Technische Universität Berlin

# Multi-domain Sentiment Analysis with an Active Learning Mechanism

by

**Nabil Douss**

**Matriculation Number 386360**

A thesis submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
Service-centric Networking

Master's Thesis

June 17, 2022

Supervised by:
Prof. Dr. Axel Küpper

Assistant supervisor:
Prof. Dr.-Ing. Sebastian Möller

SNET
SERVICE-CENTRIC NETWORKING

## Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, June 17, 2022 Nabil Douss

# Abstract

An enormous amount of data is daily generated and interchanged between different kinds of services. Some of this data is produced by users, including social media posts, product reviews or comments, which usually convey an emotion or an opinion. This thesis propose three different model architectures for binary sentiment analysis on textual data. The proposed models FT-HAN, FT-HAN-BiLSTM and FT-HAN-BiLSTM-AE aim to solve the multi-domain sentiment analysis task. They extend the Hierarchical Attention Networks (HAN) architecture [1] while incorporating FastText word embeddings [2].

This thesis showed that using FastText embeddings trained in an unsupervised fashion on the available data and adding a BiLSTM layer before the sentence-level attention layer of the HAN architecture improves the performance of the model on the chosen datasets. This work also proves the benefits of using Active Learning mechanisms to train the domain-specific classifiers. The FT-HAN-BiLSTM model proved to be comparable to current state-of-the-art classifiers. The general classifier as well as the domain-specific models produced by Active Learning yield very good results.

Especially in the case of sentiment classification, there are a lot of unlabeled data in the real world, which makes it unreasonable to train and test a model using a great portion of a carefully constructed dataset. In real applications, domains are unbalanced. Some domains have a lot less labeled data than other, and manually labeling them would require domain experts, and a lot of time, which can induce tremendous costs. The Transfer Learning approach used in this thesis assumes that a lot of the available data is unlabeled by only selecting a portion of the domain-specific training-set. The data sampling was done by two approaches. In the first one, the data is randomly sampled from the data pool, while the second method applied Active Learning to actively query the most informative observations.

First, using the three architectures, three general classifiers were trained on all domains. Second, the FT-HAN-BiLSTM model was selected to transfer knowledge to the domain-specific classifiers, which used the trained weights by the general model as a starting point. Three different approaches were used, and the experiments showed that the sentence-level transfer learning approach yields better results. In this approach, the transferred weights to the word-level layers do not get updated throughout the training, as opposed to the weights of the sentence-level layers. Third, the general FT-HAN-BiLSTM classifier and the domain-specific models using Active Learning with Entropy Sampling and outlier detection were compared to other state-of-the-art models [3][4][5][6][7].

# Zusammenfassung

Täglich wird eine enorme Menge an Daten erzeugt und zwischen verschiedenen Diensten ausgetauscht. Ein Teil dieser Daten wird von Nutzern erzeugt, darunter Beiträge in sozialen Medien, Produktbewertungen oder Kommentare, die in der Regel eine Emotion oder eine Meinung vermitteln. In dieser Arbeit werden drei verschiedene Modellarchitekturen für die binäre Stimmungsanalyse ("Sentiment Analysis") von Textdaten vorgeschlagen. Die vorgeschlagenen Modelle FT-HAN, FT-HAN-BiLSTM und FT-HAN-BiLSTM-AE zielen darauf ab, die Multidomäne-Sentiment-Analyse zu lösen. Sie erweitern die Hierarchical Attention Networks (HAN)-Architektur [1], indem sie FastText-Worteinbettungen ("Word Embeddings") [2] einbeziehen.

In dieser Arbeit wurde gezeigt, dass die Verwendung von FastText-Worteinbettungen, die auf den verfügbaren Daten unüberwacht trainiert wurden, und das Hinzufügen einer BiLSTM-Schicht vor der satzbezogenen "Attention"-schicht der HAN-Architektur die Leistung des Modells auf den ausgewählten Datensätzen verbessert. Diese Arbeit beweist auch die Vorteile der Verwendung von Active Learning-Mechanismen zum Trainieren der domänenspezifischen Klassifikatoren. Das FT-HAN-BiLSTM-Modell erwies sich als vergleichbar mit aktuellen State-of-the-Art-Klassifikatoren. Sowohl der allgemeine Klassifikator als auch die von Active Learning erstellten domänenspezifischen Modelle lieferten sehr gute Ergebnisse.

Vor allem bei der Klassifizierung von Gefühlen gibt es in der realen Welt viele unbeschriftete Daten, so dass es nicht sinnvoll ist, ein Modell mit einem großen Teil eines sorgfältig erstellten Datensatzes zu trainieren und zu testen. In realen Anwendungen sind die Domänen unausgewogen. Einige Domänen haben viel weniger beschriftete Daten als andere, und die manuelle Beschriftung würde Domänenexperten und eine Menge Zeit erfordern, was enorme Kosten verursachen kann. Der in dieser Arbeit verwendete Transfer-Learning-Ansatz geht davon aus, dass ein Großteil der verfügbaren Daten nicht beschriftet ist, indem nur ein Teil des domänenspezifischen Trainingssatzes ausgewählt wird. Die Datenauswahl wurde mit zwei Ansätzen durchgeführt. Bei der ersten Methode werden die Daten nach dem Zufallsprinzip aus dem Datenpool gezogen, während bei der zweiten Methode Active Learning eingesetzt wird, um die informativsten Beobachtungen aktiv abzufragen.

Erstens wurden unter Verwendung der drei Architekturen drei allgemeine Klassifikatoren für alle Domänen trainiert. Zweitens wurde das FT-HAN-BiLSTM-Modell ausgewählt, um Wissen auf die domänenspezifischen Klassifikatoren zu übertragen, die die trainierten Gewichte des allgemeinen Modells als Ausgangspunkt verwendeten. Es wurden drei verschiedene Ansätze verwendet, und die Experimente zeigten, dass der Ansatz des Transferlernens auf Satzebene bessere Ergebnisse lieferte. Bei diesem Ansatz werden die auf die Wortschichten übertragenen Gewichte während des gesamten Trainings nicht aktualisiert, im Gegensatz zu den Gewichten der Satzschichten. Drittens wurden der allgemeine FT-HAN-BiLSTM-Klassifikator

und die domänenspezifischen Modelle, die Active Learning mit Entropy Sampling und Ausreißererkennung verwenden, mit anderen State-of-the-Art-Modellen verglichen [3][4][5][6][7].

# Contents

# 1 Introduction

Sentiment Analysis is a research area of Natural Language Processing (NLP), which aims to automatically extract the sentiments or ideas transmitted by these users. In the face of such a large flux of data, it becomes challenging to solve this task, especially, when the underlying generated data is mostly unlabelled. It is then necessary to manually label them, in order to feed a sentiment analysis model a descent amount of labelled training data, and hence, to correctly learn from extracted features and solve the specific task. However, manual labelling requires expert domain knowledge, time and resources, which can become very expensive.

Aside from the scarcity of labelled data, some of the challenges of sentiment analysis lies in the inherent nuance in the human languages. Since, two sentences that have the same words can convey different meanings.
Moreover, a lot of words have multiple definitions, which can make the sentiment extraction task more challenging. Even in the case of words that only have one definition, they can convey different sentiments, depending on the context. For example the words "easy" or "expected" can transmit a positive sentiment in a review of a baby product, while they communicate a negative emotion in the case of a movie or a book review.

One of the main motivations of sentiment analysis is to know a target audience's opinion on a subject by analyzing a large amount of textual and auditory data. This can improve the customer acquisition and lower the marketing turnover for a business's marketing team, that aims to better reach their (potential) customers. Moreover, and in the same context, sentiment analysis can also help businesses better understand customer feedback on products and their target audience's needs.

Multi-domain sentiment analysis is a special case of this research area, where the available data originate from multiple domains. It aims at improving the overall performance of a model by "fusing training data from multiple domains" [8]. Usually, the fused training set is used to learn general features common to all domains, while the domain-specific data is exploited to learn the domain-specific ones. Some designed models use multi-task learning [9] to solve the multi-domain sentiment analysis problem. It consists in learning general features simultaneously through layers shared between all domains, and domain-specific features through specific layers that extract domain-dependent knowledge.

This thesis proposes three model architecture to solve the multi-domain sentiment analysis problem by using Active Learning mechanisms. The proposed architectures extend the Hierarchical Attention Networks (HAN) solution proposed by Yang et al. [1]. A lot of state-of-the-art models aim at solving the multi-domain sentiment analysis task [3][4][5][6][10], which are detailed in the next chapter. However, there is a little research on using Active Learning in this area. The purpose of this thesis is to present a novel architecture for multi-domain sentiment analysis, and to examine the effects of using Active Learning on the domain-specific tasks.

Chapter 2 details the state-of-the-art, including the architectures used and the main contributions of the authors. Chapter 3 further explains the concept applied in this thesis and gives a scientific background on the different methods and techniques. Chapter 4 lists the actual project implementation and the libraries and frameworks used. Chapter 5 is dedicated to the evaluation of the proposed models, including the conducted experiments and a comparison with the state-of-the-art. Finally, chapter 6 contains the conclusion.

# 2 Related Work

In their paper "Multi-domain Sentiment Classification", Li and Zong proposed two approaches to improve the performance of sentiment classification model through fusing training data from multiple domains [8].
They consider two methods of fusion in their experiments (feature-level and classifier-level).

- Feature-Level fusion approach:
  learn the global sentiment information (some terms occur in all domains and convey the same sentiment).
  1. Pool the training data from all domains for training.
  2. Use a common set of terms to construct a uniform feature vector $x'$.
  3. Train a predictor using all training data ($x'$ + all training data).
- Classifier-Level fusion approach:
  1. Train multiple base classifiers.
  2. Combine the base classifiers (e.g., by meta-learning (ML))

The Empirical studies from their research show that the classifier-level approach generally outperforms the feature approach. Compared to single domain classification, multi-domain classification with the classifier-level approach can consistently achieve much better results [8].

In 2017, Liu et al. proposed a new adversarial multi-task learning framework to solve the multi-domain sentiment analysis problem. They published their paper "Adversarial Multi-task Learning for Text Classification", where they addressed a common problem for multi-task learning. This type of learning usually utilizes shared layers to learn task-invariant and general features. "However, in most existing approaches, the extracted shared features are prone to be contaminated by task-specific features or the noise brought by other tasks" [3]. The authors try to solve this problem by introducing a framework, in which the domain specific and general features are learned non-redundantly, therefore separating between the shared and task-dependent features of different tasks.

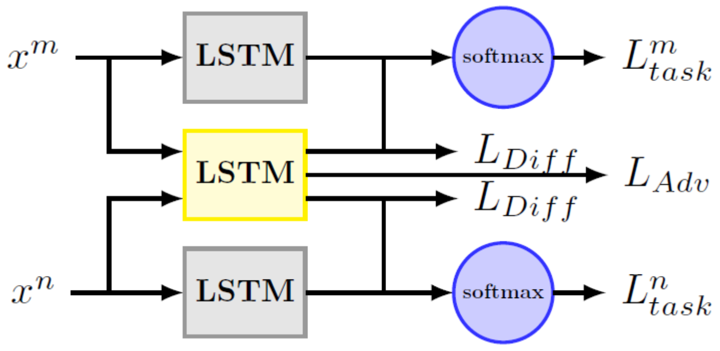

**Figure 2.1:** Adversarial shared-private model. Framework for ASP-MTL [6].

The proposed framework applies adversarial training and orthogonality constraints. "The adversarial training is used to ensure that the shared feature space simply contains common and task-invariant information, while the orthogonality constraint is used to eliminate redundant features from the private and shared spaces" [3].

In the paper "Same Representation, Different Attentions: Shareable Sentence Representation Learning from Multiple Tasks" of 2018, Zheng et al. proposed a new scheme of information sharing for multi-task learning [5].
The authors used a model, where the data from all domains share the same sentence representation. To extract the domain-specific feature, they use attention and a task-dependent query vector to select the domain-specific information from the shared sentence representation. The result of the attention layers is passed then to a softmax output layer to predict the sentiment. Their contributions consist of [5]:

1. A new information sharing scheme for multitask learning. As a side effect, the model can be easily visualized and shows what specific parts of the sentence are focused in different tasks.
2. The scheme can learn a shareable generic sentence representation, which can be easily transferred to other tasks. The shareable sentence representation can also be improved by the auxiliary tasks, such as POS Tagging and Chunking.
3. Experiments show that the proposed model is space efficient and converges quickly.

Liu et. al proposed a new model in their paper "Learning Domain Representation for Multi-Domain Sentiment Classification (2018)" [4]. The model learns domain-specific input representations by using a bidirectional LSTM to learn a general sentence-level representation. Adversarial training is used on the Bi-LSTM representation, which is then mapped into a domain-specific representation by attention over the input sentence using explicitly learned domain descriptors. In addition to the domain descriptors, they further introduce a memory network for explicitly representing domain knowledge, which can offer useful background context. This model adopts self-attention to learn a domain-specific descriptor vector and uses BiLSTM to learn the general sentence-level vector. Then, the general and domain-specific are concatenated into one vector.

In 2019, Cai and Wan proposed a novel model in their paper "Multi-Domain Sentiment Classification Based on Domain-Aware Embedding and Attention" [6]. Their goal was to leverage data from different domains to improve the classification performance in each domain.
Their approach can be summarized as follows:

1. Use BiLSTM for domain classification and extracts domain-specific features
2. Combine them with general word embeddings to form domain-aware word embeddings.
3. These embeddings are fed into another BiLSTM to extract sentence features
4. The domain-aware attention mechanism is used for selecting significant features, by using the domain aware sentence representation as the query vector.

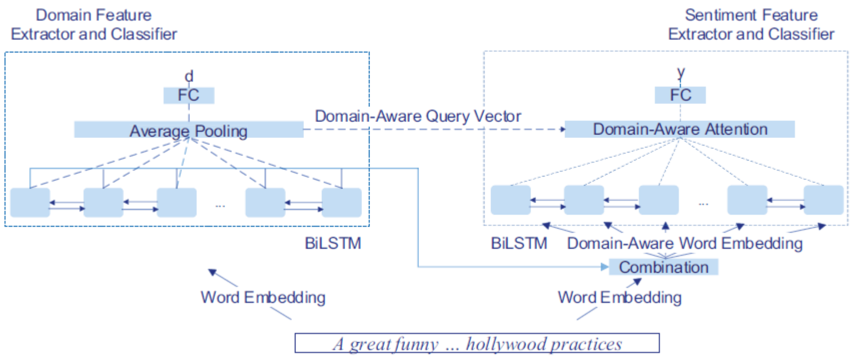
Domain Feature Extractor and Classifier
Sentiment Feature Extractor and Classifier
d
FC
y
FC
Average Pooling
Domain-Aware Query Vector
Domain-Aware Attention
BiLSTM
BiLSTM
Domain-Aware Word Embedding
Combination
Word Embedding
Word Embedding
A great funny ... hollywood practices

**Figure 2.2:** Overview of the model proposed by Cai and Wan [6].

The paper proposes a novel completely shared neural model based on domain-aware word embeddings and domain aware attention mechanism to make use of training data among all domains for multi-domain sentiment classification. The model does not only focus on the significant words but also distinguish their sentiment polarity with the help of a domain classifier [6].
The authors conduct experiments on datasets with 16 different domains, which are also used in this thesis. The results of multi-domain sentiment classification show the effectiveness of the model. The authors perform experiment on cross-domain sentiment classification and knowledge transferring. Results reveal the generalization ability and transferability of the model [6].

The paper "Multi-domain Transfer Learning for Text Classification" by Su et al. proposes a generic dual channels multi-task learning framework (GLR-MTL) for multi-domain text classification, which can capture global-shared, local-shared, and private features simultaneously [10].

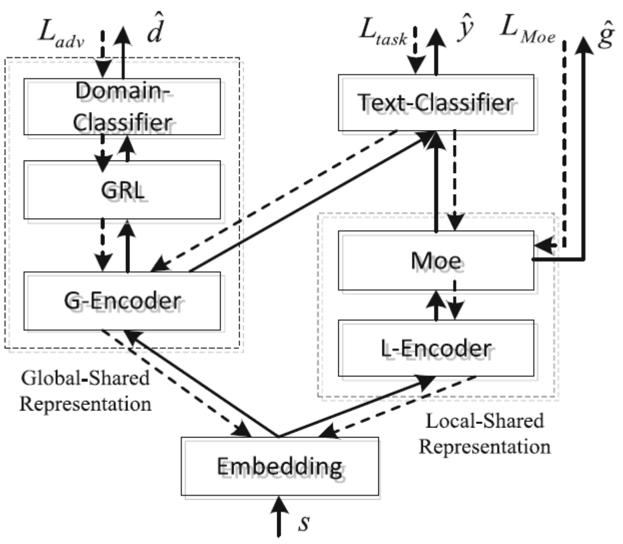

**Figure 2.3:** Overall framework for GLR-MTL[10].

The model consists of four parts: embedding layer, global-shared representation network, local-shared representation network and text classification layer.
Given an input text, the model first concatenates its global-shared representation and local-shared representation into a completed representation, which is then passed into the text classification module, which has a fully connected layer followed by a softmax non-linear layer that predicts the probability distribution over classes [10].

# 3 Concept and Design

## 3.1 (Multi-domain) Sentiment Analysis

Sentiment analysis is a current research area of Natural Language Processing (NLP). It aims to extract emotions from textual or audible data.
This thesis focuses on a binary classification of textual data, which means that the implemented models classify data into a positive or a negative class.

Multi-domain sentiment classification is a special case of this research area where the model learns using data from different domains and sources (e.g., movie reviews, social media posts, ...). This is more challenging, because data drawn from distinct domains usually have different distributions and a characteristic vocabulary. Sometimes one term can convey a positive sentiment in a domain, but a negative emotion in another one.
The most common approach for this problem is to learn general and domain-specific features simultaneously (e.g., using Multi-task learning). The implemented model would have shared and domain-specific layers. The shared ones aim to obtain general knowledge from the datasets, while the other ones try to learn features characteristic of one domain.

Instead of Multi-task learning, another approach of transfer learning is employed in the context of this thesis. A general classifier is trained using data drawn from all domains, which was employed to transfer knowledge to domain-specific models. The general features learned were used as a starting point for training by transmitting the pre-trained weights. The transfer learning method is explained in more details in Section 3.10.

## 3.2 Long Short Term Memory Networks (LSTM)

LSTM [11] is a kind of recurrent neural network (RNN) architecture proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. Like most recurrent neural networks, the LSTM network is universal in the sense that, given a sufficient number of network elements, it can perform any computation that a conventional computer is capable of, which requires an appropriate matrix of weights.
Unlike traditional RNNs, LSTM-networks are well adapted to learning on the tasks of classification, processing and prediction of time series in cases when important parts are separated by time lags with undefined duration and boundaries.

In a traditional RNN, the long-term gradients which are back-propagated can "vanish", which leads to a loss of information. This hinders the detection of long-term relationships between distant parts of the input sequence [12].

Instead of a single neuronal function in the LSTM module, there are four that interact with each other. One LSTM module contains three gates (The input gate, the forget gate and the output gate) and one inner cell.
At a time-step $t$, the functions of the forget gate $f_t$, the input gate $i_t$, the output gate $o_t$, the cell $c_t$ and the hidden state $h_t$ are defined in this way:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.1}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.2}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.3}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3.4}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.5}$$

Where $x_t$ is the input at position $t$, $\odot$ is the element-wise matrix multiplication, $\sigma$ is the sigmoid function, and $W$, $U$ and $b$ are weights and biases proper to the respective gate or cell.
The hidden state $h_t$ and the cell value $c_t$ are passed on to the next LSTM unit.

Bidirectional LSTM (BiLSTM) is a network that consists of two LSTMs. The first one processes the input in a forward direction, and the second in a backwards direction. In this type of networks, forward propagation is conducted twice, once for the forward cells and once for the backward cells. The activation functions of both LSTM networks would be considered to calculate the hidden state $h_t$ at any step $t$. BiLSTMs effectively improve the context available to the process [13].

## 3.3 Embeddings

Word embeddings are learned vector representations for textual data. They map words from a vocabulary to a multidimensional feature space. A good model would project words with similar meaning or often appearing in the same context to vectors having a close distance in the feature space.
There are a lot of techniques for word embeddings. Some of them are listed below:

- Word2Vec is a technique published in 2013, which learns word embeddings from large textual data. The tool takes a text corpus as input and produces the word vectors as output. It is one of the first tools to implement the skip-gram and the continuous bag of words architectures defined in the subsection below [14].
- FastText: a library created by Facebook's AI Research (FAIR) lab and initially released in 2015. It is used for learning word embeddings from textual data and for text classification [2].

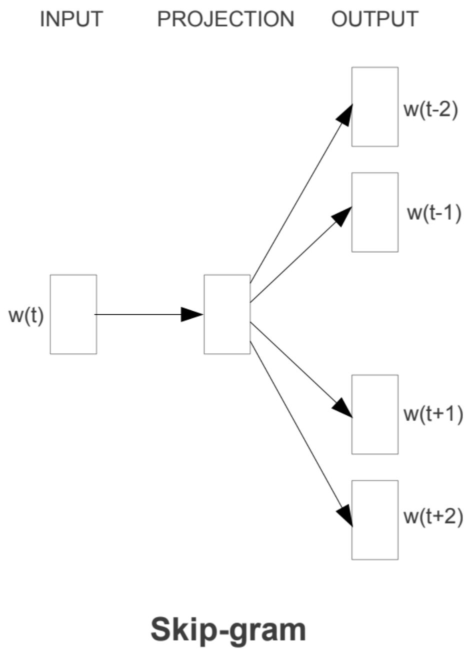

**Figure 3.1:** The Skip-gram Architecture [14].

### 3.3.1 Embeddings Architectures

#### 3.3.1.1 Skip-gram

The skip-gram model learns by considering a word from a given sentence. In other words, to train the neural network a word $W$ is extracted from a sentence $s$, which will be the input to the neural network $w(t)$. The rest of the sentence will be the output $\{w(t-2),\ w(t-1),\ w(t+1),\ w(t+2)\}$.
In other words, each word is fed into a classifier with a continuous projection layer, which predicts words within a specified range before and after the current word. Less weight is assigned to the distant words, since they are frequently less connected to the present word than those adjacent to it [14].

#### 3.3.1.2 Continuous Bag of Words (CBOW)

The Continuous Bag-of-words model is the inverse image of the skip-gram model. From a context $\{w(t-2),\ w(t-1),\ w(t+1),\ w(t+2)\}$, it learns to recognize a word $w(t)$, the set $\{w(t-2),\ w(t-1),\ w(t),\ w(t+1),\ w(t+2)\}$ form a sentence $s$ of the language [14].

## 3.4 Attention

The attention mechanism in deep learning is a technique used by a lot of state of the art models to enhance their performance. The technique, like its name suggests, consists of giving greater importance to some parts of the input data as opposed to others.

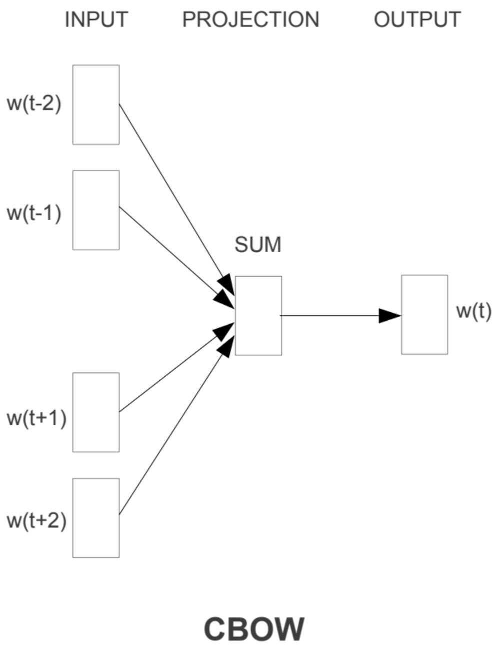

**Figure 3.2:** The CBOW Architecture [14].

The mechanism was first introduced by Bahdanau et al. in 2014, in the context of an encoder-decoder network, to address the problem of having a fixed-length vector produced by the encoder. This is expected to be particularly difficult for long complicated sequences, because the fixed-length vector carries the responsibility of encoding the complete "meaning" of the input sequence, regardless of its length [15].

Bahdanau et al.'s approach divides the attention mechanism into three consecutive steps:

1. **Computing the alignment scores:** At each position $t$, and for each hidden state $h_i$, the alignment score is calculated using the following equation

$$e_{t,i} = a(s_{t-1}, h_i) \tag{3.6}$$

   $s_{t-1}$ is the output of the decoder at the previous step $t-1$ and $a()$ is a function that can be represented using a feed-forward network.

2. **Computing the weights:** The weights are computing using a $softmax$ function applied to the previously calculated alignment scores

$$\alpha_{t,i} = softmax(e_{t,i}) \tag{3.7}$$

3. **Computing the context vector:** At each position $t$, the vector fed to the decoder is calculated as the weighted sum of all hidden states $h_i$.

$$c_t = \sum_i \alpha_{t,i} h_i \tag{3.8}$$

The general attention model re-defines the three previously enumerated steps using three components, which are the queries Q, the keys K and the values V.
The decoder output $s_{t-1}$ is analogous to a query $q$, the hidden state $h_i$ in the first equation to a key $k_i$ and the hidden state in the last equation to a value $v_i$. It also uses a dot product as the function $a()$ in the alignment scores equation.
Finally, the attention mechanism redefined the three equations as follow:

$$e_{q,k_i} = q.k_i \tag{3.9}$$

$$\alpha_{q,k_i} = softmax(e_{q,k_i}) \tag{3.10}$$

$$c_t = Attention(q, K, V) = \sum_i \alpha_{q,k_i} v_i \tag{3.11}$$

### 3.4.1 Self-Attention

The particularity of the self-attention in comparison to the general definition of attention comes in the fact that in a self-attention layer all of the keys, values and queries come from the same input. The self-attention mechanism allows the inputs to interact with each other. The outputs are aggregates of these interactions and attention scores.
A self-attention module compares each word in the sentence to every other word, including itself. It then recalculate the weights of the word embeddings of each word to account for contextual significance. It accepts n word embeddings without context and returns n word embeddings with context [16].

### 3.4.2 Hierarchical Attention Networks

Hierarchical Attention Networks is a model for document classification proposed by Yang et al. in 2016.
The model features a hierarchical structure comparable to document hierarchies and two levels of attention mechanisms used at the word and sentence levels, allowing it to focus differently to more and less significant input parts while generating the document representation [1].
The most important parts of the hierarchical attention network as depicted in the figure 3.3, are the word-encoder, word attention, sentence encoder and sentence attention. In the encoder parts, the GRU [15] is used with two types of gates; the reset gate $r$ and the update gate $z$ to encode the input sequences. The output of the GRU is calculated according to the following steps [15]:

1. Calculate the update gate at position $t$ using the input sequence at position $t$, $x_t$, and the

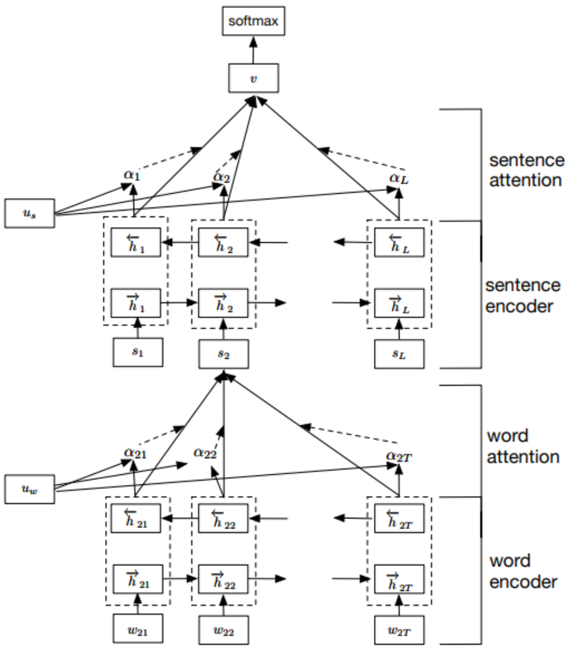

**Figure 3.3:** Hierarchical Attention Network [1].

previous hidden state $h_{t-1}$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{3.12}$$

where $W_z$ and $U_z$ are weights proper to the update gate $z$, and $b_z$ is its own bias.

2. Calculate the reset gate at position $t$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{3.13}$$

Analogous to the update gate $z$, $W_r$, $U_r$ and $b_r$ are proper weights and bias to the reset gate $r$.

3. Calculate the candidate state $\hat{h_t}$: by using the reset gate to store the relevant information from previous positions.

$$\hat{h_t} = tanh(W_h x_t + r_t \odot U_h h_{t-1} + b_h) \tag{3.14}$$

Where $\odot$ is the element-wise product function.

4. Compute the new state $h_t$ which holds information for the current unit and gets passed down to the network

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h_t} \tag{3.15}$$

For the attention parts, the output is calculated as follows [1]:

- On the word-level:

$$u_{it} = tanh(W_w h_{it} + b_w) \tag{3.16}$$

$$\alpha_{it} = \frac{\exp(u_{it}^\intercal u_w)}{\sum_t \exp(u_{it}^\intercal u_w)} \tag{3.17}$$

$$s_i = \sum_t \alpha_{it} h_{it} \tag{3.18}$$

Where $h_{it}$ is the hidden state produced by the GRU of the word-encoder from the $t$-th word of the $i$-th sentence. $W_w$, $u_w$, and $b_w$ are weights and biases specific to the attention layer.

- On the sentence-level:

$$u_i = tanh(W_s h_i + b_s) \tag{3.19}$$

$$\alpha_i = \frac{\exp(u_i^\intercal u_s)}{\sum_i \exp(u_i^\intercal u_s)} \tag{3.20}$$

$$v = \sum_i \alpha_i h_i \tag{3.21}$$

Where $h_i$ is the hidden state produced by the GRU of the sentence-encoder from the $i$-th output of the word-level attention layer, which corresponds to the $i$-th sentence.

## 3.5 Auto-encoder

An auto-encoder is an artificial neural network used for unsupervised learning of features [17]. The goal of an auto-encoder is to learn a representation (encoding) of a data set, usually with the goal of reducing the dimensionality of the data set or de-noising it.
The concept of an auto-encoder has become more widely used for generative model learning [18].
An auto-encoder consists always of two parts, the encoder and the decoder. The encoder maps the input to a feature space, which usually has a lower dimension than the input space, while the decoder reconstructs the input using the encoded data.
The encoder and the decoder can be defined as functions $\phi$ and $\psi$ such that:

$$\begin{aligned}
&\phi : \mathcal{X} \rightarrow \mathcal{F} \\
&\psi : \mathcal{F} \rightarrow \mathcal{X} \\
&\phi, \psi : \underset{\phi,\psi}{argmin}\ \mathcal{L}(\mathcal{X}, \psi(\phi(\mathcal{X})))
\end{aligned}$$

Where $\mathcal{X}$ is the input space, $\mathcal{F}$ is the feature space, and $\mathcal{L}$ is a loss function, which calculates the dissimilarity between $\mathcal{X}$ and its reconstruction $\psi(\phi(\mathcal{X}))$.

## 3.6 Loss Functions

The term objective function is used in mathematical optimization and operations research to designate a function that serves as a criterion for determining the best solution to an optimization problem. In concrete terms, it associates a value with an instance of an optimization problem. The goal of the optimization problem is then to minimize or maximize this function until the optimum is reached.

The loss function, or cost function is a function which represents a cost associated with an instance of the optimization problem. The solution to the problem should be an instance which minimizes it.

In the context of machine learning, the loss function calculates the dissimilarity or the deviation of a prediction from the actual true value. Some common loss function are:

- Mean square error (MSE) is measured as the average of squared difference between predictions and actual observations. It's only concerned with the average magnitude of error across all $n$ observations.

$$MSE = \frac{\sum_i (\hat{y} - y)^2}{n} \tag{3.22}$$

- 0-1 Loss function returns 1 if the current observation is not equal the prediction and 0, otherwise. This loss function does not measure the deviation of the calculated prediction from the actual true value, and only returns 0 if the values are identical.

$$\mathcal{L}(\hat{y}, y) = \mathcal{I}(\hat{y} \neq y) \tag{3.23}$$

  Where $\mathcal{I}$ is the indicator function.

- Cross Entropy: It assesses the performance of a classification model, the output of which is a probability value ranging from 0 to 1. Cross-entropy loss increases when the projected probability diverges from the actual label.
  - Binary cross-entropy:

$$\mathcal{L}(\hat{y}, y) = ylog(\hat{y}) + (1 - y)log(1 - \hat{y}) \tag{3.24}$$

    where $y$ is the true label (0 or 1) and $\hat{y}$ is the predicted probability.
  - Categorical cross-entropy (for $N$ classes, where $N > 2$):

$$\mathcal{L}(\hat{y}, y) = \sum_i^N y_i log(\hat{y}_i) \tag{3.25}$$

    Where $y_i$ is a binary indicator that equals to 1, only if the $i$-th class is the correct classification of the current observation, and $\hat{y}_i$ is the predicted probability that the observation corresponds to the $i$-th class.

## 3.7 Optimizers

The weights and parameters of machine learning model are tweaked and updated during the training phase in order to minimize the chosen loss function and make the predictions as accurate and optimized as feasible.
An optimizer updates the model in response to the output of the loss function, with the goal of decreasing it. The updated weights and biases should result in a lower loss value. Normally, a learning rate $\eta$ is initially chosen and fed to the optimizer, which dictates the magnitude of the step taken during the parameters' update. Taking large steps can affect the algorithm in such a way, that it does not converge.

Some of the widely used optimizers within the data science community are enumerated in the next subsections.

### 3.7.1 Gradient Descent

The gradient descent optimization algorithm moves the model's parameters along their gradient to achieve the local minimum, where the gradient of each parameter $w$ is the partial derivative of the loss function with respect to $w$.
The gradient descent algorithm works as follows:

1. Start with randomly selected weights and biases
2. In time step $t$, update each parameter $w_i$ according to this equation:

$$w_i(t) = w_i(t-1) - \eta \Delta w_i(t-1) \tag{3.26}$$

   where $\Delta w_i$ is the gradient of $w_i$, and $\eta$ the chosen learning rate
3. Repeat the first two steps, until a local minimum is found.

There are some drawbacks to adopting the gradient descent approach. The algorithm depends on the initially chosen parameters and converges to a local minimum, which is not necessarily the global optimum. Moreover, in large datasets, it becomes expensive to calculate the gradients, since the algorithm depends on the loss function applied on the totality of the training set to update the weights and biases. It is more appropriate to choose a better-suited optimizer for more complex tasks with abundant data.

### 3.7.2 Stochastic Gradient Descent (SGD)

The SGD algorithm follows the same principle as the gradient descent. However, it exploits a subset of the training data in each iteration. In each step, the stochastic gradient descent algorithms selects, randomly, a batch of the data and updates the weights based on their estimated gradients.
In high or extremely high-dimensional optimization problems, this trick minimizes the computing cost, allowing for quicker iterations in exchange for a decreased convergence rate [19].

### 3.7.3 Adaptive Gradient Descent (AdaGrad)

The difference between the previously seen techniques and AdaGrad [20] is that the latter uses different learning rates for each parameter. The rate also changes in each iteration $t$. The method adjusts parameters related with frequently occurring features with larger learning rates than parameters associated with uncommon features.
Each weight $w$ is updated according to the following formula:

$$w_i(t) = w_i(t-1) - \frac{\eta}{\sqrt{G(t)_{i,i} + \epsilon}} \Delta w_i(t-1) \tag{3.27}$$

$G(t)$ is a matrix where $G(t)_{i,i}$ stores the sum of the squares of the gradients with respect to $w_i$ up to time step $t$, while $\epsilon$ is a small added value to prevent a division by zero.
One of the shortcomings of Adagrad is the buildup of squared gradients in the matrix $G(t)$. Because each added squared gradient is positive, the denominator $\sqrt{G(t)_{i,i} + \epsilon}$ keeps growing. As a result, the learning rate decreases and finally becomes infinitely small, which hinders the learning process.

### 3.7.4 RMSProp

The RMSProp algorithm is an unpublished technique proposed by Professor Geoffrey Hinton [1], which tries to solve AdaGrad's decaying learning rate problem. Instead of accumulating all past squared gradients in G(t), RMSProp redefines the sum of gradients as a decaying average of all past squared gradients. At a time step $t$, the calculated average $\mathbb{E}(w_i, t)$ for weight $w_i$ is calculated as follows:

$$\mathbb{E}(w_i, t) = \gamma \mathbb{E}(w_i, t-1) + (1-\gamma)(\Delta w_i(t))^2 \tag{3.28}$$

Where $\gamma$ is a pre-defined constant (usually equals to 0.9).
The update formula for the parameter $w_i$ becomes:

$$w_i(t) = w_i(t-1) - \frac{\eta}{\sqrt{\mathbb{E}(w_i, t-1) + \epsilon}} \Delta w_i(t-1) \tag{3.29}$$

### 3.7.5 Adaptive Moment Estimation (Adam)

Adam [21] stores an exponentially decaying average of past gradients $m(w_i, t)$ in addition to the exponentially decaying average of past squared gradients like RMSProp. The optimizer actually uses a bias-corrected version of $m(w_i, t)$ and $\mathbb{E}(w_i, t)$ which are calculated using the

[1] http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

following formulas:

$$\hat{\mathbb{E}}(w_i,t) = \frac{\gamma_1 \mathbb{E}(w_i,t-1) + (1-\gamma_1)(\Delta w_i(t))^2}{1-\gamma_1^t} \tag{3.30}$$

$$\hat{m}(w_i,t) = \frac{\gamma_2 m(w_i,t-1) + (1-\gamma_2)\Delta w_i(t)}{1-\gamma_2^t} \tag{3.31}$$

The Adam update rule becomes:

$$w_i(t) = w_i(t-1) - \frac{\eta}{\sqrt{\hat{\mathbb{E}}(w_i,t)+\epsilon}}\hat{m}(w_i,t) \tag{3.32}$$

## 3.8 Anomaly Detection

Anomaly or outlier detection is a technique used to detect odd observations within a dataset. It identifies data points that differ considerably from the rest of the data and do not adhere to a well-defined idea of normal behavior [22].
Typically, anomalies indicate a problem such as bank fraud, a structural defect, a medical problem or an error in textual data [23]. In most cases, unsupervised techniques are applied due to the generally unavailable labelled data.

Outlier detection can also be used in machine learning as part of the data pre-processing phase, to clean the data-set from odd or faulty observations, which normally would improve the model and facilitates its ability to learn the correct underlying features and solve the task.

The anomaly detection techniques used in the context of this thesis are defined in the following subsections.

### 3.8.1 Isolation Forests

Isolation forests, or iForest, is a "fundamentally different model-based method" that explicitly isolates outliers instead of modelling the normal observations [24].
According to the authors, isolation forests is a faster anomaly detector which employs binary trees to isolate anomalies. It directly targets them without profiling all the normal instances.

The idea behind isolation forests is to partition the available data, by building a binary tree. The sample data $X$ is recursively divided by randomly selecting an attribute, or feature, $q$ and a split value $p$, until either: "(i) the tree reaches a height limit, (ii) $|X| = 1$ or (iii) all data in $X$ have the same values" [24]. The anomalies correspond to the data-points which have a considerably shorter path length than the others. A path length is calculated by the number of edges traversed by a point $x$ in the binary tree from the root node to a leaf node. The Figure 3.4 shows that isolating an outlier is faster and easier than a random point. The point $x_i$ depicted in the figures requires twelve random partitions to be isolated, whereas the outlier $x_o$ only requires four.

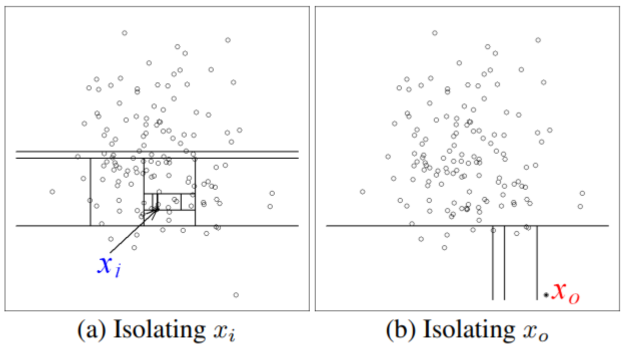

**Figure 3.4:** The Difference between Isolation a Non-anomalous point $x_i$ and an outlier $x_o$ [24].

### 3.8.2 Elliptic Envelope

Elliptic Envelope is a method, which detects anomalies in Gaussian distributed data. This method assumes that the data is from a known distribution. If an ellipse is constructed around the data's Gaussian distribution, anything outside of it is deemed an outlier. Figure 3.5 depicts

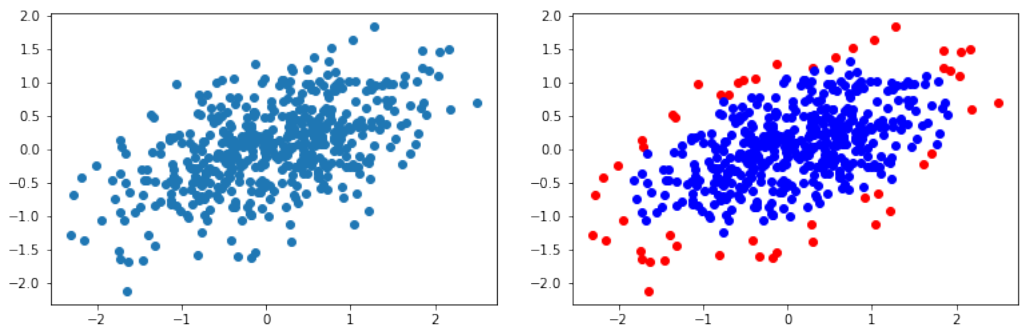

**Figure 3.5:** Elliptic Envelope applied on Gaussian distributed data.

an example of the elliptical envelope method in action. The left sub-figure shows 500 datapoints randomly sampled from a Gaussian distribution, and the right sub-figure depicts the result of the elliptic envelope algorithm applied on the data. The blue points correspond to the normal points, and the red points, which reside outside of the learned ellipse, correspond to the found outliers.

## 3.9 Model Architectures

For solving the underlying problem, three different architectures alongside a baseline model were designed.

### 3.9.1 Baseline model

The baseline is defined as a starting point used for comparisons. It is important to have such a model to facilitate the evaluation of the applied approaches. The results produced by applying the baseline architecture enable the verification of the usability, relevance and the benefits drawn from adding components. If, for example, adding a component to this architecture or adopting a different, more complex, approach yields similar or worse results, it does not make sense to continue on that direction.

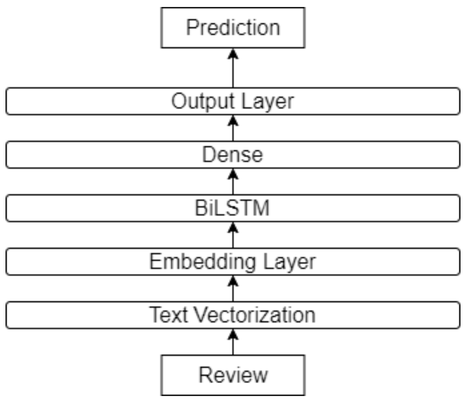

**Figure 3.6:** The Baseline Architecture.

### 3.9.2 Hierarchical Attention Networks with fastText Embedding (FT-HAN)

This model implements the hierarchical attention networks architecture [1], with the difference that this implementation uses fastText [2] embeddings learned from the available data.

### 3.9.3 FT-HAN-BiLSTM

The FT-HAN-BiLSTM architecture extends the previous one by replacing the sentence-level GRU layer by a BiLSTM network.

### 3.9.4 FT-HAN-BiLSTM with Auto-encoder (FT-HAN-BiLSTM-AE)

An auto-encoder is implemented to reduce the dimensionality of the observations and to extract relevant features, which should help improve the performance of the classifier.
As depicted in Figure 3.9, the encoder and decoder parts are not applied on the reviews, but rather on each sentence of each review. The reviews are split by sentences like in the previous architectures and fastText embeddings are produced from each word. The encoder is a BiLSTM network, which extracts features from the embedded sentences, while the decoder tries to reconstruct these embeddings using the encoded data.
The loss function used by the auto-encoder calculates the dissimilarity between the embeddings and the reconstructed data for each sentence.

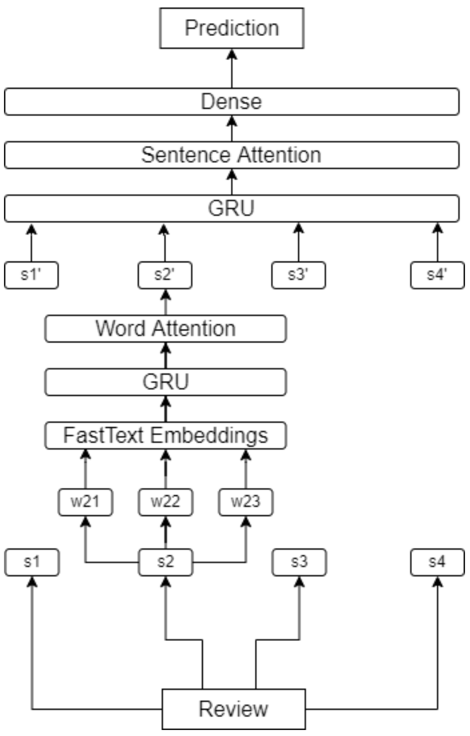

**Figure 3.7:** The FT-HAN Architecture.

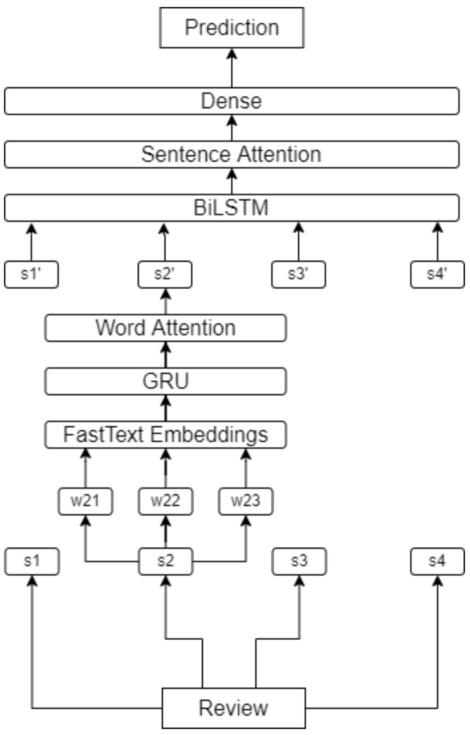

**Figure 3.8:** The FT-HAN-BiLSTM Architecture.

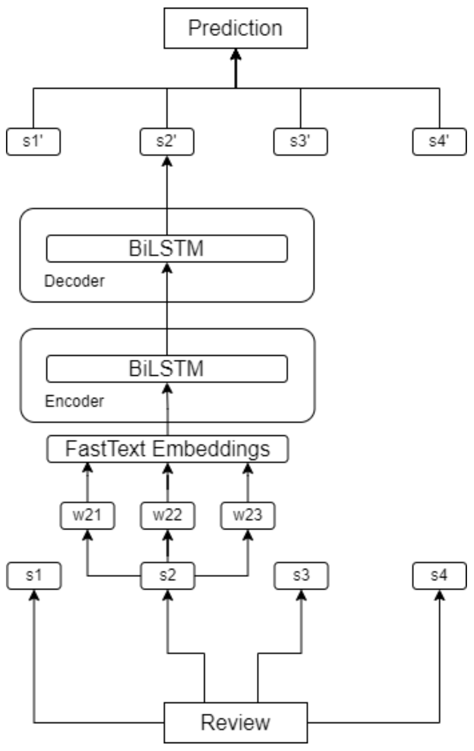

**Figure 3.9:** The Auto-encoder Architecture.

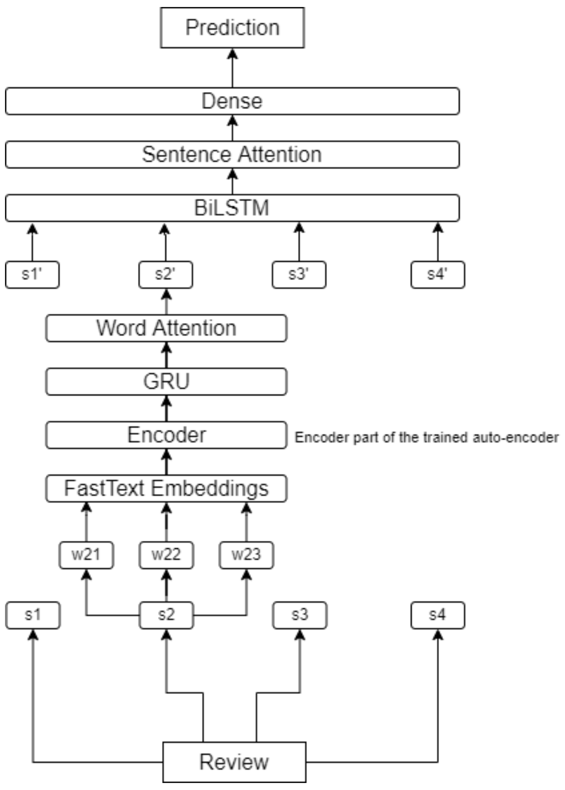

**Figure 3.10:** The FT-HAN-BiLSTM-AE Architecture.

After training the auto-encoder, the encoder part is saved and integrated in this final architecture depicted in Figure 3.10. The produced embeddings are passed through the encoder which reduces its dimensionality and enhances its relevant features.
The remaining components are similar to the previous two architectures.

## 3.10 Transfer Learning

Transfer learning is a machine learning technique applied to re-use and exploit knowledge between tasks. Usually, a model developed for a (source) task is re-employed as the initial state of a model on a second (target) task.
In some situations, only a small amount of labelled data is available and generating new labeled data-points require expert domain knowledge and is usually very expensive. Applying knowledge learned from tasks for which a lot of labelled data is available becomes crucial and optimally using existing datasets becomes advantageous. Transfer learning works best, if the features learned from the source task are general.

There are three possible benefits to look for when using transfer learning [25]:

- Higher start: The initial skill (before refining the model) on the source model is higher than it otherwise would be.
- Higher slope: The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
- Higher asymptote: The converged skill of the trained model is better than it otherwise would be.

In order to solve the multi-domain sentiment analysis problem, the previously described architectures in section 3.9 are implemented and general classifiers are learned using data from all datasets. The idea is to generate domain-specific classifiers for each domain using transfer learning from the general models.
After comparing the various architectures, transfer learning is applied using the best performing general classifier. The pre-trained weights are re-used, and some of them are fine-tuned during another training phase using data from one target domain, in order to produce a domain-specific classifier.

Three approaches of fine-tuning weights are adopted in this phase, which are listed below:

- **Sentence-level**: In this approach, only the weights of the sentence level layers of the previously described architectures are fine-tuned. The word level layers are "frozen" and do not get trained during the domain-specific training phase.
- **Word-level**: Analogously to the first approach, only the weights of the word level layers are fine-tuned.
- **All layers**: During the domain-specific training, all pre-trained weights from the general classifier continue to train using data from the target domain. There are no frozen layers in this approach.

## 3.11 Active Learning

Active learning is a semi-supervised learning model where the user intervenes during the process [26]. Specifically, unlike the classical framework where the data is known and imposed, in active learning it is the learning algorithm that requests information for specific data. This type of learning is mainly useful in situations where a lot of unlabelled data is available but manual labeling is expensive.
It can be used to optimize the data points chosen for labelling and training a model based on them. Active learning is the name used for the process of prioritizing the data which needs to be labelled in order to have the highest impact to training a supervised model.

In Natural Language Processing, there are a lot of tasks which require an abundance of labelled data, and usually, there is a very high cost to labelling it. Therefore, it is useful to have an active learning pipeline in place, which can substantially decrease the needed amount of labelled data. However, using active learning does not always yield good results, when there are a lot of outliers in the available data. Moreover, if the underlying data have a nearly uniform distribution, this approach do not offer any improvements in comparison to randomly sampling the data to be labelled.

In an active learning process, the algorithm chooses some data from a data-pool to be labelled. It uses a metric for this purpose, which is called a priority score. There are several approaches to assign a priority score to each data point. Some of them, which are used in this thesis, are listed below:

- Least Confidence: It takes the highest probability for each data point's prediction and sorts them from smaller to larger. The chosen observations $x^*$ are sampled according to this equation:

$$x^* = \underset{x}{argmax}\ (1 - p(\hat{y}|x)) \tag{3.33}$$

- Entropy Sampling: In the case of high entropy, it would mean that the model distributes equally the probability for all classes as it is not certain at all which class that data point belongs to. It is therefore straightforward to prioritize data points with higher entropy to the ones with lower entropy. The chosen observations $x^*$ are sampled according to this equation:

$$x^* = \underset{x}{argmax}\ - \sum_i p(y_i|x) log(p(y_i|x)) \tag{3.34}$$

# 4 Implementation

## 4.1 Datasets

During the conducted experiments, the following datasets were used to evaluate the designed architectures:

- Multi-Domain Sentiment Dataset [27]:
  This set consists of a collection of product reviews from Amazon.com and are grouped into 25 categories. Only 14 of the 25 domains are considered, because they have a comparable number of reviews and they have been used in other related works [3] [6]. This will facilitate comparing the solutions presented in this thesis with past ones.
- IMDb Dataset [28]:
  Multiple movie reviews with binary classes from the Internet Movie Database (IMDb). The IMDb domain is by far the largest one among other datasets. It contains $25,000$ positive and $25,000$ negative reviews. In order to balance the different domains, only a portion of this dataset is going to be exploited.
- MR Dataset [29]:
  This dataset also consists of movie reviews with two classes. It originates from the rotten tomatoes website [1].

## 4.2 Data Pre-processing

### 4.2.1 Raw Data Processing

The goal of the raw data processing is to have one folder for each of the 16 domains with the following structure:

```
Domain
├── train
│   ├── pos
│   └── neg
└── test
    ├── pos
    └── neg
```

The Listing 4.2 shows the method used to split an array of reviews into train and test folders. The method also takes the actual label of the aforementioned reviews (*neg* or *pos*), and creates

[1] https://www.rottentomatoes.com/

the necessary directories to store the data and to achieve the previously mentioned folder structure. It also uses the helper method listed in Listing 4.1 to write each review in a separate file in the corresponding folder.

**Listing 4.1:** Save Reviews in Separate Files.

```python
import os

def write_to_file(reviews, path):
    for i in range(len(reviews)):
        with open(os.path.join(path, str(i) + '.txt'), 'w', encoding="latin
            -1") as text_file:
             text_file.write(reviews[i])
```

**Listing 4.2:** Split Reviews into Train and Test Sets.

```python
import os
from sklearn.model_selection import train_test_split

def generate_train_test_files(reviews, path, label):
    train_path = os.path.join(path, 'train/' + label)
    test_path = os.path.join(path, 'test/' + label)
    os.makedirs(train_path)
    os.makedirs(test_path)
    reviews_train, reviews_test = train_test_split(reviews, test_size=0.2,
        random_state=42)
    write_to_file(reviews_train, train_path)
    write_to_file(reviews_test, test_path)
```

#### 4.2.1.1 Multi-domain Sentiment Dataset

The multidomain sentiment dataset as provided by Blitzer et al. [2] contains one folder for each of the 25 domains. Each directory has the following structure:

```
Domain
 ├─ all.review
 ├─ negative.review
 ├─ positive.review
 ├─ processed.review
 ├─ processed.review.balanced
 └─ unlabeled.review
```

The *negative.review*, *positive.review* and *unlabeled.review* files contain a succession of reviews with their metadata in an xml-format. For each review, the actual text to be extracted and used is found under the <*review_text*> tag.

The Listing 4.4 lists the method *process_multi_domain_sentiment_dataset* used to process each of the domain folders in the multidomain sentiment dataset. Given a parameter *root_path*, which corresponds to the path leading to a domain folder, the method iterates over the *negative.review*,

[2] https://www.cs.jhu.edu/ mdredze/datasets/sentiment/

*positive.review* and *unlabeled.review* files, extracts the actual reviews using the method listed in Listing 4.3, and splits them into train and test folders using the method listed in Listing 4.2.

**Listing 4.3:** Extract Reviews From Files.

```python
import os
def get_reviews_from_file(filename):
    content = []
    read_next = False
    try:
        with open(filename, "r+", encoding="latin-1") as file:
            review = ''
            for line in file:
                if line.rstrip() == '</review_text>':
                    read_next = False
                    content.append(review)
                    review = ''
                if read_next:
                    review += line
                if line.rstrip() == '<review_text>':
                    read_next = True
    except IOError:
        print("[ERR] File does not exist")

    return content
```

**Listing 4.4:** Preprocess Data from the Multidomain Sentiment Dataset.

```python
import os
def process_multi_domain_sentiment_dataset(root_path):
    for root,  dirs, files in os.walk(root_path):
        for f in files:
            if f == 'positive.review':
                print(f'processing: {root}/{f}')
                reviews = get_reviews_from_file(os.path.join(root, f))
                generate_train_test_files(reviews, root, 'pos')
            if f == 'negative.review':
                print(f'processing: {root}/{f}')
                reviews = get_reviews_from_file(os.path.join(root, f))
                generate_train_test_files(reviews, root, 'neg')
            if f == 'unlabeled.review':
                print(f'processing: {root}/{f}')
                reviews = get_reviews_from_file(os.path.join(root, f))
                generate_train_test_files(reviews, root, 'unlabeled')
```

#### 4.2.1.2 IMDb Dataset

The IMDb dataset as provided by Maas et al. [3] already has the desired folder structure. There are no pre-processing steps needed to be performed on the raw data in this case.
However, since this dataset is very large ($50,000$ labeled and $50,000$ unlabeled reviews), the

[3] https://ai.stanford.edu/ amaas/data/sentiment/

only step to be made before starting the training phase is to randomly select a small portion of the set to be used.

#### 4.2.1.3 MR Dataset

The reviews of the MR Dataset, as provided by Pang and Lee [4], are already stored in individual files in two folders, mainly *pos* and *neg*. The pre-processing procedure to be performed is described in Listings 4.5, 4.6 and 4.7. It consists in splitting the content of the *pos* and *neg* folders into train and test sets, and moving the sets to newly generated directories, in order to attain the desired folder structure.

**Listing 4.5:** Move Files to Another Folder.

```
import os

def move_files(files, source_folder, destination_folder):
    for file in files:
        os.replace(os.path.join(source_folder, file), os.path.join(
            destination_folder, file))
```

**Listing 4.6:** Split Reviews into Train and Test sets.

```
import os

def split_folder(path, label):
    subfolder = os.path.join(path, label)
    train_path = os.path.join(path, 'train/' + label)
    test_path = os.path.join(path, 'test/' + label)
    os.makedirs(train_path)
    os.makedirs(test_path)
    files = next(os.walk(subfolder))[2]
    train_files, test_files = train_test_split(files, test_size=0.2,
        random_state=42)
    move_files(train_files, subfolder, train_path)
    move_files(test_files, subfolder, test_path)
```

**Listing 4.7:** Preprocess MR Dataset.

```
def process_mr_dataset(root_path):
    split_folder(root_path, 'neg')
    split_folder(root_path, 'pos')
```

### 4.2.2 Load Data

Listing 4.8 shows the method used to load the training, validation and test-sets. It returns a Tensorflow [30] Batch Dataset [5].

**Listing 4.8:** Load Training

[4] 5https://www.cs.cornell.edu/people/pabo/movie-review-data/
[5] https://www.tensorflow.org/api_docs/python/tf/data/Dataset

```python
import os
from enum import Enum, unique
from tensorflow.keras import utils

@unique
class Domain(Enum):
    Books = 'Books'
    Electronics = 'Electronics'
    DVD = 'DVD'
    Kitchen = 'Kitchen'
    Apparel = 'Apparel'
    Camera = 'Camera'
    Health = 'Health'
    Music = 'Music'
    Toys = 'Toys'
    Video = 'Video'
    Baby = 'Baby'
    Magazines = 'Magazines'
    Software = 'Software'
    Sports = 'Sports'
    IMDb = 'IMDb'
    MR = 'MR'


# global constant for all domains
ALL_DOMAINS = tuple(Domain)


def load_domain(dataset_path, n_val=0.125, batch_size=32, seed=42):
    train_dir = os.path.join(dataset_path, 'train')

    raw_train_ds = utils.text_dataset_from_directory(
        train_dir,
        batch_size=batch_size,
        validation_split=n_val,
        subset='training',
        seed=seed)

    # Create a validation set.
    raw_val_ds = utils.text_dataset_from_directory(
        train_dir,
        batch_size=batch_size,
        validation_split=n_val,
        subset='validation',
        seed=seed)

    test_dir = os.path.join(dataset_path, 'test')

    # Create a test set.
    raw_test_ds = utils.text_dataset_from_directory(
        test_dir,
        batch_size=batch_size)

    return raw_train_ds, raw_val_ds, raw_test_ds
```

### 4.2.3 Learning Word Embeddings

In order to facilitate the learning process of word embeddings, a helper method was implemented to generate a file containing the totality of the available data. All the reviews were concatenated and saved in a text file, which will be used by the fastText library to train the word embedding model. The Listing 4.9 shows the method used to generate the file.

**Listing 4.9:** Corpus File Generation.

```python
import os
def generate_corpus_file(path, output_path):
    with open(output_path, 'w', encoding="latin-1") as outfile:
        for root, dirs, files in os.walk(path):
            for file in files:
                with open(os.path.join(root, file), "r", encoding="latin-1"
                    ) as infile:
                    outfile.write(infile.read())
```

The advantage with using fastText is its ability to quickly learn vector representations for words using an unsupervised algorithm. This bypasses the shortage problem of labeled data, since it is possible to use the totality of available data from each domain to learn domain-specific embeddings or from all domains to learn more general representations.

Using the *train_unsupervised* method of fastText's python module and the file generated using the algorithm mentioned above in Listing 4.9, it is possible to learn word embeddings using the underlying data and save the learned model using the *save_model* method, as described in Listing 4.10.

**Listing 4.10:** Learn Word Embeddings.

```python
import fasttext
def produce_embeddings(path, output_path):
    embedding = fasttext.train_unsupervised(path)
    embedding.save_model(output_path)
```

## 4.3 Training Modules

For the training phase of the baseline and the three proposed architectures, the following specifications are used:

- **Loss function:** The Binary Crossentropy function as described in Section 3.6 was chosen as the objective to be minimized during training. Since the models solve a binary classification problem, this loss function is well-suited for this task.
- **Optimizer:** The Adam optimizer [21] was applied with an initial learning rate $\eta$ of 0.0001.
- **Epochs:** The algorithm ran for a maximum of 100 Epochs during the general classifier learning phase.

- **Callback:** an EarlyStopping callback to interrupt the training, in case there is no improvement in the loss function during five consecutive epochs.

### 4.3.1 Baseline Model

The baseline model is a basic implementation of a neural network, which can solve the sentiment analysis problem. It serves as a control model to compare the other architectures. Listing 4.11 shows the actual code of the baseline model's architecture and training function and it consists of:

- A TextVectorization layer [6] which maps text features to integer sequences. The vocabulary size was set to 1000. The least common words that do not fit under the 1000 most frequent are mapped to the '[UNK]' token.
- An Embedding layer [7], which maps the input sequences to word embeddings having a dimension equal to 64.
- BiLSTM network consisting of two consecutive BiLSTM layers of 64 and 32 units, respectively.
- A Dense layer having an output dimension of 64.
- A Dropout layer with a probability of 0.2.
- A Dense output layer with one unit, since the model solves a binary classification problem.

**Listing 4.11:** Training the baseline Model.

```python
def train(train_ds, val_ds, epochs=50):
    """
        Trains the baseline model
        :param train_ds: training data_set
        :param val_ds: validation data_sets
        :param epochs: number of epochs for training
        :return: trained model and history
        """
    # Encoder
    vocab_size = 1000
    encoder = tf.keras.layers.TextVectorization(
        max_tokens=vocab_size,
        output_sequence_length=50)
    encoder.adapt(train_ds.map(lambda text, label: text))

    # Basic LSTM Model with encoder and embedding layers
    model = tf.keras.Sequential([
        encoder,
        tf.keras.layers.Embedding(
            input_dim=vocab_size,
            output_dim=64,
            # Use masking to handle the variable sequence lengths
```

[6] https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization
[7] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

```
            mask_zero=True),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,
           return_sequences=True)),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1)
    ])

    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)

    model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True)
       ,
                  optimizer=tf.keras.optimizers.Adam(1e-4),
                  metrics=['accuracy'])

    val_steps = val_ds.cardinality().numpy()
    history = model.fit(train_ds, epochs=epochs,
                        validation_data=val_ds,
                        validation_steps=val_steps,
                        callbacks=[callback])

    return model, history
```

### 4.3.2 Remaining Models (FT-HAN, FT-HAN-BiLSTM and FT-HAN-BiLSTM-AE)

#### 4.3.2.1 Data Transformation

Before training the remaining models, it is necessary to perform some pre-processing steps on the data. Since, the architectures are divided into word- and sentence-level layers, the data should also be re-arranged by sentence. Instead of having an input batch with the shape $(batch\_size, )$, where each element consists of a review in string format, the input shape is changed to $(batch\_size, max\_sentences, max\_len)$. $max\_sentences$ and $max\_len$ are parameters given by the user and they correspond to the maximum permitted number of sentences per review and number of words per sentence, respectively. For example, if the user sets $max\_sentences$ to 20, only the first 20 sentences of each review is retained. Analogously, only the first $max\_len$ words of each sentence is kept after the pre-processing phase.

After reshaping the input data, each word is mapped to an embedding vector using the trained fastText model. After the pre-processing step, the final shape of the input data is $(batch\_size, max\_sentences, max\_len, fasttext\_dimension)$, where $fasttext\_dimension$ is the output embedding dimension of the fastText model.

The class shown in Listing 4.12 contains the method $preprocess\_ds\_fasttext$, which can perform the previously described step, provided that the class is initialized and a trained fasttext model is set using the $set\_fasttext\_model$ method.

**Listing 4.12:** Preprocessor Class.

```
import re
import fasttext
import numpy as np
import tensorflow as tf
from nltk import tokenize
from tensorflow.keras.preprocessing.text import text_to_word_sequence


class Preprocessor(object):
    def __init__(self, max_sentences=50, max_len=50, fasttext_path=None):
        self.max_sentences = max_sentences
        self.max_len = max_len
        if fasttext_path is not None:
            self.fasttext_model = fasttext.load_model(fasttext_path)

    # Tokenization/string cleaning for dataset
    def clean_str(self, string):
        string = re.sub(r"\\", "", string)
        string = re.sub(r"\'", "", string)
        string = re.sub(r"\"", "", string)
        return string.strip().lower()

    def set_fasttext_model(self, fasttext_path):
        self.fasttext_model = fasttext.load_model(fasttext_path)

    def preprocess_ds_fasttext(self, r):
        data = np.zeros((r.shape[0], self.max_sentences, self.max_len, self
            .fasttext_model.get_dimension()),
                        dtype='float32')
        for i, review in enumerate(r):
            text = self.clean_str((review.numpy().decode("latin-1")))
            sentences = tokenize.sent_tokenize(text)[:self.max_sentences]
            for j, sent in enumerate(sentences):
                if j < self.max_sentences:
                    word_tokens = text_to_word_sequence(sent)[:self.max_len
                        ]
                    for k, word in enumerate(word_tokens):
                        data[i, j, k] = self.fasttext_model[word]
        return tf.constant(data,
                           shape=(r.shape[0], self.max_sentences, self.
                               max_len, self.fasttext_model.get_dimension())
                               ,
                           dtype=tf.float32)
```

#### 4.3.2.2 Attention

The attention class implemented as shown in Listing 4.13 inherits from the *tensorflow.keras.layers.Layer*[8] class and is used in the designed models of the FT-HAN, FT-HAN-BiLSTM and FT-HAN-BiLSTM-AE architectures. The class implements the attention functions as described in the

[8] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer

Subsection 3.4.2 and the paper published by Yang et al. [1], and serves as the attention layer for the word- and sentence-level attention mechanisms.

**Listing 4.13:** Attention Class.

```python
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Layer


# class defining the custom attention layer
class AttentionNetwork(Layer):
    def __init__(self, att_dim, **kwargs):
        self.att_dim = att_dim
        super(AttentionNetwork, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3
        self.W = self.add_weight(shape=(input_shape[-1], self.att_dim),
                                 name='att_W',
                                 initializer='random_normal',
                                 trainable=True)
        self.b = self.add_weight(shape=(self.att_dim,),
                                 name='att_b',
                                 initializer='random_normal',
                                 trainable=True)
        self.u = self.add_weight(shape=(self.att_dim, 1),
                                 name='att_u',
                                 initializer='random_normal',
                                 trainable=True)

        super(AttentionNetwork, self).build(input_shape)

    def call(self, x):
        uit = K.tanh(K.bias_add(K.dot(x, self.W), self.b))

        ait = K.exp(K.squeeze(K.dot(uit, self.u), -1))

        ait /= K.cast(K.sum(ait, axis=1, keepdims=True) + K.epsilon(), K.
            floatx())
        weighted_input = x * K.expand_dims(ait)
        output = K.sum(weighted_input, axis=1)

        return output

    def compute_output_shape(self, input_shape):
        return input_shape[0], input_shape[-1]

    def get_config(self):
        config = super().get_config()
        config.update({
            "att_dim": self.att_dim
        })
        return config
```

#### 4.3.2.3 Model Training

One Training module was implemented for the three remaining architectures. Since the three models have the same core architecture and the difference between them lies in adding one component or two, it is sufficient to have the same training function for them. This function accepts two input parameters to indicate if a component needs to be appended or added.
For example, the *use_bilstm* parameter is a boolean value indicating if a BiLSTM network is added before applying the sentence level attention, and the *encoder* parameter is a model corresponding to the encoder part of a pre-trained auto-encoder. This Parameter is defaulted to *None*.

The implemented models consist of:

- A TimeDistributed layer [9] which envelopes the *word_encoder* layers including the word-level attention layer. It applies these layers to each sentence of the input. The *word_encoder* includes:
  - The encoder part of the pre-trained auto-encoder (Only for the FT-HAN-BiLSTM-AE architecture).
  - A bidirectional GRU layer with 100 units.
  - An attention layer having an attention dimension equal to 100.
- BiLSTM network consisting of two consecutive BiLSTM layers of 100 and 50 units, respectively, in case of the FT-HAN-BiLSTM architecture. Otherwise, a bidirectional GRU layer with 100 units.
- An attention layer having an attention dimension equal to 100.
- A Dropout layer with a probability of 0.1.
- A Dense output layer with one unit, since the model solves a binary classification problem.

**Listing 4.14:** Training the Model.

```
def train(train_ds, val_ds, epochs=100, encoder=None, use_bilstm=False):
    """
        :param train_ds: training data_set
        :param val_ds: validation data_set
        :param epochs: number of epochs for training
        :param encoder: the pre-trained encoder from auto-encoder
        :param use_bilstm: a flag to use a biLSTM network or not on the
           sentence-level network
        :return: trained model and history
        """

    # Add encoder layer from pre-trained auto-encoder
    if encoder is not None:
```

[9] https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed

```
        word_encoder = Sequential([
            encoder.layers[0],
            Bidirectional(GRU(100, return_sequences=True)),
            AttentionNetwork(100),
        ])
    else:
        word_encoder = Sequential([
            Bidirectional(GRU(100, return_sequences=True)),
            AttentionNetwork(100),
        ])

    # Add a BiLSTM network
    if use_bilstm:
        model = Sequential([
            TimeDistributed(word_encoder),
            Bidirectional(LSTM(100, return_sequences=True)),
            Bidirectional(LSTM(50, return_sequences=True)),
            AttentionNetwork(100),
            Dropout(0.1),
            Dense(1)
        ])
    else:
        model = Sequential([
            TimeDistributed(word_encoder),
            Bidirectional(GRU(100, return_sequences=True)),
            AttentionNetwork(100),
            Dropout(0.1),
            Dense(1)
        ])

    if encoder is not None:
        model.layers[0].layer.layers[0].trainable = False

    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)

    model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                  optimizer=tf.keras.optimizers.Adam(1e-4),
                  metrics=['accuracy'])

    val_steps = val_ds.cardinality().numpy()
    history = model.fit(train_ds, epochs=epochs,
                        validation_data=val_ds,
                        validation_steps=val_steps,
                        callbacks=[callback])

    return model, history
```

The Listing 4.14 shows the implementation of the three architectures and the training function. For the FT-HAN-BiLSTM-AE model, the encoder part of the pre-trained auto-encoder should not continue training alongside the other layers and it is made not trainable, before fitting the training set.

## 4.4 Testing Module

All previously described training modules produce a Tensorflow Sequential Model [30]. Such a model can be evaluated using the *evaluate*() method and a test dataset [10].
For this reason, only one testing module was implemented for all four architectures.

**Listing 4.15:** Testing the model.

```python
def test(model, test_ds):
    """
        Evaluates the model
        :param model: trained model
        :param test_ds: test data_sets
        :return: test loss & accuracy
        """
    if isinstance(model, tf.keras.Sequential):
        test_loss, test_acc = model.evaluate(test_ds)

        print('Test Loss:', test_loss)
        print('Test Accuracy:', test_acc)
        return test_loss, test_acc
    else:
        raise TypeError("The model ist not of type tf.keras.Sequential")
```

## 4.5 Transfer Learning

For the Transfer Learning phase, two main methods are applied on the domain-specific data to train the corresponding models. The first one is random sampling, where a portion of the training set is randomly sampled and used during training. This approach is evaluated using bootstrapping, which is a re-sampling method, where the experiment is re-conducted each time with a new random sample [31]. The second approach is Active Learning, where the data sampling is actively chosen by the algorithm using a pre-defined metric like the ones defined in Section 3.11.

Listing 4.16 shows an example method for Transfer Learning using random sampling. The sampled training data corresponds to a portion of the whole domain-specific training set. Different ratios are selected each time, which are 5%, 10%, 15%, 20% and 100%. For each proportion, the experiment, including the training and evaluation phases, is re-conducted using another random sample. All the test accuracies and losses are then returned by the method.

**Listing 4.16:** Transfer Learning on one domain using random sampling.

```python
import math
import numpy as np
import tensorflow as tf
from attention import AttentionNetwork


```

[10] https://www.tensorflow.org/api_docs/python/tf/keras/Model

```
def get_random_sampling_results(domain_preprocessed_train_ds,
    domain_preprocessed_val_ds, domain_preprocessed_test_ds, model_path)
    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)
    results = np.zeros((5, 10, 2), dtype='float32')
    i = 0
    for ratio in [0.05, 0.1, 0.15, 0.2, 1]:

        for j in range(10):
            model = tf.keras.models.load_model(model_path,custom_objects={"
                AttentionNetwork": AttentionNetwork})

            model.compile(loss=tf.keras.losses.BinaryCrossentropy(
                from_logits=True),
                          optimizer=tf.keras.optimizers.Adam(1e-4),
                          metrics=['accuracy'])

            val_steps = math.ceil(domain_preprocessed_val_ds.cardinality().
                numpy() * ratio)
            train_steps = math.ceil(domain_preprocessed_train_ds.
                cardinality().numpy() * ratio)
            used_domain_train_ds = domain_preprocessed_train_ds.take(
                train_steps).cache()
            used_domain_val_ds = domain_preprocessed_val_ds.take(val_steps)
                .cache()

            model.fit(used_domain_train_ds, epochs=10,
                      validation_data=used_domain_val_ds,
                      validation_steps=val_steps,
                      callbacks=[callback])

            results[i, j, 0], results[ i, j, 1] = model.evaluate(
                domain_preprocessed_test_ds)

        i += 1
    return results
```

Listing 4.17 lists an example method for Active Learning using Entropy Sampling. In the context of this thesis, four approaches for Active Learning are considered. Namely, Entropy Sampling and Uncertainty Sampling, both with and without anomaly detection using Isolation Forests [24]. In the Active Learning experiment, different ratios of the domain-specific training set are selected each time. The proportions used are identical to the ones from the random sampling approach.
The modAL framework is used for the Active Learning process [32].

**Listing 4.17:** Transfer Learning on One Domain using Active Learning.

```
import math
import numpy as np
import tensorflow as tf
from modAL.models import ActiveLearner
from modAL.uncertainty import entropy_sampling
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from attention import AttentionNetwork


def get_entropy_sampling_results(domain_preprocessed_train_ds,
    domain_preprocessed_val_ds, domain_preprocessed_test_ds, model_path)

    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)
    results = np.zeros((5, 2), dtype='float32')

    x_train = []
    y_train = []

    for r, l in domain_preprocessed_train_ds.unbatch():
        x_train.append(r.numpy())
        y_train.append(l.numpy())

    x_train = np.array(x_train)
    y_train = np.array(y_train)

    i = 0
    for ratio in [0.05, 0.1, 0.15, 0.2, 1]:
        model = tf.keras.models.load_model(model_path,custom_objects={"
            AttentionNetwork": AttentionNetwork})
        classifier = KerasClassifier(lambda: model)
        classifier.model = model
        # initialize ActiveLearner
        learner = ActiveLearner(
            estimator=classifier,
            query_strategy=entropy_sampling,
            verbose=1
        )

        query_idx, query_instance = learner.query(x_train, n_instances=math
            .ceil(len(x_train) * ratio), verbose=0)
        domain_preprocessed_train_ds = tf.data.Dataset.from_tensor_slices((
            x_train[query_idx], y_train[query_idx]))\
            .shuffle(32).batch(batch_size)
        val_steps = domain_preprocessed_val_ds.cardinality().numpy()

        model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=
            True),
                      optimizer=tf.keras.optimizers.Adam(1e-4),
                      metrics=['accuracy'])

        model.fit(domain_preprocessed_train_ds,
                  epochs=10,
                  validation_data=domain_preprocessed_val_ds,
                  validation_steps=val_steps,
                  callbacks=[callback])

        results[i, 0], results[i, 1] = model.evaluate(
            domain_preprocessed_test_ds)

        i += 1

    return results
```

Both listings 4.17 and 4.16 show an example of Transfer Learning, where all general pre-trained weights are transferred to the domain-specific models and all their corresponding layers continue the training phase. This only one of the methods used in the context of this thesis. There are actually three approaches adopted, which are explained in Section 3.10.

## 4.6 Anomaly Detection

In order to further improve the performance of the model, specifically during the Active Learning phase, anomaly detection is performed on the training set.
Two outlier detection techniques, isolation forest and elliptic envelope 3.8, are applied and compared. The results are presented in the evaluation chapter 5.

Listings 4.18 and 4.19 list the methods used to fit the anomaly detection classifiers on the training set. Here the *sklearn.ensemble.IsolationForest* and *sklearn.covariance.EllipticEnvelope* classes of scikit-learn [33] are employed on the totality of the training data from all domains.

**Listing 4.18:** Fitting Isolation Forest Classifier.

```python
from sklearn.ensemble import IsolationForest

def fit_isolation_forest(x_train):
    """
        :param x_train: training set
        :return: isolation forest classifier
        """
    clf = IsolationForest(max_samples=140)
    clf.fit(x_train)
    return clf
```

**Listing 4.19:** Fitting Elleptic Envelope Classifier.

```python
from sklearn.covariance import EllipticEnvelope

def fit_elliptic_envelope(x_train):
    """
        :param x_train: training set
        :return: elliptic envelope classifier
        """
    cov = EllipticEnvelope(random_state=0)
    cov.fit(x_train)
    return cov
```

## 4.7 Hyperparameter Optimization

Hyperparameter optimization is a crucial step in any data science or machine learning task. It is applied on implemented models, in order to search for the best hyperparameters. This should further enhance the performance of the underlying model and yield better results in the task.

There are a lot of techniques, which can be used for this purpose, like Random Search, Bayesian Optimization and the Hyperband algorithm. The latter was applied using the KerasTuner framework [34] on the models after their initial evaluation.
The Hyperband method was chosen, because it learns from previously tested parameter combinations and is more optimal in terms of time and memory allocations. On the other hand, Random Search samples combinations randomly from the search space and do not learn from previous iterations.

The search space that the algorithm chooses from is described as follows:

- The number of units of the word-level GRU layer $units_1 \in \{25, 50, 75, 100\}$
- The attention dimension of the word-level attention layer $attention\_dim_1 \in \{50, 100\}$
- The number of units of the first sentence-level BiLSTM layer $units_2 \in \{25, 50, 75, 100\}$
- The number of units of the secont sentence-level BiLSTM layer $units_3 \in \{25, 50, 75, 100\}$
- The attention dimension of the sentence-level attention layer $attention\_dim_2 \in \{50, 100\}$
- The rate of the dropout layer $rate \in \{0, 0.05, 0.1, 0.15, 0.2\}$
- The starting learning rate of the Adam optimizer: Log-sampling of $lr$ between 0.0001 and 0.01

**Listing 4.20:** An Example of a build_model Method supporting Hyperparameter Optimization.

```python
import tensorflow as tf
from tensorflow.keras.layers import GRU, LSTM, Bidirectional,
    TimeDistributed, Dropout, Dense
from attention import AttentionNetwork
from tensorflow.keras.models import Sequential


def build_model(hp):

    word_encoder = Sequential([
        Bidirectional(GRU(units=hp.Int("units_1", min_value=25, max_value
            =100, step=25), return_sequences=True)),
        AttentionNetwork(hp.Choice("attention_dim_1", [50, 100])),
    ])

    model = Sequential([
        TimeDistributed(word_encoder),
        Bidirectional(LSTM(units=hp.Int("units_2", min_value=25, max_value
            =100, step=25), return_sequences=True)),
        Bidirectional(LSTM(units=hp.Int("units_3", min_value=25, max_value
            =100, step=25), return_sequences=True)),
        AttentionNetwork(hp.Choice("attention_dim_2", [50, 100])),
        Dropout(hp.Float("rate", min_value=0.0, max_value=0.2, step=0.05)),
        Dense(1)
    ])
    learning_rate = hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling
        ="log")
    model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True)
        ,
                  optimizer=tf.keras.optimizers.Adam(learning_rate),
                  metrics=['accuracy'])
    return model
```

**Listing 4.21:** Hyperparameter Optimzation using Keras Tuner.

```
import tensorflow as tf
import keras_tuner as kt
import training

def search_hp(preprocessed_train_ds, preprocessed_val_ds):
    tuner = kt.Hyperband(
        training.build_model,
        objective='val_loss',
        max_epochs=100,
        factor=3,
        hyperband_iterations=5,
        seed=42,
        hyperparameters=None,
        tune_new_entries=True,
        allow_new_entries=True,
        overwrite=True
    )

    callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=5)
    val_steps = preprocessed_val_ds.cardinality().numpy()

    tuner.search(preprocessed_train_ds, epochs=100, validation_data=
        preprocessed_val_ds, validation_steps=val_steps, callbacks=[callback
        ])

    best_model = tuner.get_best_models()[0]
    return best_model
```

Listing 4.21 lists the method used to search for the optimal hyperparameters, using the Hyperband algorithm of the Keras Tuner framework. The method takes as input the pre-processed training and validation sets. The underlying model is drawn from the *build$_m$odel* method of the *training* module. The method is listed in Listing 4.20.

## 4.8 Tools and Libraries

For the version control of this project, the GitLab instance provided by the Technical University of Berlin (TU Berlin) was used. The implemented code during this thesis can be found under https://git.tu-berlin.de/nabildouss-1997/multi-domain-sentiment-analysis.git.

The project was implemented in Python 3.9.7 [35] with Conda [36] as the package management system. The totality of the libraries and frameworks used are listed below:

**Listing 4.22:** environment.yml File.

```
name: multidomain-sentiment-analysis
channels:
  - conda-forge
  - defaults
dependencies:
  - asttokens=2.0.5=pyhd3eb1b0_0
  - backcall=0.2.0=pyhd3eb1b0_0
  - beautifulsoup4=4.10.0=pyha770c72_0
  - blas=1.0=mkl
  - brotli-bin=1.0.9=h8ffe710_6
  - bs4=4.10.0=hd8ed1ab_0
  - ca-certificates=2021.10.8=h5b45459_0
  - certifi=2021.10.8=py39hcbf5309_2
  - click=8.0.4=py39hcbf5309_0
  - colorama=0.4.4=pyhd3eb1b0_0
  - cudatoolkit=11.3.1=h59b6b97_2
  - cudnn=8.2.1=cuda11.3_0
  - cycler=0.11.0=pyhd8ed1ab_0
  - debugpy=1.5.1=py39hd77b12b_0
  - decorator=5.1.1=pyhd3eb1b0_0
  - entrypoints=0.3=py39haa95532_0
  - executing=0.8.3=pyhd3eb1b0_0
  - fonttools=4.31.2=py39hb82d6ee_0
  - freetype=2.10.4=h546665d_1
  - html5lib=1.1=pyh9f0ad1d_0
  - icc_rt=2019.0.0=h0cc432a_1
  - icu=69.1=h0e60522_0
  - intel-openmp=2021.4.0=haa95532_3556
  - ipykernel=6.9.1=py39haa95532_0
  - ipython=8.1.1=py39haa95532_0
  - ipython_genutils=0.2.0=pyhd3eb1b0_1
  - jbig=2.1=h8d14728_2003
  - jedi=0.18.1=py39haa95532_1
  - joblib=1.1.0=pyhd3eb1b0_0
  - jpeg=9e=h8ffe710_0
  - jupyter_client=7.1.2=pyhd3eb1b0_0
  - jupyter_core=4.9.2=py39haa95532_0
  - kiwisolver=1.4.0=py39h2e07f2f_0
  - lcms2=2.12=h2a16943_0
  - lerc=3.0=h0e60522_0
  - libbrotlicommon=1.0.9=h8ffe710_6
- libbrotlidec=1.0.9=h8ffe710_6
- libbrotlienc=1.0.9=h8ffe710_6
- libdeflate=1.10=h8ffe710_0
- libpng=1.6.37=h1d00b33_2
- libtiff=4.3.0=hc4061b1_3
- libwebp=1.2.2=h57928b3_0
- libwebp-base=1.2.2=h8ffe710_1
- libxcb=1.13=hcd874cb_1004
- libzlib=1.2.11=h8ffe710_1014
- lz4-c=1.9.3=h8ffe710_1
- m2w64-gcc-libgfortran=5.3.0=6
- m2w64-gcc-libs=5.3.0=7
- m2w64-gcc-libs-core=5.3.0=7
- m2w64-gmp=6.1.0=2
- m2w64-libwinpthread-git=5.0.0.4634.697f757=2
- matplotlib=3.5.1=py39hcbf5309_0
- matplotlib-base=3.5.1=py39h581301d_0
- matplotlib-inline=0.1.2=pyhd3eb1b0_2
- mkl=2021.4.0=haa95532_640
- mkl-service=2.4.0=py39h2bbff1b_0
- mkl_fft=1.3.1=py39h277e83a_0
- mkl_random=1.2.2=py39hf11a4ad_0
- modal=0.4.1=pyhd8ed1ab_0
- msys2-conda-epoch=20160418=1
- munkres=1.1.4=pyh9f0ad1d_0
- nest-asyncio=1.5.1=pyhd3eb1b0_0
- nltk=3.6.7=pyhd8ed1ab_0
- numpy=1.21.5=py39ha4e8547_0
- numpy-base=1.21.5=py39hc2deb75_0
- openjpeg=2.4.0=hb211442_1
- openssl=1.1.1n=h8ffe710_0
- packaging=21.3=pyhd8ed1ab_0
- pandas=1.4.1=py39h2e25243_0
- parso=0.8.3=pyhd3eb1b0_0
- pickleshare=0.7.5=pyhd3eb1b0_1003
- pillow=9.0.1=py39ha53f419_2
- pip=21.2.4=py39haa95532_0
- prompt-toolkit=3.0.20=pyhd3eb1b0_0
- pthread-stubs=0.4=hcd874cb_1001
- pure_eval=0.2.2=pyhd3eb1b0_0
- pygments=2.11.2=pyhd3eb1b0_0
- pyparsing=3.0.7=pyhd8ed1ab_0
- pyqt=5.12.3=py39hcbf5309_8
- pyqt-impl=5.12.3=py39h415ef7b_8
- pyqt5-sip=4.19.18=py39h415ef7b_8
- pyqtchart=5.12=py39h415ef7b_8
- pyqtwebengine=5.12.1=py39h415ef7b_8
- python=3.9.7=h6244533_1
- python-dateutil=2.8.2=pyhd3eb1b0_0
- python_abi=3.9=2_cp39
- pytz=2022.1=pyhd8ed1ab_0
- pywin32=302=py39h827c3e9_1
- pyzmq=22.3.0=py39hd77b12b_2
- qt=5.12.9=h556501e_6
- regex=2022.3.15=py39hb82d6ee_0
- scikit-learn=1.0.2=py39hf11a4ad_1
- scipy=1.7.3=py39h0a974cb_0
- setuptools=58.0.4=py39haa95532_0
- six=1.16.0=pyhd3eb1b0_1
- soupsieve=2.3.1=pyhd8ed1ab_0
- sqlite=3.38.0=h2bbff1b_0
- stack_data=0.2.0=pyhd3eb1b0_0
- threadpoolctl=2.2.0=pyh0d69192_0
- tk=8.6.12=h8ffe710_0
- tornado=6.1=py39h2bbff1b_0
- tqdm=4.63.1=pyhd8ed1ab_0
- traitlets=5.1.1=pyhd3eb1b0_0
- tzdata=2021e=hda174b7_0
- unicodedata2=14.0.0=py39hb82d6ee_0
- vc=14.2=h21ff451_1
- vs2015_runtime=14.27.29016=h5e58377_2
- wcwidth=0.2.5=pyhd3eb1b0_0
- webencodings=0.5.1=py_1
- wheel=0.37.1=pyhd3eb1b0_0
- wincertstore=0.2=py39haa95532_2
- xorg-libxau=1.0.9=hcd874cb_0
- xorg-libxdmcp=1.1.3=hcd874cb_0
- xz=5.2.5=h62dcd97_1
- zlib=1.2.11=h8ffe710_1014
- zstd=1.5.2=h6255e5f_0
- pip:
  - absl-py==1.0.0
  - astunparse==1.6.3
  - brotli==1.0.9
  - cachetools==5.0.0
  - charset-normalizer==2.0.12
  - flatbuffers==2.0
  - gast==0.4.0
  - google-auth==2.6.0
  - google-auth-oauthlib==0.4.6
  - google-pasta==0.2.0
  - grpcio==1.44.0
  - h5py==3.6.0
  - idna==3.3
  - importlib-metadata==4.11.3
  - keras==2.7.0
  - keras-preprocessing==1.1.2
  - keras-self-attention==0.51.0
  - libclang==13.0.0
  - markdown==3.3.6
  - oauthlib==3.2.0
  - opt-einsum==3.3.0
  - protobuf==3.19.4
  - pyasn1==0.4.8
```

```
    - pyasn1-modules==0.2.8
    - pybind11==2.9.1
    - requests==2.27.1
    - requests-oauthlib==1.3.1
    - rsa==4.8
    - tensorboard==2.7.0
    - tensorboard-data-server==0.6.1
    - tensorboard-plugin-wit==1.8.1
    - tensorflow-estimator==2.7.0
    - tensorflow-gpu==2.7.0
    - tensorflow-io-gcs-filesystem==0.24.0
    - termcolor==1.1.0
    - tf-estimator-nightly==2.8.0.dev2021122109
    - typing-extensions==4.1.1
    - urllib3==1.26.8
    - werkzeug==2.0.3
    - wrapt==1.14.0
    - zipp==3.7.0
```

# 5 Evaluation

## 5.1 Datasets

| Domain | Train | Validation | Test | Unlabeled |
|---|---|---|---|---|
| Books | 1400 | 200 | 400 | 2000 |
| Electronics | 1400 | 200 | 400 | 2000 |
| DVD | 1400 | 200 | 400 | 2000 |
| Kitchen | 1400 | 200 | 400 | 2000 |
| Apparel | 1400 | 200 | 400 | 2000 |
| Camera | 1398 | 200 | 400 | 2000 |
| Health | 1400 | 200 | 400 | 2000 |
| Music | 1400 | 200 | 400 | 2000 |
| Toys | 1400 | 200 | 400 | 2000 |
| Video | 1400 | 200 | 400 | 2000 |
| Baby | 1312 | 188 | 360 | 2000 |
| Magazines | 1358 | 194 | 388 | 2000 |
| Software | 1281 | 183 | 366 | 475 |
| Sports | 1400 | 200 | 400 | 2000 |
| IMDb | 1400 | 200 | 400 | 2000 |
| MR | 1400 | 200 | 400 | 0 |

**Table 5.1:** Statistics of the 16 Domains.

## 5.2 Experiments

The conducted experiments, including the models' training and testing and the production of results and visualizations, were performed on a machine with the following specifications:

- CPU: AMD Ryzen 5 4600H
- GPU: NVIDIA GeForce RTX 2600
- RAM: 16GB

After the implementation of the architectures defined in the section 3.9, the models were evaluated by following the steps listed below:

1. Split the pre-processed dataset into training, validation and test-sets.
2. Produce a corpus file, which contains all reviews.
3. Using the corpus file and the fastText library, word embeddings are learned in an unsupervised manner.

4. Train a general classifier using a concatenated training and a validation-set from all domains.
5. Evaluate the general classifier on each domain using the corresponding test-set.
6. Calculate an average test accuracy and loss.
7. Choose the most performing model and start the transfer learning phase to produce domain-specific models.
8. For the training phase of the domain-specific classifiers, various experiments are conducted using different ratios of the training-set and different transfer learning approaches.
   a) Ratios: Use respectively 5%, 10%, 15%, 20% and 100% of the available training data.
   b) Approaches: Random Sampling of data and Active Learning using Entropy Sampling and Uncertainty Sampling.

## 5.3 General Classifiers

Table 5.2 shows the losses and accuracies of the test-set as produced by the baseline model and the three proposed architectures. The models were trained using training data from all domains, and their general performance is presented in the last row under "Avg". The Table also describes the performance of each model on each domain's test-set.
As the Table 5.2 shows, the FT-HAN-BiLSTM was the best performing model with an average

| | Baseline | | FT-HAN | | FT-HAN-BiLSTM | | FT-HAN-BiLSTM-AE | |
|---|---|---|---|---|---|---|---|---|
| Domain | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 1.108 | 78.85% | 0.678 | 84.25% | 0.430 | **87.00%** | **0.368** | 85.50% |
| Electronics | 0.893 | 80.20% | 0.450 | **89.00%** | **0.297** | 88.75% | 0.373 | 81.50% |
| DVD | 1.232 | 76.25% | 0.560 | **85.25%** | 0.413 | 84.25% | **0.345** | 84.75% |
| Kitchen | 1.057 | 78.55% | 0.476 | **89.50%** | **0.277** | 89.00% | 0.357 | 85.75% |
| Apparel | 0.923 | 75.25% | 0.481 | **89.50%** | 0.344 | 88.75% | **0.321** | 85.50% |
| Camera | 0.656 | 86.70% | 0.419 | 90.75% | **0.260** | **91.00%** | 0.274 | 87.75% |
| Health | 1.212 | 78.35% | 0.416 | 89.75% | **0.286** | **91.00%** | 0.287 | 86.00% |
| Music | 1.009 | 77.10% | 0.674 | 83.75% | 0.463 | **84.75%** | **0.428** | 82.50% |
| Toys | 0.855 | 80.60% | 0.510 | 88.50% | 0.325 | **88.50%** | **0.318** | 86.25% |
| Video | 1.093 | 77.10% | 0.447 | 89.50% | 0.331 | **90.25%** | **0.330** | 84.75% |
| Baby | 1.100 | 77.94% | 0.706 | 83.33% | **0.346** | **88.89%** | 0.390 | 81.11% |
| Magazines | 0.693 | 86.44% | 0.452 | 89.95% | **0.238** | **92.53%** | 0.371 | 85.05% |
| Software | 0.891 | 82.90% | 0.405 | **89.07%** | **0.310** | 87.70% | 0.311 | 84.43% |
| Sports | 0.941 | 83.55% | 0.401 | **88.75%** | 0.309 | 87.50% | **0.281** | 86.25% |
| IMDb | 1.161 | 76.27% | 0.657 | 83.00% | 0.410 | 86.25% | **0.300** | **88.00%** |
| MR | 1.037 | 75.70% | 0.745 | 85.00% | 0.497 | **87.50%** | **0.367** | 85.25% |
| Avg | 0.992 | 79.46% | 0.530 | 87.44% | 0.346 | **88.34%** | **0.339** | 85.05% |

**Table 5.2:** Performance of the General Classifiers on Each Domain.

test accuracy of 88.34%. The accuracy of this model was better than its counterparts in 9 of the 16 domains, where the FT-HAN had a larger test accuracy in the Electronics, DVD, Kitchen, Apparel, Software and Sports domains.
The FT-HAN-BiLSTM-AE model had a greater accuracy only in the IMDb domain, but it managed to reach a lower test loss than the other models in 9 of the 16 domains with an average

test loss of 0.339.

Table 5.3 shows the effect of using the FastText embeddings on the performance of the FT-HAN and FT-HAN-BiLSTM models. These embeddings were produced by training a fastText model [2] on the whole available dataset, including unlabelled data, in an unsupervised manner. The table contains a comparison between using the FastText embeddings and Tensorflow's [30] embedding layer [1].

As shown in Table 5.3, both the FT-HAN and FT-HAN-BiLSTM models achieve a higher test

| | HAN | | HAN-BiLSTM | | FT-HAN | | FT-HAN-BiLSTM | |
|---|---|---|---|---|---|---|---|---|
| Domain | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 0.854 | 75.50% | 0.667 | 80.75% | 0.678 | 84.25% | 0.430 | 87.00% |
| Electronics | 0.574 | 84.50% | 0.510 | 85.25% | 0.450 | 89.00% | 0.297 | 88.75% |
| DVD | 0.583 | 84.00% | 0.579 | 82.75% | 0.560 | 85.25% | 0.413 | 84.25% |
| Kitchen | 0.547 | 84.75% | 0.490 | 85.50% | 0.476 | 89.50% | 0.277 | 89.00% |
| Apparel | 0.500 | 86.00% | 0.448 | 86.25% | 0.481 | 89.50% | 0.344 | 88.75% |
| Camera | 0.492 | 86.50% | 0.358 | 89.50% | 0.419 | 90.75% | 0.260 | 91.00% |
| Health | 0.587 | 82.75% | 0.506 | 84.50% | 0.416 | 89.75% | 0.286 | 91.00% |
| Music | 0.849 | 77.00% | 0.679 | 80.00% | 0.674 | 83.75% | 0.463 | 84.75% |
| Toys | 0.717 | 82.00% | 0.607 | 82.00% | 0.510 | 88.50% | 0.325 | 88.50% |
| Video | 0.680 | 81.00% | 0.530 | 84.50% | 0.447 | 89.50% | 0.331 | 90.25% |
| Baby | 0.603 | 83.61% | 0.501 | 85.28% | 0.706 | 83.33% | 0.346 | 88.89% |
| Magazines | 0.506 | 86.08% | 0.428 | 87.89% | 0.452 | 89.95% | 0.238 | 92.53% |
| Software | 0.518 | 85.52% | 0.508 | 84.70% | 0.405 | 89.07% | 0.310 | 87.70% |
| Sports | 0.532 | 84.75% | 0.525 | 84.75% | 0.401 | 88.75% | 0.309 | 87.50% |
| IMDb | 0.667 | 80.25% | 0.644 | 82.50% | 0.657 | 83.00% | 0.410 | 86.25% |
| MR | 0.645 | 81.25% | 0.723 | 78.00% | 0.745 | 85.00% | 0.497 | 87.50% |
| Avg | 0.617 | 82.82% | 0.545 | 83.99% | 0.530 | 87.44% | 0.346 | 88.34% |

**Table 5.3:** The Effect of Using FastText Embeddings on the Performance of the General Classifiers.

accuracy and a lower test loss in comparison to their HAN and HAN-BiLSTM counterparts, that do not include the FastText embeddings. The FT-HAN-BiLSTM is superior than the HAN-BiLSTM model in all domains. However, the HAN model achieves a higher accuracy and a lower test loss than the FT-HAN model in the Baby domain, and has a lower test loss in the MR domain.

## 5.4 Transfer Learning

The tables 5.4, 5.5 and 5.6 show the test accuracies and losses of the transfer learning approach used on the FT-HAN-BiLSTM model. It represents a comparison between the general classifier's performance, and the results produced by the random sampling and active learning approaches. In these tables, the active learning results were generated by entropy sampling. The random sampling results are averages over 10 consecutive trials using bootstrapping, as explained in Section 4.5.

[1] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Sentence-Level | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 5% | 0.430 | **87.00%** | 0.485 | 85.20% | **0.420** | 84.50% |
| | 10% | | | 0.473 | 84.83% | 0.484 | 85.50% |
| | 15% | | | 0.462 | 84.88% | 0.487 | 86.00% |
| | 20% | | | 0.469 | 85.50% | 0.464 | 86.25% |
| | 100% | | | 0.478 | 85.83% | 0.497 | 86.25% |
| Electronics | 5% | **0.297** | 88.75% | 0.310 | **89.05%** | 0.322 | 89.00% |
| | 10% | | | 0.321 | **88.87%** | 0.373 | 88.00% |
| | 15% | | | 0.314 | 88.72% | 0.389 | 88.50% |
| | 20% | | | 0.306 | **88.98%** | 0.383 | 88.50% |
| | 100% | | | 0.336 | **89.62%** | 0.322 | 88.75% |
| DVD | 5% | **0.413** | 84.25% | 0.499 | 84.45% | 0.446 | **85.00%** |
| | 10% | | | 0.488 | **84.80%** | 0.543 | 84.50% |
| | 15% | | | 0.513 | **84.55%** | 0.550 | 83.75% |
| | 20% | | | 0.543 | **85.03%** | 0.518 | 84.25% |
| | 100% | | | 0.504 | 84.40% | 0.510 | **84.50%** |
| Kitchen | 5% | **0.277** | 89.00% | 0.294 | **89.73%** | 0.327 | 89.50% |
| | 10% | | | 0.307 | **89.73%** | 0.313 | 89.25% |
| | 15% | | | 0.284 | **89.77%** | 0.290 | 89.75% |
| | 20% | | | 0.291 | **89.92%** | 0.292 | 89.25% |
| | 100% | | | 0.322 | 89.05% | 0.315 | **89.50%** |
| Apparel | 5% | 0.344 | 88.75% | 0.404 | 88.27% | **0.337** | **88.75%** |
| | 10% | | | 0.404 | 88.15% | 0.379 | 88.50% |
| | 15% | | | 0.392 | 88.18% | 0.360 | 88.50% |
| | 20% | | | 0.376 | 87.98% | 0.375 | 88.50% |
| | 100% | | | 0.405 | 88.05% | 0.416 | 87.75% |
| Camera | 5% | 0.260 | 91.00% | 0.335 | 90.98% | **0.259** | 90.00% |
| | 10% | | | 0.348 | **91.10%** | 0.272 | 90.50% |
| | 15% | | | 0.315 | **91.20%** | 0.280 | 90.25% |
| | 20% | | | 0.331 | 90.95% | 0.286 | 90.00% |
| | 100% | | | 0.319 | 91.00% | 0.322 | **91.50%** |
| Health | 5% | 0.286 | **91.00%** | 0.365 | 90.13% | **0.279** | 90.75% |
| | 10% | | | 0.378 | 90.30% | 0.289 | 90.75% |
| | 15% | | | 0.372 | 90.15% | 0.292 | 90.50% |
| | 20% | | | 0.375 | 90.17% | 0.293 | 90.00% |
| | 100% | | | 0.326 | 90.37% | 0.323 | 90.50% |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Sentence-Level | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Music | 5% | **0.463** | 84.75% | 0.523 | 84.08% | 0.531 | 84.25% |
| | 10% | | | 0.544 | 84.17% | 0.618 | **84.75%** |
| | 15% | | | 0.541 | 84.40% | 0.581 | 84.00% |
| | 20% | | | 0.502 | 84.38% | 0.588 | 84.50% |
| | 100% | | | 0.516 | 84.23% | 0.529 | 83.75% |
| Toys | 5% | 0.325 | 88.50% | 0.327 | 88.27% | **0.289** | **90.00%** |
| | 10% | | | 0.316 | 88.77% | **0.310** | **89.25%** |
| | 15% | | | 0.343 | 89.05% | **0.318** | **89.75%** |
| | 20% | | | 0.322 | 88.87% | **0.307** | **89.50%** |
| | 100% | | | **0.309** | **90.38%** | 0.322 | 90.25% |
| Video | 5% | **0.331** | **90.25%** | 0.375 | 89.90% | 0.370 | 89.75% |
| | 10% | | | 0.378 | 89.55% | 0.411 | 89.00% |
| | 15% | | | 0.405 | 89.25% | 0.356 | 89.75% |
| | 20% | | | 0.391 | 89.35% | 0.365 | 89.50% |
| | 100% | | | 0.363 | 88.95% | 0.370 | 88.50% |
| Baby | 5% | **0.346** | 88.89% | 0.369 | 87.92% | 0.391 | 88.33% |
| | 10% | | | 0.375 | 88.19% | 0.411 | **89.17%** |
| | 15% | | | 0.408 | 88.17% | 0.418 | **89.72%** |
| | 20% | | | 0.370 | 88.39% | 0.384 | **89.17%** |
| | 100% | | | 0.358 | **88.92%** | 0.365 | 88.06% |
| Magazines | 5% | 0.238 | 92.53% | 0.269 | 92.19% | **0.227** | **92.78%** |
| | 10% | | | 0.265 | 92.11% | 0.252 | 92.27% |
| | 15% | | | 0.259 | 92.11% | 0.246 | **93.30%** |
| | 20% | | | 0.251 | 92.32% | 0.258 | **93.30%** |
| | 100% | | | 0.238 | **92.91%** | 0.252 | 92.78% |
| Software | 5% | **0.310** | 87.70% | 0.400 | 88.25% | 0.356 | **88.52%** |
| | 10% | | | 0.419 | 88.52% | 0.335 | **88.80%** |
| | 15% | | | 0.430 | 88.61% | 0.364 | **89.34%** |
| | 20% | | | 0.404 | **88.88%** | 0.333 | 88.52% |
| | 100% | | | 0.354 | 88.88% | 0.375 | **89.34%** |
| Sports | 5% | **0.309** | 87.50% | 0.330 | **88.13%** | 0.338 | 87.50% |
| | 10% | | | 0.326 | **88.13%** | 0.378 | 87.75% |
| | 15% | | | 0.341 | 88.17% | 0.339 | **88.50%** |
| | 20% | | | 0.352 | 88.18% | 0.361 | **88.50%** |
| | 100% | | | 0.367 | 87.68% | 0.380 | **88.00%** |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Sentence-Level | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| IMDb | 5% | **0.410** | 86.25% | 0.480 | 86.43% | 0.461 | **86.75%** |
| | 10% | | | 0.496 | 86.63% | 0.492 | **86.75%** |
| | 15% | | | 0.519 | 87.15% | 0.471 | **87.25%** |
| | 20% | | | 0.527 | **86.85%** | 0.496 | 86.75% |
| | 100% | | | 0.542 | 86.13% | 0.549 | 85.75% |
| MR | 5% | **0.497** | 87.50% | 0.535 | 87.53% | 0.557 | **87.75%** |
| | 10% | | | 0.551 | 87.22% | 0.580 | **88.00%** |
| | 15% | | | 0.575 | 87.18% | 0.555 | 86.75% |
| | 20% | | | 0.573 | **87.65%** | 0.564 | 86.50% |
| | 100% | | | 0.616 | 86.83% | 0.637 | 87.25% |
| Avg. | 5% | **0.346** | 88.34% | 0.394 | 88.16% | 0.369 | 88.32% |
| | 10% | | | 0.399 | 88.19% | 0.402 | 88.30% |
| | 15% | | | 0.405 | 88.22% | 0.393 | **88.48%** |
| | 20% | | | 0.399 | **88.34%** | 0.392 | 88.31% |
| | 100% | | | 0.397 | 88.33% | 0.405 | 88.28% |

**Table 5.4:** Performance of the Domain Specific Classifiers Using Sentence-Level Transfer Learning.

Table 5.4 shows the performance of the domain-specific models, trained by random sampling and by active learning in comparison to the general classifiers. In this approach, only the sentence-level layers continued training after the knowledge transfer from the general models. The word-level layers were "frozen", meaning they retained their pre-trained weights during the training phase.

In average, the active learning approach using entropy sampling and only 15% of the domain-specific training-set performs better than the general classifier and the random sampling approach (88.48% test accuracy). However, the latter achieves a higher accuracy of 88.34%, when using 20% of the training-set. The general classifier attains a greater test accuracy than the domain-specific approaches in only 3 of the 16 domains, however it has a lower test loss in 10 domains, and achieves a lower average loss of 0.346.

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Word-Level | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| | 5% | | | 0.487 | 84.65% | 0.434 | 84.75% |
| | 10% | | | 0.471 | 85.13% | 0.499 | 85.00% |
| Books | 15% | **0.430** | **87.00%** | 0.473 | 84.90% | 0.502 | 84.75% |
| | 20% | | | 0.485 | 84.90% | 0.449 | 85.25% |
| | 100% | | | 0.450 | 85.78% | 0.469 | 86.50% |
| | 5% | | | 0.340 | 88.30% | 0.299 | **90.00%** |
| | 10% | | | 0.364 | 88.33% | 0.356 | **89.00%** |
| Electronics | 15% | **0.297** | 88.75% | 0.355 | 88.55% | 0.353 | **90.00%** |
| | 20% | | | 0.355 | 89.12% | 0.359 | **90.25%** |
| | 100% | | | 0.363 | **88.95%** | 0.362 | 88.50% |
| | 5% | | | 0.479 | **84.58%** | 0.496 | 83.75% |
| | 10% | | | 0.459 | 84.65% | 0.472 | **85.25%** |
| DVD | 15% | **0.413** | 84.25% | 0.482 | 84.23% | 0.459 | **86.00%** |
| | 20% | | | 0.482 | 84.48% | 0.479 | **86.25%** |
| | 100% | | | 0.481 | 84.50% | 0.476 | **85.00%** |
| | 5% | | | 0.319 | 89.43% | 0.321 | **90.25%** |
| | 10% | | | 0.310 | **89.55%** | 0.334 | 89.00% |
| Kitchen | 15% | **0.277** | 89.00% | 0.293 | **90.30%** | 0.330 | 89.00% |
| | 20% | | | 0.300 | **90.27%** | 0.378 | 88.25% |
| | 100% | | | 0.301 | 90.83% | 0.309 | **91.00%** |
| | 5% | | | 0.390 | 88.28% | **0.314** | **89.00%** |
| | 10% | | | 0.401 | 88.20% | **0.343** | **89.25%** |
| Apparel | 15% | 0.344 | 88.75% | 0.404 | 88.18% | **0.344** | 88.50% |
| | 20% | | | 0.399 | 88.00% | 0.345 | 88.25% |
| | 100% | | | 0.387 | 88.30% | 0.393 | 88.00% |
| | 5% | | | 0.332 | 90.88% | 0.286 | 90.50% |
| | 10% | | | 0.319 | **91.18%** | 0.299 | 90.75% |
| Camera | 15% | **0.260** | 91.00% | 0.298 | 91.38% | 0.277 | **91.75%** |
| | 20% | | | 0.300 | **91.58%** | 0.289 | 91.50% |
| | 100% | | | 0.291 | **92.00%** | 0.292 | 91.25% |
| | 5% | | | 0.375 | 89.80% | 0.304 | 90.00% |
| | 10% | | | 0.373 | 89.83% | 0.310 | 89.50% |
| Health | 15% | **0.286** | **91.00%** | 0.366 | 90.20% | 0.327 | 90.25% |
| | 20% | | | 0.378 | 89.85% | 0.326 | 88.75% |
| | 100% | | | 0.318 | 90.05% | 0.299 | 90.25% |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Word-Level | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Music | 5% | **0.463** | 84.75% | 0.543 | 83.30% | 0.492 | **84.75%** |
| | 10% | | | 0.566 | 83.10% | 0.533 | **85.50%** |
| | 15% | | | 0.566 | 83.88% | 0.533 | **85.50%** |
| | 20% | | | 0.570 | 83.53% | 0.514 | 84.50% |
| | 100% | | | 0.534 | 84.55% | 0.534 | **85.00%** |
| Toys | 5% | 0.325 | 88.50% | 0.336 | 87.05% | 0.343 | 86.00% |
| | 10% | | | 0.347 | 87.53% | 0.340 | **88.50%** |
| | 15% | | | 0.333 | 88.20% | **0.324** | **89.25%** |
| | 20% | | | 0.346 | 87.65% | **0.284** | **90.00%** |
| | 100% | | | **0.311** | 89.00% | 0.333 | **89.50%** |
| Video | 5% | **0.331** | **90.25%** | 0.370 | 89.73% | 0.382 | 89.50% |
| | 10% | | | 0.372 | 89.42% | 0.408 | 88.75% |
| | 15% | | | 0.376 | 89.45% | 0.365 | 89.25% |
| | 20% | | | 0.363 | 89.30% | 0.381 | 89.25% |
| | 100% | | | 0.359 | 89.10% | 0.348 | 89.25% |
| Baby | 5% | **0.346** | 88.89% | 0.458 | 86.17% | 0.427 | **88.89%** |
| | 10% | | | 0.443 | 87.58% | 0.484 | 87.50% |
| | 15% | | | 0.454 | 87.64% | 0.472 | 87.22% |
| | 20% | | | 0.444 | 87.22% | 0.552 | 86.11% |
| | 100% | | | 0.405 | 87.78% | 0.466 | 87.50% |
| Magazines | 5% | 0.238 | 92.53% | 0.305 | 91.31% | **0.237** | 92.27% |
| | 10% | | | 0.274 | 91.93% | **0.235** | **92.78%** |
| | 15% | | | 0.275 | 91.80% | 0.253 | 92.27% |
| | 20% | | | 0.274 | 91.93% | 0.270 | **92.78%** |
| | 100% | | | 0.273 | 92.40% | 0.282 | 92.27% |
| Software | 5% | **0.310** | 87.70% | 0.372 | 88.17% | 0.315 | **88.25%** |
| | 10% | | | 0.369 | 88.33% | 0.333 | **89.07%** |
| | 15% | | | 0.383 | 88.11% | 0.350 | **88.52%** |
| | 20% | | | 0.387 | 88.28% | 0.348 | **89.07%** |
| | 100% | | | 0.353 | **89.18%** | 0.361 | 88.25% |
| Sports | 5% | **0.309** | 87.50% | 0.347 | **88.03%** | 0.377 | 87.50% |
| | 10% | | | 0.328 | **87.90%** | 0.357 | 87.75% |
| | 15% | | | 0.342 | 87.85% | 0.339 | **88.00%** |
| | 20% | | | 0.326 | **88.23%** | 0.352 | 87.00% |
| | 100% | | | 0.366 | 87.40% | 0.379 | **88.50%** |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Word-Level | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| IMDb | 5% | **0.410** | 86.25% | 0.479 | 85.80% | 0.479 | 85.50% |
| | 10% | | | 0.487 | 85.20% | 0.458 | **86.25%** |
| | 15% | | | 0.481 | 86.00% | 0.462 | 85.75% |
| | 20% | | | 0.484 | 85.95% | 0.443 | 86.00% |
| | 100% | | | 0.493 | 85.10% | 0.501 | 85.00% |
| MR | 5% | 0.497 | 87.50% | 0.522 | 87.00% | 0.561 | 87.00% |
| | 10% | | | 0.524 | 87.38% | **0.473** | **88.25%** |
| | 15% | | | 0.529 | 87.30% | **0.478** | 86.50% |
| | 20% | | | 0.517 | **87.53%** | 0.509 | 87.50% |
| | 100% | | | 0.517 | **87.50%** | 0.515 | 87.00% |
| Avg. | 5% | **0.346** | **88.34%** | 0.403 | 87.65% | 0.379 | 87.99% |
| | 10% | | | 0.400 | 87.83% | 0.390 | 88.26% |
| | 15% | | | 0.401 | 88.00% | 0.385 | 88.28% |
| | 20% | | | 0.401 | 87.99% | 0.392 | 88.17% |
| | 100% | | | 0.388 | 88.28% | 0.395 | 88.30% |

**Table 5.5:** Performance of the Domain Specific Classifiers Using Word-Level Transfer Learning.

Table 5.5 shows the performance of the domain-specific models, trained by random sampling and by active learning in comparison to the general classifiers. In this approach, only the word-level layers continued training after the knowledge transfer from the general models.

In average, the general classifier outperformed the domain-specific models with an accuracy of 88.34% and a test loss of 0.346.
Nevertheless, the active learning approach achieved better test accuracies, in average, across all data ratios in comparison to the random sampling method. It also has a lower test loss, except when using 100% of the data.

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | All Layers | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 5% | **0.430** | **87.00%** | 0.546 | 84.38% | 0.439 | 84.50% |
| | 10% | | | 0.529 | 84.58% | 0.511 | 85.25% |
| | 15% | | | 0.505 | 84.58% | 0.546 | 85.50% |
| | 20% | | | 0.528 | 84.65% | 0.487 | 85.00% |
| | 100% | | | 0.494 | 85.70% | 0.468 | 86.00% |
| Electronics | 5% | **0.297** | 88.75% | 0.392 | 88.33% | 0.352 | **90.75%** |
| | 10% | | | 0.387 | 88.30% | 0.408 | **89.25%** |
| | 15% | | | 0.367 | 88.65% | 0.429 | **89.50%** |
| | 20% | | | 0.388 | 88.58% | 0.442 | **89.75%** |
| | 100% | | | 0.400 | 88.40% | 0.375 | **88.75%** |
| DVD | 5% | **0.413** | 84.25% | 0.564 | **84.33%** | 0.550 | 83.50% |
| | 10% | | | 0.575 | 84.48% | 0.589 | **84.50%** |
| | 15% | | | 0.597 | 84.13% | 0.571 | **85.25%** |
| | 20% | | | 0.648 | 84.45% | 0.594 | **85.00%** |
| | 100% | | | 0.516 | **84.28%** | 0.598 | 84.00% |
| Kitchen | 5% | **0.277** | 89.00% | 0.341 | **89.65%** | 0.349 | 88.75% |
| | 10% | | | 0.343 | **90.20%** | 0.359 | 88.50% |
| | 15% | | | 0.317 | **90.37%** | 0.333 | 90.25% |
| | 20% | | | 0.341 | **90.30%** | 0.366 | 90.00% |
| | 100% | | | 0.336 | 90.63% | 0.330 | **92.00%** |
| Apparel | 5% | 0.344 | 88.75% | 0.449 | 87.95% | **0.329** | **89.50%** |
| | 10% | | | 0.456 | 87.65% | 0.366 | **89.25%** |
| | 15% | | | 0.454 | 87.75% | 0.401 | 87.25% |
| | 20% | | | 0.436 | 87.73% | 0.419 | 88.25% |
| | 100% | | | 0.424 | 88.15% | 0.471 | 87.00% |
| Camera | 5% | **0.260** | 91.00% | 0.363 | 91.05% | 0.265 | **91.75%** |
| | 10% | | | 0.369 | 91.43% | 0.312 | **91.50%** |
| | 15% | | | 0.349 | **91.65%** | 0.339 | 91.25% |
| | 20% | | | 0.354 | 91.78% | 0.323 | **92.25%** |
| | 100% | | | 0.363 | **91.58%** | 0.359 | 91.50% |
| Health | 5% | **0.286** | **91.00%** | 0.432 | 89.20% | 0.305 | 90.75% |
| | 10% | | | 0.428 | 89.70% | 0.357 | 89.50% |
| | 15% | | | 0.437 | 89.65% | 0.330 | 89.75% |
| | 20% | | | 0.442 | 89.78% | 0.326 | 89.25% |
| | 100% | | | 0.340 | 89.80% | 0.327 | 87.75% |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | All Layers | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Music | 5% | **0.463** | 84.75% | 0.564 | 83.30% | 0.552 | 84.25% |
| | 10% | | | 0.604 | 83.30% | 0.658 | **85.00%** |
| | 15% | | | 0.598 | 83.95% | 0.612 | **85.75%** |
| | 20% | | | 0.568 | 83.85% | 0.626 | 84.50% |
| | 100% | | | 0.549 | 84.12% | 0.560 | 83.50% |
| Toys | 5% | 0.325 | 88.50% | 0.384 | 86.75% | 0.357 | 85.00% |
| | 10% | | | 0.379 | 88.10% | 0.374 | **88.75%** |
| | 15% | | | 0.363 | **88.62%** | 0.355 | 88.25% |
| | 20% | | | 0.365 | 88.53% | **0.304** | **90.00%** |
| | 100% | | | 0.327 | 89.40% | **0.314** | **90.00%** |
| Video | 5% | **0.331** | 90.25% | 0.397 | 89.30% | 0.411 | 89.25% |
| | 10% | | | 0.398 | 89.00% | 0.471 | 88.25% |
| | 15% | | | 0.402 | 89.60% | 0.375 | **90.50%** |
| | 20% | | | 0.406 | 89.48% | 0.465 | 89.00% |
| | 100% | | | 0.389 | 88.80% | 0.397 | 89.25% |
| Baby | 5% | **0.346** | **88.89%** | 0.506 | 85.97% | 0.526 | 88.06% |
| | 10% | | | 0.464 | 87.39% | 0.468 | 87.50% |
| | 15% | | | 0.497 | 87.50% | 0.598 | 86.39% |
| | 20% | | | 0.479 | 87.11% | 0.550 | 87.22% |
| | 100% | | | 0.459 | 87.25% | 0.438 | 88.06% |
| Magazines | 5% | **0.238** | 92.53% | 0.350 | 91.31% | 0.248 | 92.27% |
| | 10% | | | 0.335 | 91.44% | 0.445 | 86.60% |
| | 15% | | | 0.318 | 91.44% | 0.269 | **92.78%** |
| | 20% | | | 0.325 | 91.42% | 0.310 | 92.27% |
| | 100% | | | 0.285 | 92.22% | 0.281 | **92.53%** |
| Software | 5% | **0.310** | 87.70% | 0.423 | 88.06% | 0.375 | **88.80%** |
| | 10% | | | 0.430 | 87.98% | 0.411 | **88.25%** |
| | 15% | | | 0.451 | 87.84% | 0.392 | **89.07%** |
| | 20% | | | 0.442 | 88.42% | 0.393 | **88.80%** |
| | 100% | | | 0.369 | **89.02%** | 0.392 | 87.43% |
| Sports | 5% | **0.309** | 87.50% | 0.364 | **88.37%** | 0.424 | 87.50% |
| | 10% | | | 0.338 | **88.30%** | 0.484 | 88.00% |
| | 15% | | | 0.357 | **87.97%** | 0.377 | 87.75% |
| | 20% | | | 0.365 | **88.27%** | 0.406 | 87.75% |
| | 100% | | | 0.390 | 88.13% | 0.355 | **89.50%** |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | All Layers | | | |
| | | | | Random Sampling* | | Active Learning | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| IMDb | 5% | **0.410** | 86.25% | 0.541 | 85.65% | 0.571 | 84.50% |
| | 10% | | | 0.549 | 85.83% | 0.590 | 85.50% |
| | 15% | | | 0.550 | 85.95% | 0.528 | **87.00%** |
| | 20% | | | 0.567 | 85.75% | 0.546 | 85.75% |
| | 100% | | | 0.562 | 84.90% | 0.547 | 85.00% |
| MR | 5% | 0.497 | 87.50% | 0.559 | 86.83% | 0.518 | 86.75% |
| | 10% | | | 0.558 | 87.53% | **0.469** | **89.25%** |
| | 15% | | | 0.555 | **87.57%** | 0.577 | 87.00% |
| | 20% | | | 0.558 | 87.55% | 0.503 | **87.75%** |
| | 100% | | | 0.543 | 86.83% | 0.649 | 87.25% |
| Avg. | 5% | **0.346** | **88.34%** | 0.448 | 87.53% | 0.411 | 87.87% |
| | 10% | | | 0.446 | 87.82% | 0.454 | 87.80% |
| | 15% | | | 0.445 | 87.95% | 0.440 | 88.33% |
| | 20% | | | 0.451 | 87.98% | 0.441 | 88.28% |
| | 100% | | | 0.422 | 88.07% | 0.429 | 88.09% |

**Table 5.6:** Performance of the Domain Specific Classifiers Using All-Layers Transfer Learning.

Table 5.6 shows the performance of the domain-specific models, trained by random sampling and by active learning in comparison to the general classifiers. In this approach, all layers continued training after the knowledge transfer from the general models.

In this approach too, the general classifier outperformed the domain-specific models and the active learning approach achieved better test accuracies, in average, across all data ratios in comparison to the random sampling method.

Tables 5.4, 5.5 and 5.6 showed that only the sentence-level approach produced domain-specific models that, in average, outperforms the general classifiers. The next experiments leverage this, by considering only the sentence-level approach.

To further evaluate the FT-HAN-BiLSTM architecture, four different methods combining active learning and outlier detection are employed.

| | | FT-HAN-BiLSTM | | | |
|---|---|---|---|---|---|
| | | Entropy Sampling | | Entropy Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 5% | 0.420 | **84.50%** | **0.399** | **84.50%** |
| | 10% | 0.484 | **85.50%** | **0.478** | 85.00% |
| | 15% | **0.487** | **86.00%** | **0.487** | 85.50% |
| | 20% | **0.464** | **86.25%** | 0.484 | 85.75% |
| | 100% | 0.497 | **86.25%** | **0.489** | 86.00% |
| Electronics | 5% | 0.322 | 89.00% | **0.303** | **89.50%** |
| | 10% | 0.373 | 88.00% | **0.365** | **88.75%** |
| | 15% | **0.389** | **88.50%** | 0.407 | 88.25% |
| | 20% | 0.383 | 88.50% | **0.373** | **89.00%** |
| | 100% | **0.322** | 88.75% | 0.326 | **89.25%** |
| DVD | 5% | 0.446 | **85.00%** | **0.430** | **85.00%** |
| | 10% | 0.543 | 84.50% | **0.499** | **85.25%** |
| | 15% | **0.550** | 83.75% | 0.574 | **84.50%** |
| | 20% | **0.518** | **84.25%** | 0.540 | 84.00% |
| | 100% | 0.510 | 84.50% | **0.508** | **84.75%** |
| Kitchen | 5% | 0.327 | 89.50% | **0.297** | **90.00%** |
| | 10% | **0.313** | **89.25%** | 0.316 | **89.25%** |
| | 15% | 0.290 | **89.75%** | **0.289** | 89.50% |
| | 20% | **0.292** | **89.25%** | 0.295 | 89.00% |
| | 100% | 0.315 | **89.50%** | **0.313** | **89.50%** |
| Apparel | 5% | **0.337** | 88.75% | 0.346 | **89.00%** |
| | 10% | **0.379** | **88.50%** | 0.392 | **88.50%** |
| | 15% | 0.360 | **88.50%** | **0.355** | **88.50%** |
| | 20% | 0.375 | **88.50%** | **0.370** | **88.50%** |
| | 100% | 0.416 | **87.75%** | **0.406** | **87.75%** |
| Camera | 5% | **0.259** | **90.00%** | 0.265 | 89.75% |
| | 10% | **0.272** | **90.50%** | 0.276 | 90.25% |
| | 15% | 0.280 | 90.25% | **0.262** | **90.75%** |
| | 20% | 0.286 | **90.00%** | **0.277** | **90.00%** |
| | 100% | **0.322** | **91.50%** | 0.324 | 90.50% |
| Health | 5% | **0.279** | **90.75%** | 0.288 | 90.50% |
| | 10% | **0.289** | **90.75%** | 0.295 | **90.75%** |
| | 15% | **0.292** | **90.50%** | 0.297 | **90.50%** |
| | 20% | **0.293** | 90.00% | **0.293** | **90.50%** |
| | 100% | **0.323** | **90.50%** | **0.323** | **90.50%** |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | |
|---|---|---|---|---|---|
| | | Entropy Sampling | | Entropy Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Music | 5% | **0.531** | **84.25%** | 0.533 | 84.00% |
| | 10% | 0.618 | **84.75%** | **0.545** | 84.50% |
| | 15% | 0.581 | **84.00%** | **0.531** | **84.00%** |
| | 20% | 0.588 | 84.50% | **0.581** | **84.75%** |
| | 100% | 0.529 | 83.75% | **0.518** | **84.25%** |
| Toys | 5% | 0.289 | **90.00%** | **0.284** | 89.25% |
| | 10% | **0.310** | 89.25% | 0.325 | **89.50%** |
| | 15% | **0.318** | **89.75%** | 0.333 | **89.75%** |
| | 20% | **0.307** | **89.50%** | 0.319 | 89.25% |
| | 100% | 0.322 | 90.25% | **0.313** | **90.50%** |
| Video | 5% | 0.370 | 89.75% | **0.367** | **90.00%** |
| | 10% | 0.411 | **89.00%** | **0.398** | **89.00%** |
| | 15% | **0.356** | 89.75% | 0.362 | **90.00%** |
| | 20% | **0.365** | 89.50% | 0.371 | **90.25%** |
| | 100% | 0.370 | 88.50% | **0.355** | **89.25%** |
| Baby | 5% | 0.391 | 88.33% | **0.386** | **89.17%** |
| | 10% | 0.411 | **89.17%** | **0.407** | 88.89% |
| | 15% | **0.418** | **89.72%** | 0.446 | 89.44% |
| | 20% | **0.384** | **89.17%** | **0.384** | **89.17%** |
| | 100% | 0.365 | 88.06% | **0.363** | **89.44%** |
| Magazines | 5% | **0.227** | **92.78%** | 0.242 | 92.53% |
| | 10% | **0.252** | **92.27%** | 0.264 | 91.75% |
| | 15% | 0.246 | **93.30%** | **0.244** | 93.04% |
| | 20% | **0.258** | **93.30%** | 0.259 | **93.30%** |
| | 100% | 0.252 | 92.78% | **0.241** | **93.04%** |
| Software | 5% | **0.356** | **88.52%** | 0.371 | 87.98% |
| | 10% | **0.335** | **88.80%** | 0.370 | **88.80%** |
| | 15% | 0.364 | 89.34% | **0.352** | **89.89%** |
| | 20% | 0.333 | 88.52% | **0.329** | **88.80%** |
| | 100% | **0.375** | 89.34% | **0.375** | **89.89%** |
| Sports | 5% | 0.338 | 87.50% | **0.333** | **88.25%** |
| | 10% | 0.378 | 87.75% | **0.370** | **88.25%** |
| | 15% | **0.339** | **88.50%** | 0.360 | **88.50%** |
| | 20% | **0.361** | **88.50%** | 0.366 | 88.00% |
| | 100% | 0.380 | **88.00%** | **0.368** | 86.75% |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | |
|---|---|---|---|---|---|
| | | Entropy Sampling | | Entropy Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| IMDb | 5% | 0.461 | 86.75% | **0.445** | **87.00%** |
| | 10% | 0.492 | 86.75% | **0.472** | **87.25%** |
| | 15% | 0.471 | **87.25%** | **0.468** | **87.25%** |
| | 20% | 0.496 | **86.75%** | **0.478** | **86.75%** |
| | 100% | **0.549** | **85.75%** | 0.559 | 85.25% |
| MR | 5% | 0.557 | **87.75%** | **0.535** | **87.75%** |
| | 10% | **0.580** | **88.00%** | 0.582 | 87.75% |
| | 15% | **0.555** | **86.75%** | 0.556 | 86.50% |
| | 20% | 0.564 | **86.50%** | **0.555** | **86.50%** |
| | 100% | 0.637 | **87.25%** | **0.626** | 87.00% |
| Avg. | 5% | 0.369 | 88.32% | **0.364** | **88.39%** |
| | 10% | 0.402 | 88.30% | **0.397** | **88.34%** |
| | 15% | **0.393** | 88.48% | 0.395 | **88.49%** |
| | 20% | **0.392** | 88.31% | **0.392** | **88.34%** |
| | 100% | 0.405 | 88.28% | **0.400** | **88.35%** |

**Table 5.7:** Performance of the Domain Specific Classifiers Using Sentence-Level Transfer Learning and Entropy Sampling.

Table 5.7 shows the performance of the domain-specific models using an active learning mechanism with entropy sampling. It uses maximum entropy as its prioritization score, which is explained in Section 3.11. The table consists of a comparison between using entropy sampling with and without a prior domain-specific outlier detection. The anomaly detection technique chosen and applied is Isolation Forests

In average, Table 5.7 shows that the entropy sampling method with anomaly detection using Isolation Forests achieves a higher test accuracy across all training-set ratios. It even achieved an accuracy of 88.49%, in average, when using 15% of the domain-specific training-sets, which is the highest test accuracy achieved across all the other approaches. [24].

| | | FT-HAN-BiLSTM | | | |
|---|---|---|---|---|---|
| | | Uncertainty Sampling | | Uncertainty Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 5% | **0.374** | **85.00%** | 0.381 | 84.50% |
| | 10% | 0.443 | **86.00%** | **0.440** | 85.75% |
| | 15% | **0.439** | **85.75%** | 0.452 | **85.75%** |
| | 20% | **0.479** | 85.50% | 0.487 | **86.50%** |
| | 100% | **0.475** | **86.00%** | 0.487 | **86.00%** |
| Electronics | 5% | 0.314 | 89.00% | **0.301** | **89.75%** |
| | 10% | 0.324 | 89.00% | **0.323** | **89.25%** |
| | 15% | 0.312 | **89.00%** | **0.310** | **89.00%** |
| | 20% | 0.325 | 88.75% | **0.323** | **89.50%** |
| | 100% | 0.325 | 89.25% | **0.321** | **89.75%** |
| DVD | 5% | **0.431** | 84.50% | 0.465 | **85.00%** |
| | 10% | 0.500 | **85.00%** | **0.480** | 84.75% |
| | 15% | 0.502 | **84.75%** | **0.483** | 84.50% |
| | 20% | **0.467** | **84.50%** | **0.467** | 84.00% |
| | 100% | **0.496** | **84.25%** | 0.511 | 83.75% |
| Kitchen | 5% | 0.282 | **89.75%** | **0.281** | **89.75%** |
| | 10% | 0.307 | **89.75%** | **0.300** | **89.75%** |
| | 15% | 0.298 | 88.75% | **0.297** | **89.50%** |
| | 20% | **0.300** | 88.50% | 0.304 | **89.25%** |
| | 100% | 0.318 | **89.25%** | **0.312** | 88.50% |
| Apparel | 5% | **0.350** | **88.50%** | **0.350** | **88.50%** |
| | 10% | 0.361 | **88.50%** | **0.358** | **88.50%** |
| | 15% | **0.360** | 88.25% | **0.360** | **88.50%** |
| | 20% | 0.385 | **88.00%** | **0.376** | 87.75% |
| | 100% | 0.413 | 87.25% | **0.395** | **88.25%** |
| Camera | 5% | 0.306 | **91.00%** | **0.297** | 90.75% |
| | 10% | 0.283 | **91.25%** | **0.280** | **91.25%** |
| | 15% | 0.301 | 90.75% | **0.298** | **91.25%** |
| | 20% | **0.303** | **91.50%** | 0.304 | 91.25% |
| | 100% | 0.328 | 90.25% | **0.322** | **91.25%** |
| Health | 5% | **0.282** | 91.00% | 0.287 | **91.25%** |
| | 10% | 0.307 | 90.50% | **0.294** | **90.75%** |
| | 15% | 0.339 | 90.00% | **0.333** | **90.50%** |
| | 20% | **0.336** | **90.50%** | 0.346 | 90.25% |
| | 100% | **0.328** | **90.75%** | **0.328** | 90.50% |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | |
|---|---|---|---|---|---|
| | | Uncertainty Sampling | | Uncertainty Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Music | 5% | 0.520 | **84.75%** | **0.506** | 84.50% |
| | 10% | 0.524 | **84.25%** | **0.507** | 84.00% |
| | 15% | **0.552** | **84.75%** | 0.554 | **84.75%** |
| | 20% | 0.536 | **84.50%** | **0.531** | 84.25% |
| | 100% | 0.515 | 84.50% | **0.500** | **85.50%** |
| Toys | 5% | **0.276** | 88.75% | 0.284 | **89.50%** |
| | 10% | **0.302** | 89.75% | 0.317 | **90.00%** |
| | 15% | 0.316 | 89.75% | **0.308** | **90.00%** |
| | 20% | 0.321 | **90.00%** | **0.316** | **90.00%** |
| | 100% | **0.313** | 89.75% | 0.315 | **90.25%** |
| Video | 5% | 0.346 | **90.00%** | **0.344** | 89.25% |
| | 10% | **0.348** | 89.50% | 0.353 | **89.75%** |
| | 15% | 0.368 | 89.50% | **0.362** | **90.00%** |
| | 20% | 0.357 | 90.25% | **0.352** | **90.50%** |
| | 100% | 0.376 | 88.00% | **0.373** | **88.25%** |
| Baby | 5% | **0.356** | **88.61%** | 0.358 | **88.61%** |
| | 10% | 0.367 | **88.06%** | **0.365** | **88.06%** |
| | 15% | **0.379** | **88.89%** | 0.386 | 88.06% |
| | 20% | 0.375 | 87.78% | **0.373** | **88.06%** |
| | 100% | 0.369 | **89.17%** | **0.363** | 88.61% |
| Magazines | 5% | **0.222** | **93.30%** | 0.228 | 92.78% |
| | 10% | 0.233 | **92.78%** | **0.232** | **92.78%** |
| | 15% | 0.231 | **93.04%** | **0.229** | **93.04%** |
| | 20% | 0.232 | **92.78%** | **0.225** | **92.78%** |
| | 100% | **0.234** | **93.04%** | 0.236 | **93.04%** |
| Software | 5% | **0.306** | 87.70% | 0.331 | **88.25%** |
| | 10% | **0.343** | **87.98%** | 0.358 | **87.98%** |
| | 15% | **0.373** | 87.98% | 0.376 | **88.80%** |
| | 20% | **0.358** | 87.98% | 0.366 | **88.52%** |
| | 100% | 0.347 | **89.07%** | **0.340** | 88.80% |
| Sports | 5% | **0.309** | **88.00%** | 0.314 | **88.00%** |
| | 10% | **0.345** | **88.00%** | 0.350 | 87.50% |
| | 15% | **0.340** | **87.00%** | 0.346 | 86.75% |
| | 20% | **0.359** | 87.25% | 0.367 | **87.75%** |
| | 100% | **0.373** | 86.75% | 0.381 | **87.25%** |

*Continued on next page*

| | | FT-HAN-BiLSTM | | | |
|---|---|---|---|---|---|
| | | Uncertainty Sampling | | Uncertainty Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| IMDb | 5% | **0.443** | 86.25% | 0.449 | **86.75%** |
| | 10% | 0.459 | **86.75%** | **0.440** | 86.50% |
| | 15% | **0.467** | 86.50% | 0.471 | **86.75%** |
| | 20% | **0.483** | 86.25% | 0.484 | **86.75%** |
| | 100% | 0.555 | 86.00% | **0.534** | **86.25%** |
| MR | 5% | **0.515** | 86.50% | 0.518 | **86.75%** |
| | 10% | 0.520 | **86.50%** | **0.518** | 86.25% |
| | 15% | 0.527 | **86.50%** | **0.522** | **86.50%** |
| | 20% | 0.573 | 85.75% | **0.566** | **86.00%** |
| | 100% | 0.674 | 86.25% | **0.637** | **86.75%** |
| Avg. | 5% | **0.352** | 88.29% | 0.356 | **88.37%** |
| | 10% | 0.373 | **88.35%** | **0.370** | 88.30% |
| | 15% | 0.381 | 88.20% | **0.380** | **88.35%** |
| | 20% | **0.387** | 88.11% | **0.387** | **88.32%** |
| | 100% | 0.402 | 88.10% | **0.397** | **88.29%** |

**Table 5.8:** Performance of the Domain Specific Classifiers Using Sentence-Level Transfer Learning and Uncertainty Sampling.

Table 5.8 shows the performance of the domain-specific models using an active learning mechanism with uncertainty sampling. It uses least confidence as its prioritization score, which is explained in Section 3.11. The table also shows the effect of using Isolation Forests on the performance of the domain-specific models.

Table 5.8 further confirms that using the outlier detection prior to training the domain-specific models yields higher test accuracies across almost all experiments. However, only during the experiment using 10% of the training-set, the approach using uncertainty sampling without performing the Isolation Forest algorithm achieved a higher accuracy (88.35%) than its counterpart (88.30%).

The results presented in Tables 5.8 and 5.7 support the idea that active learning with entropy sampling outperforms the uncertainty sampling method. Moreover, using Isolation Forests prior to training further enhances the final evaluation of the models on the test-sets. Only during the experiment using 10% of the training-set, the uncertainty sampling method without outlier detection outperformed the other three approaches. However, the experiment using 15% of the domain-specific training-sets, achieved the highest test accuracy across all the other approaches (88.49%).

## 5.5 Hyperparameter Optimization

Hyperparameter optimization is a technique used to find the optimal hyperparameter values for the model to solve the underlying problem and perform better in its task.
To improve the found results even further, the hyperparameters of the FT-HAN-BiLSTM model were optimized using the KerasTuner framework [34], as shown in the previous chapter.

Using the hyperband algorithm and the loss of the validation-set as a measure, the tuner's search method produced the following results:

**Listing 5.1:** Output of the Hyperparameter Optimization Search Method.

```
Best val_loss So Far: 0.2746671438217163
Total elapsed time: 2d 19h 27m 04s

Search: Running Trial #254

Value                |Best Value So Far |Hyperparameter
25                   |50                |units_1
100                  |100               |attention_dim_1
75                   |50                |units_2
25                   |75                |units_3
100                  |50                |attention_dim_2
0.05                 |0.15              |rate
0.001192             |0.00014026        |lr
100                  |100               |tuner/epochs
0                    |0                 |tuner/initial_epoch
0                    |0                 |tuner/bracket
0                    |0                 |tuner/round
```

The FT-HAN-BiLSTM Model was re-trained and all the steps from the previously defined pipeline were re-run using the hyperparameters found by the hyperband algorithm. As Listing 5.1 shows, the new chosen hyperparameters are as follows:

- The number of units of the word-level GRU layer $units_1 = 50$
- The attention dimension of the word-level attention layer $attention\_dim_1 = 100$
- The number of units of the first sentence-level BiLSTM layer $units_2 = 50$
- The number of units of the secont sentence-level BiLSTM layer $units_3 = 75$
- The attention dimension of the sentence-level attention layer $attention\_dim_2 = 50$
- The rate of the dropout layer $rate = 0.15$
- The starting learning rate of the Adam optimizer $lr = 0.00014026$

### 5.5.1 General Classifier

Table 5.9 shows a comparison between the general classifiers of the initial FT-HAN-BiLSTM model and the one with the optimized hyperparameters. In average, the initial model has a

| | FT-HAN-BiLSTM | | FT-HAN-BiLSTM (opt.) | |
|---|---|---|---|---|
| Domain | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Books | 0.430 | 87.00% | 0.376 | 86.00% |
| Electronics | 0.297 | 88.75% | 0.302 | 87.25% |
| DVD | 0.413 | 84.25% | 0.342 | 87.75% |
| Kitchen | 0.277 | 89.00% | 0.296 | 87.00% |
| Apparel | 0.344 | 88.75% | 0.316 | 86.75% |
| Camera | 0.260 | 91.00% | 0.250 | 91.25% |
| Health | 0.286 | 91.00% | 0.272 | 89.75% |
| Music | 0.463 | 84.75% | 0.409 | 83.25% |
| Toys | 0.325 | 88.50% | 0.316 | 88.00% |
| Video | 0.331 | 90.25% | 0.309 | 87.00% |
| Baby | 0.346 | 88.89% | 0.308 | 86.67% |
| Magazines | 0.238 | 92.53% | 0.268 | 90.21% |
| Software | 0.310 | 87.70% | 0.302 | 87.43% |
| Sports | 0.309 | 87.50% | 0.281 | 87.50% |
| IMDb | 0.410 | 86.25% | 0.287 | 89.00% |
| MR | 0.497 | 87.50% | 0.364 | 86.25% |
| Avg | 0.346 | 88.34% | 0.313 | 87.57% |

**Table 5.9:** Comparison of FT-HAN-BiLSTM General Classifier's Performance between initial and Optimized Hyperparameters.

higher test accuracy of 88.34% in comparison to the second one, which achieved an accuracy of 87.57%. However, the model with the optimized hyperparameters has a lower average test loss of 0.313.

### 5.5.2 Domain-specific Classifier

In order to produce the domain-specific models with the hyperparameters chosen by the hyperband algorithm. Only the sentence-level transfer learning approach was performed in this case, since it was proven to be the only method under the three previously proposed ones to achieve better results than the general classifiers.

At first, the four active learning approaches previously applied using the initial FT-HAN-BiLSTM model are executed again on the new model with the optimized hyperparameters. Second, the experiments using the transfer learning with random sampling approach are conducted. Finally, the results from the most performing active learning approach are compared to the test losses and accuracies from random sampling and the general classifier.

| | | FT-HAN-BiLSTM (opt.) | | | |
|---|---|---|---|---|---|
| | | Entropy Sampling | | Entropy Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Avg. | 5% | 0,324 | 88,20% | **0,320** | **88,22%** |
| | 10% | **0,331** | **88,35%** | 0,334 | 88,07% |
| | 15% | 0,346 | **88,17%** | **0,345** | 88,14% |
| | 20% | 0,345 | **88,04%** | **0,344** | 88,02% |
| | 100% | **0,331** | 88,07% | 0,330 | **88,24%** |

**Table 5.10:** Average Performance of the Domain Specific Classifiers Using Sentence-Level Transfer Learning and Entropy Sampling with Optimized Hyperparameters.

Table 5.10 shows the average test accuracies and losses of FT-HAN-BiLSTM model using the sentence-level approach and entropy sampling with optimized hyperparameters. The table shows that in most experiments, entropy sampling without Isolation Forests outperformed its counterpart, as opposed to the initial FT-HAN-BiLSTM model. It is also interesting to note, that the results achieved by this model do not surpass those of the initial one.

| | | FT-HAN-BiLSTM (opt.) | | | |
|---|---|---|---|---|---|
| | | Uncertainty Sampling | | Uncertainty Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Avg. | 5% | **0,318** | 88,14% | **0,318** | **88,26%** |
| | 10% | **0,324** | **88,15%** | 0,325 | 88,12% |
| | 15% | **0,325** | 88,03% | 0,327 | **88,16%** |
| | 20% | **0,324** | 88,00% | **0,324** | **88,12%** |
| | 100% | **0,329** | 88,04% | **0,329** | **88,25%** |

**Table 5.11:** Average Performance of the Domain Specific Classifiers Using Sentence-Level Transfer Learning and Uncertainty Sampling with Optimized Hyperparameters.

Table 5.11 shows the average test accuracies and losses of FT-HAN-BiLSTM model using the sentence-level approach and uncertainty sampling with optimized hyperparameters. The table shows that in all experiments, uncertainty sampling without Isolation Forests has a lower test loss. However, it only had a higher test accuracy in the experiment using 5% of the domain-specific training-set. In all other experiments, uncertainty sampling with outlier detection reached a higher test accuracy attaining 88.26% and 88.25% in the experiments using

5% and 100% of the training-set, respectively.
Nevertheless, this approach does not exceed the results attained by the initial FT-HAN-BiLSTM model using the sentence-level approach with entropy sampling and Isolation Forests.

Table 5.12 shows a comparison of the results produced by the general classifier, the random sampling experiments and the uncertainty sampling with Isolation Forests experiments conducted using the FT-HAN-BiLSTM architecture with the optimized hyperparameters.

| | | FT-HAN-BiLSTM (opt.) | | | | | |
|---|---|---|---|---|---|---|---|
| | | General | | Random Sampling | | Uncertainty Sampling with IF | |
| Domain | Ratio of Training-set | Test Loss | Test Acc. | Test Loss | Test Acc. | Test Loss | Test Acc. |
| Avg. | 5% | **0.313** | 87.57% | 0.337 | 87.88% | 0.318 | **88.26%** |
| | 10% | | | 0.344 | 87.83% | 0.325 | **88.12%** |
| | 15% | | | 0.347 | 87.82% | 0.327 | **88.16%** |
| | 20% | | | 0.345 | 87.92% | 0.324 | **88.12%** |
| | 100% | | | 0.329 | 88.12% | 0.329 | **88.25%** |

**Table 5.12:** Average Performance of the Domain Specific Classifiers Using Sentence-Level Transfer Learning with optimized Hyperparameters.

Table 5.12 shows that the domain-specific models trained with active learning using uncertainty sampling with prior outlier detection outperforms the models trained with random sampling and the general classifiers. However, these models do not achieve better results than the models trained with the initial FT-HAN-BiLSTM architecture, before hyperparameter optimization.

For further comparisons, only the initial FT-HAN-BiLSTM general classifier and the model using entropy sampling with Isolation Forests are considered, since they produced the best results.

## 5.6 Comparison with the State-of-the-Art

In the following, the results of the FT-HAN-BiLSTM model are compared to other models from the state-of-the-art. Most of these models are mentioned in the Related Work section 2. They are listed again below:

- BERT is a pre-trained language model with deep bidirectional transformer [7]. The results shown are from a pre-trained BERT-base model that was fine-tuned by Cai et al. [6].
- ASP-MTL is an adversarial multi-task learning model [3].
- SA-MTL and DA-MTL are two proposed models by Zheng et al. that also use multi-task learning [5].

- DSR-at is another adversarial training model used on a BiLSTM representation [4].
- Cai and Wan 2019 [6]

It should be noted that the results of the models from current research were extracted from the 2018 paper by Cai and Wan, titled "Multi-Domain Sentiment Classification Based on Domain-Aware Embedding and Attention" [6].
The results shown in the last column of Table 5.13 correspond to the results of the active learning experiment using up to 100% of the domain-specific data. This is due to the fact that the results from the other models in the table were all produced using the totality of the available data.

| Domain | BERT | ASP-MTL | SA-MTL | DA-MTL | DSR-at | Cai and Wan 2019 | FT-HAN-BiLSTM (General) | FT-HAN-BiLSTM (Entropy Sampling with IF) |
|---|---|---|---|---|---|---|---|---|
| Books | 87.00% | 87.00% | 86.80% | 88.50% | **89.10%** | 89.00% | 87.00% | 86.00% |
| Electronics | 88.30% | 89.00% | 87.50% | 89.00% | 87.90% | **91.80%** | 88.75% | 89.25% |
| DVD | 85.60% | 87.40% | 87.30% | 88.00% | 88.10% | **88.30%** | 84.25% | 84.75% |
| Kitchen | **91.00%** | 87.20% | 89.30% | 89.00% | 85.90% | 90.30% | 89.00% | 89.50% |
| Apparel | **90.00%** | 88.70% | 87.30% | 88.80% | 87.80% | 89.00% | 88.75% | 87.75% |
| Camera | 90.00% | 91.30% | 90.30% | 91.80% | 90.00% | **92.00%** | 91.00% | 90.50% |
| Health | 88.30% | 88.10% | 88.30% | 90.30% | **92.90%** | 89.80% | 91.00% | 90.50% |
| Music | 86.80% | 82.60% | 84.00% | 85.00% | 84.10% | **88.00%** | 84.75% | 84.25% |
| Toys | 91.30% | 88.80% | 89.30% | 89.50% | 85.90% | **91.80%** | 88.50% | 90.50% |
| Video | 88.00% | 85.50% | 88.50% | 89.50% | 90.30% | **92.30%** | 90.25% | 89.25% |
| Baby | 91.50% | 89.80% | 88.80% | 90.50% | 91.70% | **92.30%** | 88.89% | 89.44% |
| Magazine | 92.80% | 92.50% | 92.00% | 92.00% | 92.10% | **96.50%** | 92.53% | 93.04% |
| Software | 89.30% | 87.30% | 89.30% | 90.80% | 87.00% | **92.80%** | 87.70% | 89.89% |
| Sports | **90.80%** | 86.70% | 89.80% | 89.80% | 85.80% | **90.80%** | 87.50% | 86.75% |
| IMDb | 85.80% | 85.80% | 87.50% | 89.80% | **93.80%** | 90.80% | 86.25% | 85.25% |
| MR | 74.00% | 77.30% | 73.00% | 75.50% | 73.30% | 77.00% | **87.50%** | 87.00% |
| Avg | 88.10% | 87.20% | 87.60% | 88.20% | 87.90% | **90.10%** | 88.34% | 88.35% |

**Table 5.13:** Results of Multi-domain Sentiment Classification.

Results of the proposed FT-HAN-BiLSTM architecture alongside other models from current research are listed in Table 5.13. Cai and Wan 2019 [6] achieves a higher accuracy in 10 of the 16 domains and reaches an average of 90.10%, which is the highest among the compared models. Both FT-HAN-BiLSTM models outperform the others in the MR domain with more than 10 percentage points difference, with the general FT-HAN-BiLSTM model reaching 87.50% in this domain. The DSR-at model has the highest test accuracies in the Books, Health and IMDb domains, while the pre-trained BERT model outperforms in the Kitchen, Apparel and Sports domains.
Even though, the FT-HAN-BiLSTM models have the lead in only one domain, they achieve the second and third highest accuracies across all models. The proposed model using entropy sampling with Isolation Forests reached an average accuracy of 88.35%, while the general classifier reached 88.34%.
Overall, the proposed models produced very good results compared to the state-of-the-art.

# 6 Conclusion

The purpose of this thesis is to utilize an Active Learning mechanism, in order to improve the training of domain-specific models. This work proposes three novel architectures FT-HAN, FT-HAN-BiLSTM and FT-HAN-BiLSTM-AE to solve the multi-domain sentiment analysis problem.
The architectures extend the Hierarchical Attention Networks (HAN) solution proposed by Yang et al. [1], by utilizing word embeddings produced by the FastText library [2] on the totality of the available data in the datasets. FT-HAN-BiLSTM adds a BiLSTM network consisting of two layers before the sentence-level attention mechanism in the HAN framework, while FT-HAN-BiLSTM-AE extends the architecture even further by pre-training an auto-encoder model on the training data and adding the encoder part to the architecture.

Experiments were conducted on 16 different domains, where the proposed models were trained on general data hailing from all categories, in order to produce general classifiers. The performance of these classifiers showed that the FT-HAN-BiLSTM model reached the best test accuracy of 88.34%, compared to FT-HAN (87.44%) and FT-HAN-BiLSTM-AE (85.05%).

The FT-HAN-BiLSTM model was chosen to conduct the next experiments, which consist of transferring the knowledge learned by the general classifier to domain-specific models. Comparison of the different transfer learning approaches showed that the sentence-level fine-tuning with the frozen word-level layers performed better than its counterparts.
In order to simulate the lack of labelled data in the datasets, only a portion of the available domain-specific data was used for training. Five different experiments using 5% (70 observations), 10%, 15%, 20% and 100% of the data, respectively, were conducted on each domain. The goal of these experiments is to achieve a high test accuracy with a small number of observations and to examine the impact of increasing the training set on the final results. These observations were, at first, randomly sampled from the dataset. Then, Active Learning was applied to actively select the most informative instances. Two different Active Leaning approaches, which are Uncertainty Sampling and Entropy Sampling, were used with and without prior outlier detection using Isolation Forests (IF). The Experiments showed that the Entropy Sampling with IF method yields the highest test accuracies across the different trials. The domain-specific models achieved, in average, 88.39%, 88.34%, 88.49%, 88.34% and 88.35% when using 5%, 10%, 15%, 20% and 100% of the data, respectively.

The general FT-HAN-BiLSTM classifier and the domain-specific models produced by Active Learning with Entropy Sampling and Isolation Forests were compared to other solutions from current research. The proposed models outperformed the state-of-the-art in the MR domain with more than 10 percentage points difference, with the general FT-HAN-BiLSTM model reaching 87.50% test accuracy, while the domain-specific one reached 87.00%. Even though the FT-HAN-BiLSTM models have the lead in only one domain, they achieve the second and third highest average test accuracies across all models. Overall, the proposed models produced very good results compared to the state-of-the-art.

There are currently not a lot of research on Active Learning in multi-domain sentiment analysis. This thesis showed that the Active Learning approach improved the FT-HAN-BiLSTM model's performance on the domain-specific tasks and that there is still some potential and room for improvement in this area, and that the development of novel models to further push the boundaries of this task.

# List of Tables

# List of Figures

# Bibliography

[1] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, “Hierarchical attention networks for document classification,” in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, 2016, pp. 1480–1489.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *arXiv preprint arXiv:1607.04606*, 2016.

[3] P. Liu, X. Qiu, and X. Huang, “Adversarial multi-task learning for text classification,” *arXiv preprint arXiv:1704.05742*, 2017.

[4] Q. Liu, Y. Zhang, and J. Liu, “Learning domain representation for multi-domain sentiment classification,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 541–550.

[5] R. Zheng, J. Chen, and X. Qiu, “Same representation, different attentions: Shareable sentence representation learning from multiple tasks,” *arXiv preprint arXiv:1804.08139*, 2018.

[6] Y. Cai and X. Wan, “Multi-domain sentiment classification based on domain-aware embedding and attention.” in *IJCAI*, 2019, pp. 4904–4910.

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

[8] S. Li and C. Zong, “Multi-domain sentiment classification,” in *Proceedings of ACL-08: HLT, Short Papers*, 2008, pp. 257–260.

[9] R. Caruana, “Multitask learning,” *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[10] X. Su, R. Li, and X. Li, “Multi-domain transfer learning for text classification,” in *CCF International Conference on Natural Language Processing and Chinese Computing*. Springer, 2020, pp. 457–469.

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

[12] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen netzen,” *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[13] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional lstm and other neural network architectures,” *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[15] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[17] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84–96, 2014.

[18] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[19] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," *Advances in neural information processing systems*, vol. 20, 2007.

[20] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[23] C. C. Aggarwal, *Outlier analysis*. Springer, 2017.

[24] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[25] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[26] B. Settles, "Active learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 6, no. 1, pp. 1–114, 2012.

[27] J. Blitzer, M. Dredze, and F. Pereira, "Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification," in *Proceedings of the 45th annual meeting of the association of computational linguistics*, 2007, pp. 440–447.

[28] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[29] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," *arXiv preprint cs/0506075*, 2005.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[31] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.

[32] T. Danka and P. Horvath, "modAL: A modular active learning framework for Python," available on arXiv at https://arxiv.org/abs/1805.00979. [Online]. Available: https://github.com/cosmic-cortex/modAL

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[34] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Kerastuner," https://github.com/keras-team/keras-tuner, 2019.

[35] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[36] "Anaconda software distribution," 2020. [Online]. Available: https://docs.anaconda.com/