirectional LSTM (Bi-LSTM), which involved two LSTM models trained in opposite directions so that more context is provided.

Algorithm	Features	Val Acc	Val. F1
Naive Bayes (Bernoulli)	Bag of Words	0.9783	0.9782
Naive Bayes (Multinomial)	Bag of Words	0.9739	0.9737
Softmax Regression	Bag of Words	**0.9846**	**0.9869**
Softmax Regression	GloVe	0.9725	0.9721
Softmax Regression	BERT	0.9667	0.9663
Support Vector Machines	Bag of Words	0.9290	0.9291
Support Vector Machines	GloVe	0.9797	0.9797
Support Vector Machines	BERT	**0.9812**	**0.9812**
LSTM	One-hot	**0.9812**	**0.9811**
LSTM	GloVe	0.9609	0.9610
LSTM	BERT	0.9696	0.9696
BERT	N/A	**0.9913**	**0.9913**
Naive Bayes (Bernoulli)	Bag of Words	0.9256	0.9247
Naive Bayes (Multinomial)	Bag of Words	0.9256	0.9236
Softmax Regression	Bag of Words	0.9587	0.9552
Softmax Regression	GloVe	0.9132	0.8940
Softmax Regression	BERT	**0.9669**	**0.9686**
Support Vector Machines	Bag of Words	0.9215	0.9079
Support Vector Machines	GloVe	0.9463	0.9397
Support Vector Machines	BERT	**0.9628**	**0.9568**
LSTM	One-hot	**0.9711**	**0.9692**
LSTM	GloVe	0.9132	0.8890
LSTM	BERT	0.8967	0.8960
BERT	N/A	**0.9959**	**0.9962**

Table 1: Performance comparison of different models
(upper: SNIPS, lower: ATIS)

Bidirectional Encoder Representations from Transformers (BERT) [7] is a state-of-the-art model for natural language processing. Its encoder acts as a feature extractor that provides vector embeddings for input sequences. The encoder leverages the Transformer architecture [15] to learn bidirectional representations from input texts by jointly conditioning on both left and right contexts in all layers. We took a BERT model pre-trained on a large corpus and fine-tuned its weights on our intent detection datasets.

Link to our project: https://github.com/bobyang9/NLU-for-Intent-Classification

5 Experiments and Results

5.1 Evaluation Metrics

Since we are working with a classification problem, we measure the performance of our algorithms using classification accuracy and F1 scores. The formula for classification accuracy is $\frac{TP+TN}{TP+TN+FP+FN}$, and the formula for F1 score is $2 * \frac{precision * recall}{precision + recall}$, also known as $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$, where TP = true positive, TN = true negative, FP = false positive, and FN = false negative. We use classification accuracy to see how well our model is performing overall and we use F1 score to see if we are correctly balancing precision and recall.

5.2 Performance of Machine Learning Algorithms on Validation Data

We implemented Naive Bayes and Softmax Regression from scratch, and used the scikit-learn implementation for SVM [14]. The Softmax Regression was run for 200 iterations with a learning rate of 0.001. We used the RBF kernel for SVM. We trained the models on ATIS and SNIPS with different feature extractors and then recorded their performances on validation sets in Tables 1 and 2.

The three algorithms had high performances on the SNIPS dataset. Softmax Regression with Bag of Words achieved 98.46% validation accuracy and 0.9869 validation F1 score. SVM with BERT encodings also performed well, achieving 98.12% validation accuracy and 0.9812 F1 score. Most of the other combinations achieved validation accuracies and F1 scores in the range of $96.5 - 98\%$. The same trends applies for ATIS. Softmax Regression with BERT embeddings had 96.69% validation accuracy and 0.9686 validation F1 score, while SVM with BERT embeddings had 96.28% accuracy and 0.9568 F1 score.

5.3 Performance of Deep Learning Models on Validation Data

We implemented the Bi-LSTM using TensorFlow [19] and Keras [23] using the architecture in Figure **??**. An additional BatchNorm [21] layer was added for the model trained ATIS because ATIS was less stable (The BatchNorm layer was not

Algorithm	Feature Extraction	Validation Accuracy	Validation F1 score
Naive Bayes (Bernoulli)	Bag of Words	0.9256	0.9247
Naive Bayes (Multinomial)	Bag of Words	0.9256	0.9236
Softmax Regression	Bag of Words	0.9587	0.9552
Softmax Regression	GloVe	0.9132	0.8940
Softmax Regression	BERT	**0.9669**	**0.9686**
Support Vector Machines	Bag of Words	0.9215	0.9079
Support Vector Machines	GloVe	0.9463	0.9397
Support Vector Machines	BERT	**0.9628**	**0.9568**
LSTM	One-hot	**0.9711**	**0.9692**
LSTM	GloVe	0.9132	0.8890
LSTM	BERT	0.8967	0.8960
BERT	N/A	**0.9959**	**0.9962**

Table 2: Performance comparison of machine learning models on ATIS.

Algorithm	Feature Extraction	Validation Accuracy	Validation F1 score
LSTM	One-hot	**0.9812**	**0.9811**
LSTM	GloVe	0.9609	0.9610
LSTM	BERT	0.9696	0.9696
BERT	N/A	**0.9913**	**0.9913**

Table 3: Performance comparison of deep learning models on SNIPS.

included in the model for SNIPS as it hurt performance). After noticing that the model was overfitting, we added a Dropout [20] layer with 50% probability to curb the overfitting. We trained the LSTM until early stopping defined by the training loss not improving for three epochs. We used the Adam optimization algorithm with a learning rate of 0.0001 and a batch size of 64 (a batch size that fits into memory).

For the BERT implementation, we developed a BERT neural network using PyTorch [17] and Hugging Face [18], and then fine-tuned its pre-trained weights on our datasets. The model's encoder receives input sentences and outputs a 768 dimentional encoding vector. For this classification task, we added a fully connected layer and a softmax layer following the encoder. BERT is a bigger model, so we used a smaller batch size of 16. The optimizer and the learning rate remain the same.

We trained Bi-LSTM and BERT models on ATIS and SNIPS and recorded their validation performances in Tables 3 and 4. Both models had high performances on SNIPS. BERT achieved 99.13% accuracy and 0.9913 F1 score, while LSTM with one-hot word encodings achieved 98.12% accuracy and 0.9811 F1 score. Similar trends were observed for ATIS. BERT achieved 99.59% accuracy and 0.9962 F1 score, while LSTM with one-hot word encodings achieved 97.11% accuracy and 0.9692 F1 score.

5.4 Analysis

We found that BERT was the highest performer among the models that we tried. The reason might be that its Transformer architecture had the attention mechanism that allows it to identify the important parts of the text (even those very far away) which guided it to make more informed classifications. From the confusion matrix in Figure 1, we can see that BERT had a good performance across all labels and did not get confused by any pairs of classes.

However, we found that Softmax Regression with Bag of Words feature extraction, a much simpler model, actually came pretty close in performance. In fact, it performed better than Bi-LSTM, which is considered a complex deep learning model. Another notable point is that other algorithms like Naive Bayes and SVMs also had great performance despite their simplicity.

One example that BERT predicted incorrectly was *"i'm going to leave philadelphia and i want to go to san francisco and i want to fly first class american and i want a stop in dallas can you please tell me what type of aircraft you will be flying"*. Its ground truth label was *atis_aircraft*, but the system predicted *atis_flight*. The algorithm was perhaps misled by the preponderance of destination terms like *"philadelphia"*, *"san francisco"*, and *"dallas"*, as well as many phrases related to booking a flight, including *"i want a stop in ..."*, *"i want to fly first class ..."*, etc. Only the last segment (*"can you please tell me what type of **aircraft** you will be flying"*) had to do with aircraft, so the algorithm may have believed that the category was more likely to be *atis_flight*.

Algorithm	Feature Extraction	Validation Accuracy	Validation F1 score
LSTM	One-hot	**0.9711**	**0.9692**
LSTM	GloVe	0.9132	0.8890
LSTM	BERT	0.8967	0.8960
BERT	N/A	**0.9959**	**0.9962**

Table 4: Performance comparison of deep learning models on ATIS.

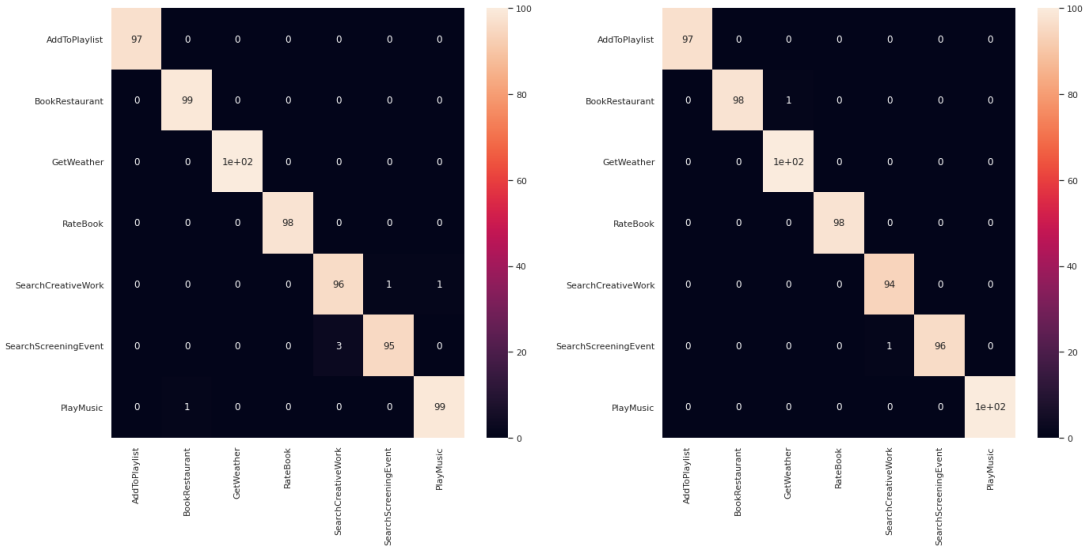

Figure 1: Confusion Matrix for BERT (left) and LSTM (right).

Test Dataset	Test Accuracy	Test F1 score
SNIPS	0.9900	0.9900
ATIS	0.9938	0.9951

Table 5: Performance on test data.

Another incorrect prediction was *"what is the schedule of ground transportation from the airport in philadelphia into downtown"*. The program outputted *atis_flight_time* instead of the correct label *atis_ground_service*. The algorithm may have put too much weight on *"schedule"*, so it believed that the text was about flight time, while the actual emphasis was on *"ground transportation"*.

Another interesting pattern is that fine-tuning the BERT model leads to a better outcome than combining a fixed pre-trained BERT or GloVe encoder with a classifier. The reason might be that the distribution of words in the pre-trained English corpora and in our intent datasets are different, because the ATIS and SNIPS examples only belong to limited domains. Therefore, the fine-tuning process updates the weights in BERT encoder to adapt better to our training sets.

5.5 Performance on Test Data

Finally, we evaluated our best model, the fine-tuned BERT, on SNIPS and ATIS test sets and recorded the performance in table 5. The model performed very well, achieving 99.00% test accuracy and 0.9900 test F1 score on SNIPS and 99.38% test accuracy and 0.9951 test F1 score on ATIS.

6 Conclusion and Future Work

In summary, we developed several machine learning and deep learning algorithms for intent detection. The best model is a fine-tuned BERT, which attains 99+% accuracy scores and 0.99+ F1 scores on the ATIS and SNIPS datasets. In the future, we will perform hyperparameter tuning on Softmax Regression, SVM, and LSTM to narrow their performance gap with BERT. We also want to evaluate our classifiers on CLINC150 [4], a much bigger and more challenging dataset. Another interesting idea is to develop algorithms for joint intent prediction and slot filling tasks, and the deploy the models in a real chatbot system.

7 Contributions

Tai implemented Naive Bayes, BERT, and feature extraction using GloVe and BERT. Robert implemented Softmax Regression, SVM, and LSTM. Robert and Tai worked together in implementing the data pipeline, writing the report and making the poster.

References

[1] Zhang, Z., Takanobu, R., Zhu, Q., Huang, M., & Zhu, X. (2020). Recent advances and challenges in task-oriented dialog systems. Science China Technological Sciences, 1-17.

[2] Hemphill, C. T., Godfrey, J. J., & Doddington, G. R. (1990). The ATIS spoken language systems pilot corpus. In Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990.

[3] Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., ... & Primet, M. (2018). Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. arXiv preprint arXiv:1805.10190.

[4] Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., ... & Mars, J. (2019). An evaluation dataset for intent classification and out-of-scope prediction. arXiv preprint arXiv:1909.02027.

[5] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[6] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[7] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[8] Peng, B., & Yao, K. (2015). Recurrent neural networks with external memory for language understanding. arXiv preprint arXiv:1506.00195.

[9] Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., ... & Zweig, G. (2014). Using recurrent neural networks for slot filling in spoken language understanding. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 23(3), 530-539.

[10] Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., ... & Zweig, G. (2014). Using recurrent neural networks for slot filling in spoken language understanding. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 23(3), 530-539.

[11] Niu, P., Chen, Z., & Song, M. (2019). A novel bi-directional interrelated model for joint intent detection and slot filling. arXiv preprint arXiv:1907.00390.

[12] Wang, Y., Shen, Y., & Jin, H. (2018). A bi-model based rnn semantic frame parsing model for intent detection and slot filling. arXiv preprint arXiv:1812.10235.

[13] Qin, L., Che, W., Li, Y., Wen, H., & Liu, T. (2019). A stack-propagation framework with token-level intent detection for spoken language understanding. arXiv preprint arXiv:1909.02188.

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[16] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Desmaison, A. (2019). Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems (pp. 8026-8037).

[18] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Brew, J. (2019). HuggingFace's Transformers: State-of-the-art Natural Language Processing. ArXiv, arXiv-1910.

[19] Abadi, M., Agarwal, A., Barham, B., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2015). Tensorflow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

[20] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

[21] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[22] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

[23] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras