# ARXIV PAPER CLASSIFICATION USING CNN WITH GLOVE EMBEDDINGS

**Sinjoy Saha**
Penn State University
sks7620@psu.edu

**Xin Dong**
Penn State University
xjd5036@psu.edu

## 1 Introduction and Related Work

Accurate classification of academic papers is critical for enhancing information retrieval systems, optimizing recommendation algorithms, and identifying emerging research trends. To achieve this, a structured classification pipeline is essential, beginning with text representation and feature extraction. The first step involves transforming textual data into numerical representations, a process known as word embedding, which enables efficient processing by machine learning models. Subsequently, a classification model is employed to extract meaningful features from these representations, allowing it to learn patterns and accurately categorize papers into predefined domains. While this pipeline provides a robust framework, the choice of text representation and classification methods significantly impacts the accuracy and scalability of the system.

There are multiple methods for obtaining word embeddings, each with distinct advantages depending on the application. Traditional approaches include one-hot encoding, where each word is represented as a high-dimensional sparse vector, and term frequency-inverse document frequency (TF-IDF), which weighs words based on their importance in a corpus. However, these methods fail to capture semantic relationships between words. More advanced techniques leverage dense vector representations learned from large corpora. Word2Vec [1], for example, utilizes the Skip-gram and Continuous Bag-of-Words (CBOW) models to learn word vectors based on contextual co-occurrence. Similarly, GloVe [2] constructs word embeddings by factorizing a word co-occurrence matrix, capturing both local and global statistical patterns in text. Recent deep learning-based embeddings, such as those produced by transformer models like BERT [3] and GPT [4], generate contextualized word representations. Unlike static embeddings, these models capture word meanings dynamically, depending on the surrounding context. In this project, we employed GloVe[2] to transform raw textual data into a numerical format that preserves semantic information, thereby serving as an effective input for the subsequent classification model.

Existing research on paper classification has explored various approaches, ranging from traditional machine-learning techniques to deep learning-based models. Early studies relied on bag-of-words (BoW) and TF-IDF representations combined with classifiers such as support vector machines (SVMs) and random forests [5, 6]. However, these methods often fail to capture contextual and semantic relationships between words. With the advancement of deep learning, models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have demonstrated superior performance in text classification tasks [7, 8]. CNNs are particularly effective in capturing local patterns and n-gram features, while RNNs excel at modeling sequential dependencies in textual data. Recent studies have also explored transformer-based models such as BERT [3], which leverage contextualized embeddings for enhanced text understanding.

## 2 Problem Statement

In this work, we address the challenge of accurately classifying research papers by exploring the strength of convolutional neural networks (CNNs). The CNN model is employed to efficiently extract hierarchical and local features from textual data. To enhance the semantic representation of the input data, we use GloVe embeddings [2], which transform words into dense vector representations enriched with pre-trained semantic knowledge.

We evaluate the performance of these architectures in classifying research papers into predefined categories. For this purpose, we utilize the ArXiv metadata dataset [9], which provides a comprehensive and structured collection of research papers, enabling the extraction of key features essential for accurate classification. Additionally, to ensure reproducibility and transparency, we have made the code files for this project publicly available on our GitHub repository, accompanied by detailed documentation to facilitate the replication of our results [1] [2]. The repository contains code for dataset curation, model development, training and testing, and visualization of experimental results. Moreover, it contains model checkpoints (files over 100MB have been compressed and the largest model is not committed due to GitHub's limitation ), and PyTorch files for train, validation, and test splits of the dataset.

## 3 Dataset

The ArXiv metadata dataset is an openly accessible resource provided by Cornell University and available via Kaggle [9]. Spanning from 2007 to 2025, this dataset serves as a comprehensive pre-print repository covering a wide range of disciplines, including computer science, mathematics, and physics. Structured in JSON format, it contains rich bibliographic details for each paper, such as titles, abstracts, authors, categories, and publication dates. This wealth of structured information makes the dataset particularly well-suited for tasks such as paper classification, citation analysis, and academic trend detection. Specifically, the dataset includes the following key attributes:

- id: A unique identifier for each paper in the arXiv repository.
- submitter: The user who submitted the paper.
- authors: A string listing all authors of the paper.
- title: The title of the paper.
- comments: Additional comments or notes about the paper, often including details about versions or conference submissions.
- journal-ref: If applicable, the reference to a published version in a journal.
- doi: The Digital Object Identifier (DOI) of the paper, if available.
- report-no: Any report number associated with the submission.
- categories: The list of subject categories assigned to the paper (e.g., "cs.CV" for Computer Vision).
- license: The type of license under which the paper is published.
- abstract: A brief summary of the paper's content.
- versions: A list of submission versions, capturing the paper's revision history.
- update_date: The last update timestamp of the paper's metadata.
- authors_parsed: A structured version of the authors list, breaking down names into individual components.
- primary_category: The main subject category of the paper.

In this project, we filtered the dataset to include only papers from the computer science domain for further analysis. As illustrated in Figure 1, the dataset encompasses over 61 distinct categories within computer science. To maintain focus and ensure a manageable scope, we narrowed our study to the five most prominent categories: **Computer Vision, Computation and Language, Machine Learning, Robotics, Cryptography and Security**. To ensure consistency and relevance, we restricted our dataset to papers published between 2015 and 2020. Additionally, to evaluate the model's generalization ability across different periods, we included papers from 2023 as an independent test set. While this study concentrates on these five categories, future work could extend the scope to incorporate additional categories, enabling a more comprehensive exploration of research topics and temporal trends. After processing, the volume for papers across the five selected categories during the five years (2015–2020) is as follows:

- cs.CV (Computer Vision): 112,317 papers (34.76%)
- cs.CL (Computation and Language): 59,447 papers (18.4%)
- cs.LG (Machine Learning): 99,416 papers (30.77%)
- cs.RO (Robotics): 27,966 papers (8.65%)
- cs.CR (Cryptography and Security): 24,961 papers (7.72%)

[1] `https://github.com/sinjoysaha/CSE587-midterm-project.git`
[2] `https://github.com/XDNG2024/CSE587-midterm-project.git`

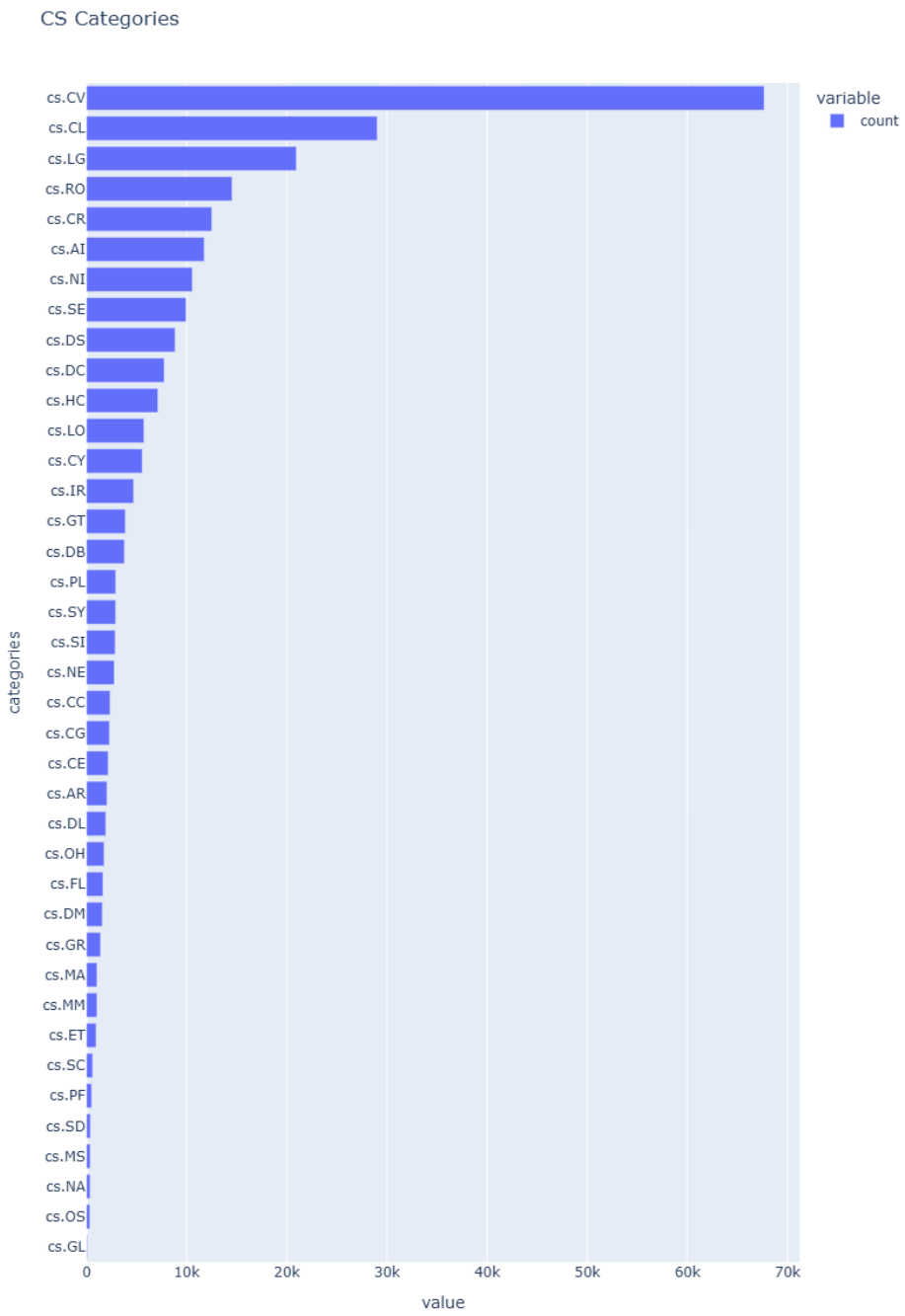



Figure 1: Paper Categories in Computer Science

The distribution of papers across different years is illustrated in Figure 2. The overall trend reveals a steady increase in the number of publications, with Computer Vision (cs.CV) and Machine Learning (cs.LG) emerging as the most dominant fields. Computation and Language (cs.CL) and Robotics (cs.RO) maintain a consistent presence, while Cryptography and Security (cs.CR) exhibit gradual growth. The most significant expansion occurred between 2017 and 2019, with 2020 marking the highest publication volume, reflecting the accelerating pace of research in these areas.

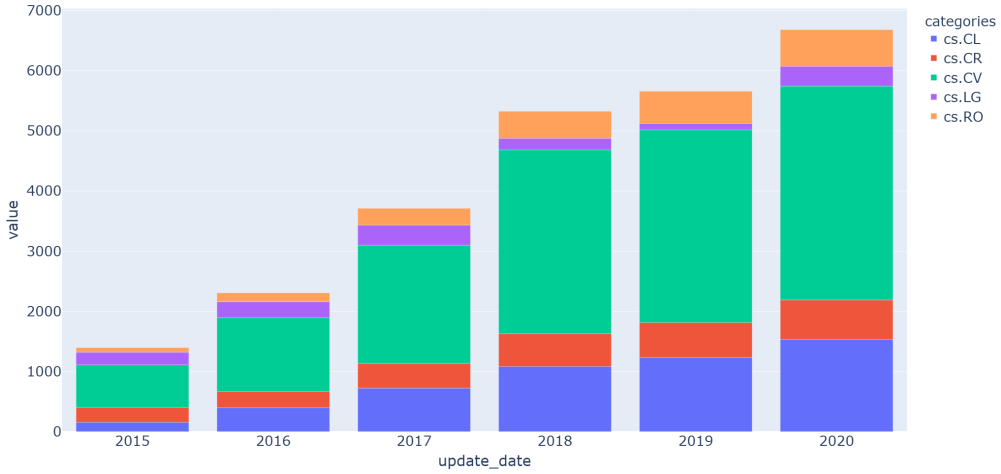



Figure 2: Dataset Distribution

## 4 Methodologies

This section describes the methods used for text classification in our study. We first present the word embedding technique employed to represent text as numerical vectors. Next, we detail the neural network architectures used: a convolutional neural network (CNN) to extract local patterns from text. This model leverages pre-trained embeddings and is trained for multi-class classification.

### 4.1 Word Embedding

The embedding layer transforms a sequence of words into numerical representations. The embeddings of the word could either be frozen or fine-tuned. Given a sequence of words $w_1, w_2, \ldots, w_T$, each word is mapped to a vector using an embedding matrix $W$:

$$x_j = W \cdot e_j, \quad x_j \in \mathbb{R}^k \tag{1}$$

where $W \in \mathbb{R}^{V \times k}$ is the pre-trained embedding matrix, $e_j$ is a one-hot vector for the $j^{th}$ word in the vocabulary, - $x_j$ is the $k$-dimensional embedding of the $j^{th}$ word.

For a given input sequence of length $T$, the embedding layer outputs:

$$X = [x_1, x_2, \ldots, x_T] \in \mathbb{R}^{T \times k} \tag{2}$$

### 4.2 CNN

To extract local patterns in textual data, we employ a convolutional neural network (CNN) for document classification. The overall architecture is depicted in Figure 3.

Given an input sequence of word embeddings $X = [x_1, x_2, \ldots, x_T]$, the model applies a series of 1D convolutional layers followed by non-linear activation functions to capture local contextual dependencies.

The convolution operation is defined as:

$$h_i = f(W_c \cdot X_{i:i+h-1} + b_c) \tag{3}$$

where $W_c$ represents the convolutional filter, $X_{i:i+h-1}$ is a window of embeddings, $b_c$ is the bias term, and $f(\cdot)$ denotes the activation function (ReLU).

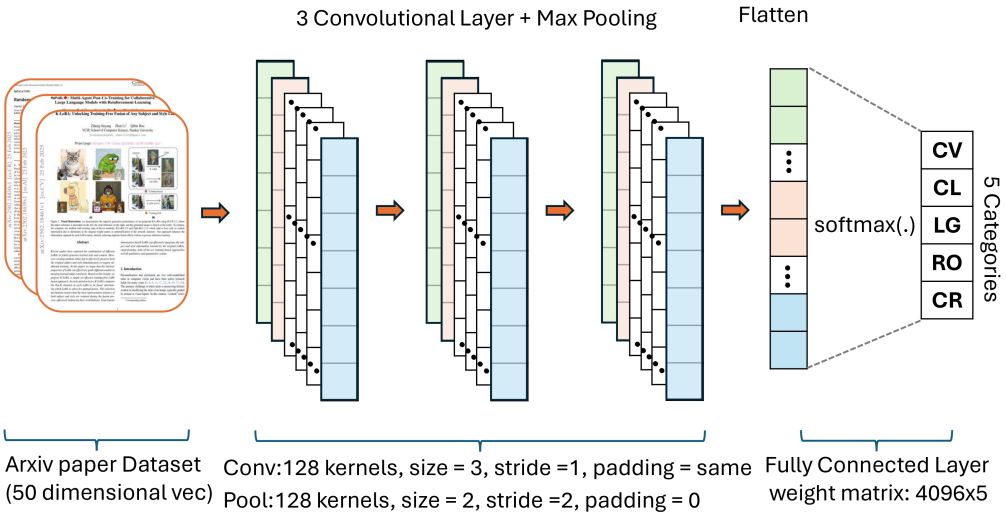



Figure 3: CNN Model Structure

To refine extracted features and improve computational efficiency, max-pooling layers are applied after each convolutional layer:

$$h_{\text{pool}} = \max(h) \tag{4}$$

where $h_{\text{pool}}$ represents the most significant feature for each filter, effectively reducing the sequence length while retaining essential information. Following the final convolution and pooling layers, the feature maps are flattened into a vector representation:

$$h_{\text{flat}} = \text{Flatten}(h_{\text{pool}}) \tag{5}$$

This flattened representation is then passed through a fully connected layer to generate classification logits:

$$\hat{y}_i = \text{softmax}(W_{\text{fc}} \cdot h_{\text{flat}} + b_{\text{fc}}) \tag{6}$$

where $W_{\text{fc}}$ and $b_{\text{fc}}$ are trainable parameters, and softmax normalizes the logits into a probability distribution over classification categories.

### 4.3 Optimization

To optimize the classification model, we employ cross-entropy loss as the objective function, which measures the divergence between the predicted probability distribution and the true class labels. Cross-entropy loss is suitable for multi-class classification as it penalizes incorrect predictions while encouraging the model to assign a high probability to the correct class. The loss function is then minimized using the Adam optimizer, which provides adaptive learning rates for efficient and stable convergence:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \hat{y}_{ij} \tag{7}$$

where $y_{ij}$ is the ground truth label, and $\hat{y}_{ij}$ is the predicted probability for class $j$ in sample $i$. The optimization process employs the Adam optimizer for efficient and stable convergence.

### 4.4 Adam Optimizer

To minimize the cross-entropy loss $\mathcal{L}$, we employ the Adam optimizer [10], which integrates momentum-based optimization and adaptive learning rates for efficient parameter updates. Adam maintains moving averages of both the gradient (first moment estimate) and its squared values (second moment estimate), which are updated as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{8}$$

where $g_t = \nabla_\theta \mathcal{L}_t$ represents the gradient of the loss function at time step $t$. Bias correction is then applied:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{9}$$

Finally, the model parameters are updated as:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{10}$$

where $\alpha$ is the learning rate, $\beta_1$ and $\beta_2$ are decay rates, and $\epsilon$ is a small constant to ensure numerical stability.

## 5 Implementation Details

This section outlines the methodologies for paper classification using titles. It presents details on text cleaning, GloVe embeddings for word representation, and the architectures of the CNN model.

### 5.1 Text Cleaning

Before embedding the text data into numerical representations, we pre-processed the selected text data to ensure consistency and improve model performance. The pre-processing pipeline involves several steps:

- We tokenize the text by splitting it into individual words or sub-words.
- We remove punctuation, non-alphanumerical characters, and excessive whitespace to standardize the input.
- We convert all alphabetic characters to lowercase to ensure uniformity and reduce variability in the data.
- We pad or truncate sequences to a fixed length to handle variable-length inputs.

### 5.2 GloVe Embeddings

To generate high-quality word representations, we employ pre-trained GloVe embeddings [2], which are derived from large-scale word co-occurrence matrices trained on a vocabulary of 400,000 words. After passing through the word embedding layer, each word is represented as a 50-dimensional vector. Then, processed text is converted into tensors for efficient training in deep-learning models.

### 5.3 Training, Validation, Test Dataset

To ensure a robust evaluation of our models, we split the dataset into three distinct subsets: training, validation, and test sets. Specifically, 50% of the data is allocated for training, 25% for validation, and 25% for testing. This partitioning allows us to train the models effectively, tune hyperparameters using the validation set, and evaluate their generalization performance on unseen data through the test set.

### 5.4 CNN Model Structure

The model architecture with different embedding sizes are summarized in Table 1, Table 2, Table 3, respectively.

Specifically, the architecture for the CNN model with the embedding size 50 is illustrated in detail: The CNN model architecture consists of three one-dimensional convolutional layers (Conv1d), each followed by a max-pooling layer

Table 1: Summary of Glove50CNN Model Architecture and Resources

| **Layer Type** | **Output Shape** | **Parameters** |
|---|---|---|
| Embedding | [512, 256, 50] | 20,000,000 |
| Conv1d (50 $\rightarrow$ 128, kernel=3) | [512, 128, 256] | 19,328 |
| MaxPool1d (kernel=2) | [512, 128, 128] | 0 |
| Conv1d (128 $\rightarrow$ 128, kernel=3) | [512, 128, 128] | 49,280 |
| MaxPool1d (kernel=2) | [512, 128, 64] | 0 |
| Conv1d (128 $\rightarrow$ 128, kernel=3) | [512, 128, 64] | 49,280 |
| MaxPool1d (kernel=2) | [512, 128, 32] | 0 |
| Fully Connected (Linear) | [512, 5] | 20,485 |
| **Total Parameters** | | **20,138,373** |
| **Trainable Parameters** | | **138,373** |
| **Non-trainable Parameters** | | **20,000,000** |
| **Resource** | | **Size (MB)** |
| Input Size for one batch | | 0.50 |
| Forward/Backward Pass | | 386.02 |
| Parameters Size | | 76.82 |
| **Estimated Total Size** | | **463.34** |

Table 2: Summary of Glove100CNN Model Architecture and Resources

| **Layer Type** | **Output Shape** | **Parameters** |
|---|---|---|
| Embedding | [512, 256, 100] | 40,000,000 |
| Conv1d (100 $\rightarrow$ 128, kernel=3) | [512, 128, 256] | 38,528 |
| MaxPool1d (kernel=2) | [512, 128, 128] | 0 |
| Conv1d (128 $\rightarrow$ 128, kernel=3) | [512, 128, 128] | 49,280 |
| MaxPool1d (kernel=2) | [512, 128, 64] | 0 |
| Conv1d (128 $\rightarrow$ 128, kernel=3) | [512, 128, 64] | 49,280 |
| MaxPool1d (kernel=2) | [512, 128, 32] | 0 |
| Fully Connected (Linear) | [512, 5] | 20,485 |
| **Total Parameters** | | **40,157,573** |
| **Trainable Parameters** | | **157,573** |
| **Non-trainable Parameters** | | **40,000,000** |
| **Resource** | | **Size (MB)** |
| Input Size for one batch | | 0.50 |
| Forward/Backward Pass | | 436.02 |
| Parameters Size | | 153.19 |
| **Estimated Total Size** | | **589.71** |

(MaxPool1d). The first convolutional layer takes an input of 50 channels from the embedding layer and applies 128 convolutional filters with a kernel size of 3, stride of 1, and padding set to 'same' to preserve sequence length. A max-pooling operation with a kernel size of 2 and stride of 2 reduces the temporal resolution by half. The second and third convolutional layers maintain 128 channels and use the same kernel size, stride, and padding configuration. Each is followed by a max-pooling layer with identical parameters, progressively reducing the sequence length. After the final convolutional block, the feature map is flattened into a 4096-dimensional vector and passed to a fully connected layer (Linear) with five output units, corresponding to the classification task.

### 5.5 Training Parameters

In our experiments, we set the hyper-parameters in the training process as shown in Table 4.

Table 3: Glove200CNN Model Summary

| **Layer Type** | **Output Shape** | **Parameters** |
|---|---|---|
| Embedding | [512, 256, 200] | 80,000,000 |
| Conv1d (200 $\rightarrow$ 128, kernel=3) | [512, 128, 256] | 76,928 |
| MaxPool1d (kernel=2) | [512, 128, 128] | 0 |
| Conv1d (128 $\rightarrow$ 128, kernel=3) | [512, 128, 128] | 49,280 |
| MaxPool1d (kernel=2) | [512, 128, 64] | 0 |
| Conv1d (128 $\rightarrow$ 128, kernel=3) | [512, 128, 64] | 49,280 |
| MaxPool1d (kernel=2) | [512, 128, 32] | 0 |
| Fully Connected (Linear) | [512, 5] | 20,485 |
| **Total Parameters** | | **80,195,973** |
| **Trainable Parameters** | | **195,973** |
| **Non-trainable Parameters** | | **80,000,000** |
| **Resource** | | **Size (MB)** |
| Input Size for one batch | | 0.50 |
| Forward/Backward Pass | | 536.02 |
| Parameters Size | | 305.92 |
| **Estimated Total Size** | | **842.44** |

Table 4: Hyperparameters in Model Training

| **Parameter** | **Value** |
|---|---|
| Learning rate ($\alpha$) | 0.001 |
| Beta parameters | |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.999 |
| Mini-batch size | 512 |
| Number of epochs | 20 |

## 6 Results

In this section, we evaluate the model's performance using multiple metrics, including the training curve of loss and accuracy, classification performances, and confusion matrices. The model is tested on data from 2015-2020 to assess in-range performance and also on the 2023 dataset to evaluate its generalization ability. The embedding sizes we explored include 50, 100, and 200.

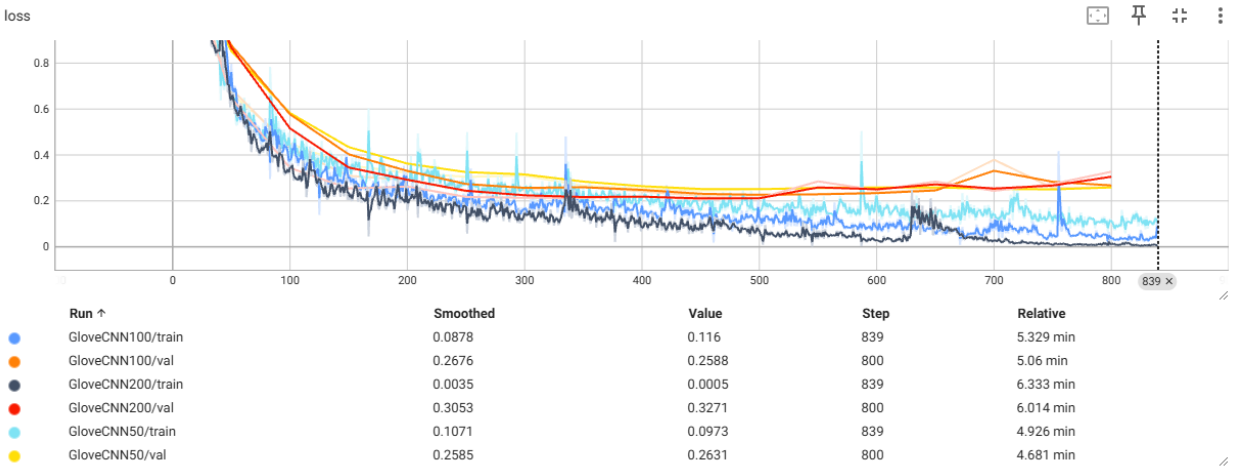
### 6.1 Training Curves

The training curve tracks learning progress, as shown in Figure 4. Figure 4a indicates that models with larger embedding sizes (100, 200) achieve lower training and validation loss compared to the 50-dimensional embedding, suggesting improved representation learning. Training loss decreases from over 0.8 to approximately 0.15, while validation loss stabilizes around 0.20, indicating effective learning and good generalization. In contrast, the 50-dimensional embedding model exhibits higher loss, indicating potential underfitting.

Similarly, Figure 4b shows a steady increase in accuracy, with the 200-dimensional embedding achieving the highest training accuracy, nearing 99%, while the 50-dimensional model stabilizes at a lower accuracy level. Validation accuracy follows a similar trend, ranging from 85% to 92% depending on embedding size. Occasional spikes in loss and accuracy suggest minor instabilities, likely due to the absence of batch normalization, leading to internal covariate shifts.

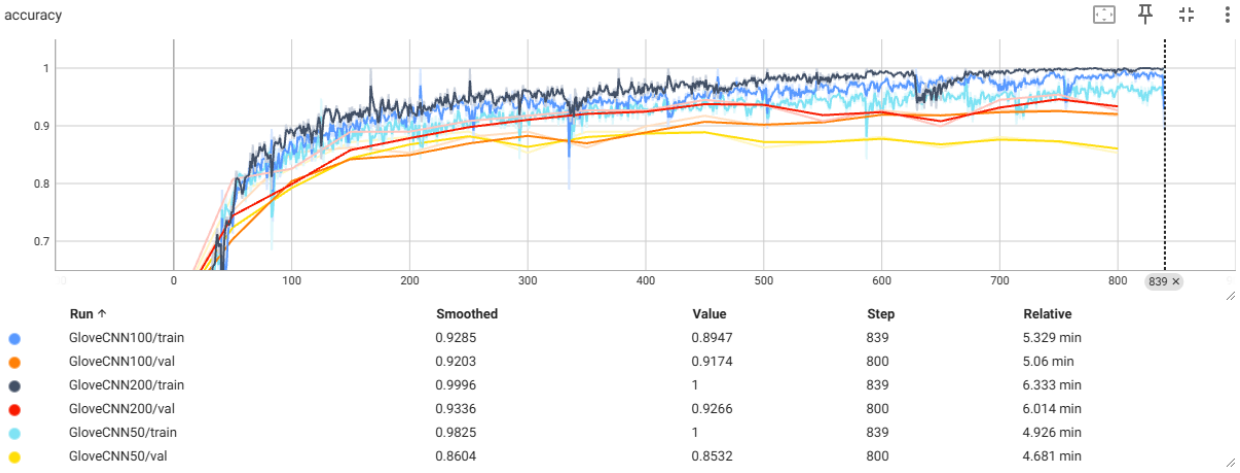
### 6.2 Performance of Glove50CNN

The precision, recall, and F1-score for the 2015–2020 and 2023 test sets are presented in Table 5 and Table 6, respectively. On the 2015–2020 test set, the model achieved a test loss of 0.2704 and an accuracy of 92%, demonstrating strong




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| GloveCNN100/train | 0.0878 | 0.116 | 839 | 5.329 min |
| GloveCNN100/val | 0.2676 | 0.2588 | 800 | 5.06 min |
| GloveCNN200/train | 0.0035 | 0.0005 | 839 | 6.333 min |
| GloveCNN200/val | 0.3053 | 0.3271 | 800 | 6.014 min |
| GloveCNN50/train | 0.1071 | 0.0973 | 839 | 4.926 min |
| GloveCNN50/val | 0.2585 | 0.2631 | 800 | 4.681 min |

(a) Training and validation loss curves for GloveCNN models with embedding dimensions 50, 100 and 200.




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| GloveCNN100/train | 0.9285 | 0.8947 | 839 | 5.329 min |
| GloveCNN100/val | 0.9203 | 0.9174 | 800 | 5.06 min |
| GloveCNN200/train | 0.9996 | 1 | 839 | 6.333 min |
| GloveCNN200/val | 0.9336 | 0.9266 | 800 | 6.014 min |
| GloveCNN50/train | 0.9825 | 1 | 839 | 4.926 min |
| GloveCNN50/val | 0.8604 | 0.8532 | 800 | 4.681 min |

(b) Training and validation accuracy curves for GloveCNN models with embedding dimensions 50, 100 and 200.

Figure 4: CNN Model Performance: Training and validation loss and accuracy curves for GloveCNN models with embedding dimensions 50, 100, and 200.

performance across four categories: cs.CL (F1: 0.94), cs.CR (F1: 0.91), cs.CV (F1: 0.94), and cs.RO (F1: 0.87). However, cs.LG (F1: 0.63) exhibited lower performance, likely due to overlapping topics with other categories, as evidenced by misclassifications in the confusion matrix. On the 2023 test set, accuracy decreased to 84%, with cs.CL (F1: 0.87), cs.CR (F1: 0.85), cs.CV (F1: 0.88), and cs.RO (F1: 0.86) maintains robust results. In contrast, cs.LG (F1: 0.63) experienced a notable decline in recall (0.49), potentially attributable to evolving trends and terminology in machine learning research. These findings highlight the model's effectiveness while underscoring the importance of periodic updates to address emerging research dynamics.

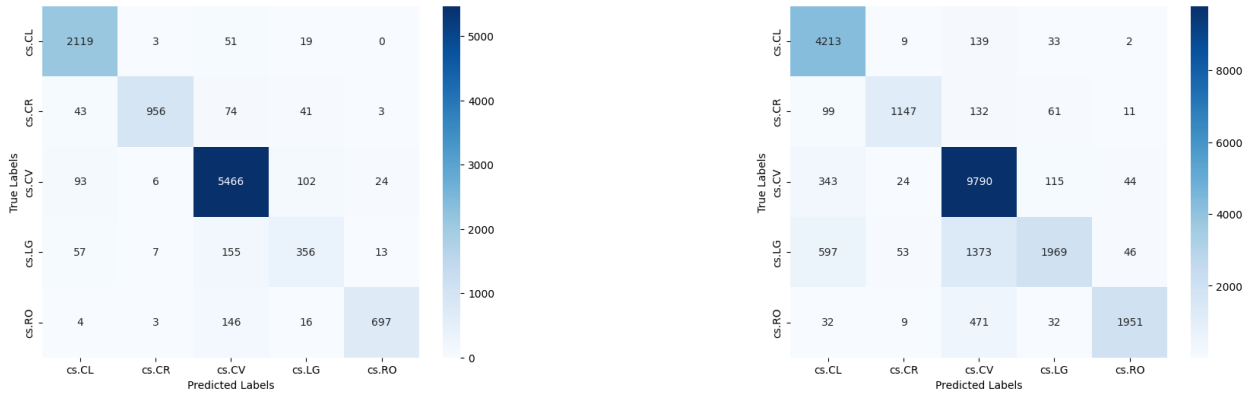
The confusion matrices for different year ranges, presented in Figure 5, highlight the model's misclassifications and performance variations over time. Figure 5a (2015–2020) demonstrates strong performance for most categories, with diagonal elements indicating high correct classification rates. However, off-diagonal elements reveal challenges in distinguishing cs.LG from cs.CV and cs.CL, likely due to overlapping topics. Figure 5b (2023) shows the model's performance on newer, unseen data, where misclassifications increase, particularly for cs.LG, reflecting the impact of evolving research trends and terminology. This comparison underscores the model's robustness on historical data while highlighting the need for adaptations to address emerging trends and improve generalization to newer datasets.

Table 5: Performance of Glove50CNN (2015-2020)

| Class | Precision | Recall | F1-score | Count |
|---|---|---|---|---|
| cs.CL | 0.91 | 0.97 | 0.94 | 4396 |
| cs.CR | 0.98 | 0.86 | 0.91 | 1450 |
| cs.CV | 0.93 | 0.96 | 0.94 | 10316 |
| cs.LG | 0.67 | 0.61 | 0.63 | 4038 |
| cs.RO | 0.95 | 0.80 | 0.87 | 2495 |
| Accuracy | | | 0.92 | |
| M Avg | 0.89 | 0.84 | 0.86 | |
| W Avg | 0.92 | 0.92 | 0.92 | |

Table 6: Performance of Glove50CNN (2023)

| Class | Precision | Recall | F1-score | Count |
|---|---|---|---|---|
| cs.CL | 0.80 | 0.96 | 0.87 | 4396 |
| cs.CR | 0.92 | 0.79 | 0.85 | 1450 |
| cs.CV | 0.82 | 0.95 | 0.88 | 10316 |
| cs.LG | 0.89 | 0.49 | 0.63 | 4038 |
| cs.RO | 0.95 | 0.78 | 0.86 | 2495 |
| Accuracy | | | 0.84 | |
| M Avg | 0.88 | 0.79 | 0.82 | |
| W Avg | 0.85 | 0.84 | 0.83 | |




(a) Confusion Matrix (2015-2020)

(b) Confusion Matrix (2023)

Figure 5: Comparison of Confusion Matrices for 2015-2020 and 2023

### 6.3 Performance of Glove100CNN

Table 7 and Table 8, respectively. On the 2015–2020 test set, the model achieved a test loss of 0.5783 and an accuracy of 90%, demonstrating strong classification performance across most categories: cs.CL (F1: 0.93), cs.CR (F1: 0.94), cs.CV (F1: 0.92), and cs.RO (F1: 0.87). However, cs.LG (F1: 0.17) showed significantly lower performance, likely due to its limited representation and misclassifications with other research categories, as observed in the confusion matrix.

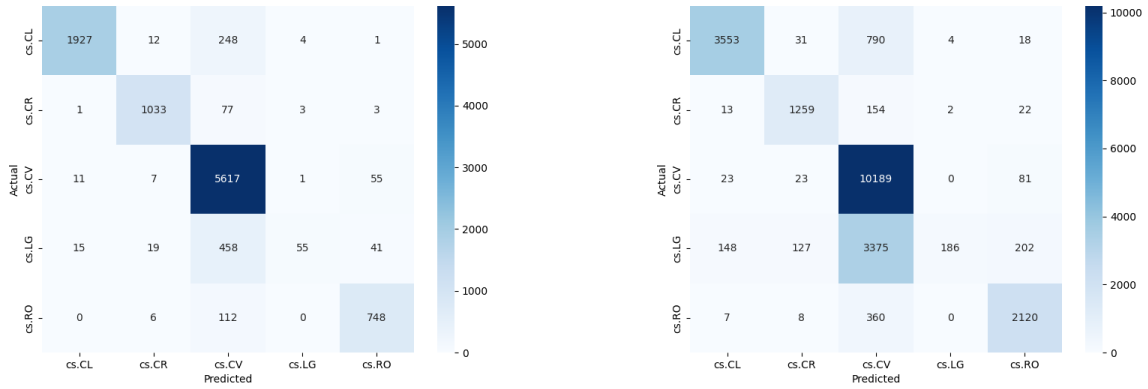
On the 2023 test set, accuracy declined to 76%, with cs.CL (F1: 0.87), cs.CR (F1: 0.87), cs.CV (F1: 0.81), and cs.RO (F1: 0.86) maintains reasonable classification performance. However, cs.LG (F1: 0.09) exhibited a drastic drop in both recall (0.05) and precision, indicating major difficulties in recognizing this category in newer data. This decline suggests that GloveCNN100 struggles with generalization, particularly for underrepresented or evolving research topics, emphasizing the importance of improving model robustness through additional training data or fine-tuning on more recent datasets.

Table 7: Performance of Glove100CNN (2015-2020)

| Class | Precision | Recall | F1-score | Count |
|---|---|---|---|---|
| cs.CL | 0.99 | 0.88 | 0.93 | 2192 |
| cs.CR | 0.96 | 0.92 | 0.94 | 1117 |
| cs.CV | 0.86 | 0.99 | 0.92 | 5691 |
| cs.LG | 0.87 | 0.09 | 0.17 | 588 |
| cs.RO | 0.88 | 0.86 | 0.87 | 866 |
| Accuracy | | | 0.90 | |
| M Avg | 0.91 | 0.75 | 0.77 | |
| W Avg | 0.90 | 0.90 | 0.88 | |

Table 8: Performance of Glove100CNN (2023)

| Class | Precision | Recall | F1-score | Count |
|---|---|---|---|---|
| cs.CL | 0.95 | 0.81 | 0.87 | 4396 |
| cs.CR | 0.87 | 0.87 | 0.87 | 1450 |
| cs.CV | 0.69 | 0.99 | 0.81 | 10316 |
| cs.LG | 0.97 | 0.05 | 0.09 | 4038 |
| cs.RO | 0.87 | 0.85 | 0.86 | 2495 |
| Accuracy | | | 0.76 | |
| M Avg | 0.87 | 0.71 | 0.70 | |
| W Avg | 0.82 | 0.76 | 0.70 | |




(a) Confusion Matrix (2015-2020)

(b) Confusion Matrix (2023)

Figure 6: Comparison of Confusion Matrices for 2015-2020 and 2023

### 6.4 Performance of Glove200CNN

The precision, recall, and F1-score for the 2015–2020 and 2023 test sets are presented in Table 9 and Table 10, respectively. On the 2015–2020 test set, the model achieved a test loss of 0.3107 and an accuracy of 94%, demonstrating strong classification performance across most categories: cs.CL (F1: 0.96), cs.CR (F1: 0.96), cs.CV (F1: 0.96), and cs.RO (F1: 0.90). The cs.LG (F1: 0.70) class exhibited lower performance, likely due to topic overlap with other categories, as observed in the confusion matrix.

On the 2023 test set, accuracy declined to 87%, with cs.CL (F1: 0.91), cs.CR (F1: 0.88), cs.CV (F1: 0.90), and cs.RO (F1: 0.88) maintains strong classification performance. However, cs.LG (F1: 0.71) showed a notable drop in recall (0.59), suggesting increased misclassifications, possibly due to evolving research trends and terminology. These results indicate that while GloveCNN200 retains high performance, particularly in major categories, it faces challenges in adapting to emerging topics, underscoring the need for periodic retraining to maintain generalization in evolving datasets.

Table 9: Performance of Glove200CNN (2015-2020)

| Class | Precision | Recall | F1-score | Count |
|---|---|---|---|---|
| cs.CL | 0.97 | 0.95 | 0.96 | 2192 |
| cs.CR | 0.96 | 0.95 | 0.96 | 1117 |
| cs.CV | 0.95 | 0.96 | 0.96 | 5691 |
| cs.LG | 0.71 | 0.68 | 0.70 | 588 |
| cs.RO | 0.91 | 0.89 | 0.90 | 866 |
| Accuracy | | | 0.94 | |
| M Avg | 0.90 | 0.89 | 0.90 | |
| W Avg | 0.94 | 0.94 | 0.94 | |

Table 10: Performance of Glove200CNN (2023)

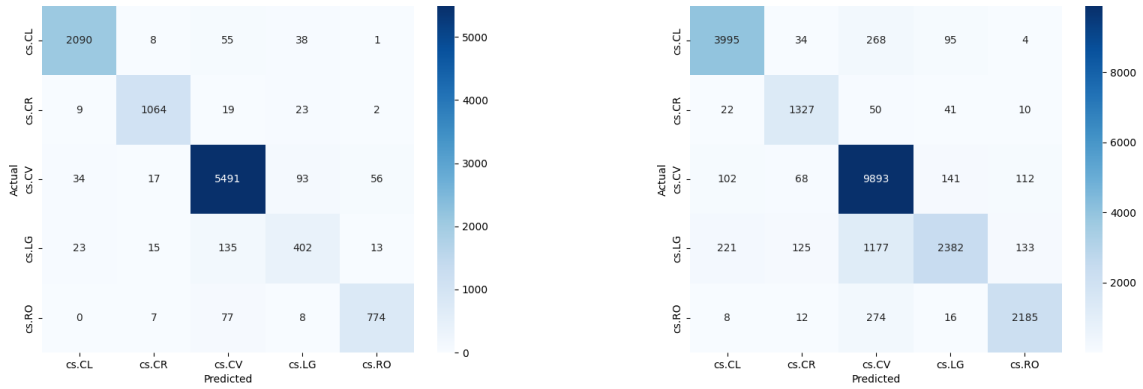
| Class | Precision | Recall | F1-score | Count |
|---|---|---|---|---|
| cs.CL | 0.92 | 0.91 | 0.91 | 4396 |
| cs.CR | 0.85 | 0.92 | 0.88 | 1450 |
| cs.CV | 0.85 | 0.96 | 0.90 | 10316 |
| cs.LG | 0.89 | 0.59 | 0.71 | 4038 |
| cs.RO | 0.89 | 0.88 | 0.88 | 2495 |
| Accuracy | | | 0.87 | |
| M Avg | 0.88 | 0.85 | 0.86 | |
| W Avg | 0.87 | 0.87 | 0.87 | |

## 7 Discussions

### 7.1 Impact of Embedding Dimension

The results indicate that increasing the embedding dimension generally improves model performance across accuracy, F1-score, and generalization ability.

Glove200CNN achieved the highest accuracy (94%) on the 2015-2020 test set, outperforming Glove100CNN (90%) and Glove50CNN (92%), suggesting that larger embeddings enable the model to capture more nuanced representations. This trend is further observed in the macro and weighted F1-scores, where Glove200CNN consistently performed better




(a) Confusion Matrix (2015-2020)

(b) Confusion Matrix (2023)

Figure 7: Comparison of Confusion Matrices for 2015-2020 and 2023

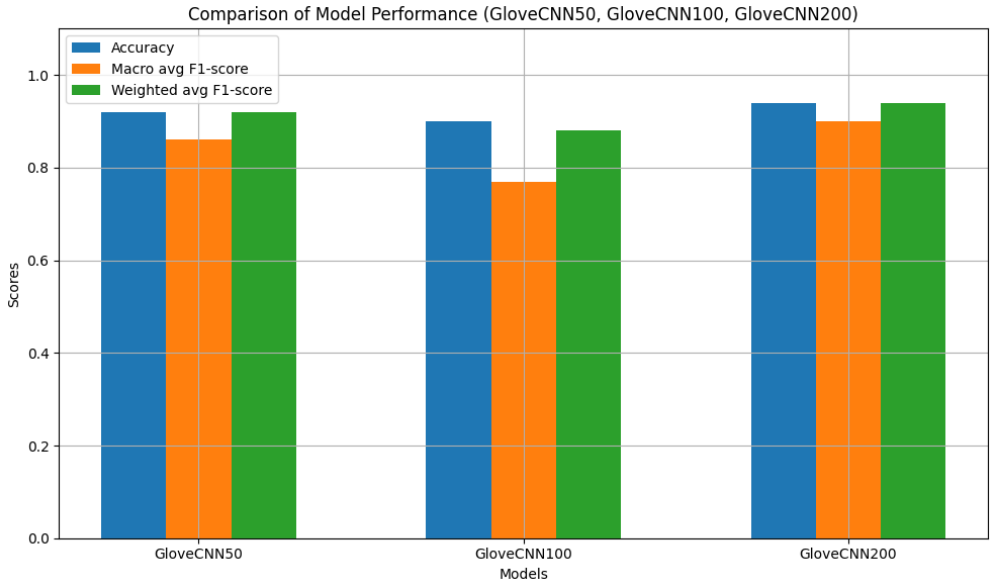



Figure 8: Performance Comparison for CNN models with different Embedding Dimensions of 50, 100, and 200

across most categories, particularly in cs.LG, which showed an F1-score of 0.70 compared to 0.17 in Glove100CNN and 0.63 in Glove50CNN.

However, while Glove200CNN maintained strong results, the performance gain from 100 to 200 dimensions was marginal, indicating potential diminishing returns in increasing embedding size. Larger embeddings also come with higher computational costs and memory requirements, as seen in the estimated total model sizes (463.34 MB for Glove50CNN, 589.71 MB for Glove100CNN, and 842.44 MB for Glove200CNN). This suggests that Glove100CNN provides a good balance between performance and efficiency, making it a more practical choice when computational resources are constrained.

### 7.2 Impact of Out-of-distribution Test Data

When tested on out-of-distribution data (2023 test set), all models experienced a performance drop, highlighting challenges in generalization. The accuracy of Glove50CNN dropped from 90% to 76%, while Glove100CNN fell from 90% to 76%, and Glove200CNN decreased from 94% to 87%. This decline suggests that the models struggle to adapt to distribution shifts, likely due to evolving research trends and terminology changes in newer datasets.

Category-wise analysis reveals that cs.LG suffered the most, with its F1-score dropping drastically, particularly in Glove100CNN (from 0.17 to 0.09) and Glove50CNN (from 0.63 to 0.09). This indicates that underrepresented or evolving categories are more vulnerable to performance degradation, reinforcing the need for periodic retraining on newer datasets. Glove200CNN, despite experiencing a drop, exhibited the most stable performance across classes, demonstrating that larger embeddings help models retain generalization better. However, other categories, such as cs.CL, cs.CR, and cs.CV, demonstrated strong generalization, particularly in Glove200CNN, where cs.CL (F1: 0.91), cs.CR (F1: 0.88), and cs.CV (F1: 0.90) maintained high performance despite the dataset shift. Glove200CNN exhibited the most stable performance across classes. While Glove100CNN and Glove50CNN showed a more noticeable performance drop, they still maintained relatively consistent results in cs.CL and cs.CR, with F1 scores remaining above 0.85.

## 8 Conclusion

In this study, we evaluated the performance of GloVe-CNN models with varying embedding dimensions (50, 100, and 200) for research paper classification. Our findings reveal a trade-off between embedding size, classification performance, and computational efficiency. While larger embeddings generally improved accuracy and F1-scores, GloVe200CNN achieved the best performance, particularly in macro-average and weighted-average F1 scores. However, the marginal performance gain from 100 to 200 dimensions suggests diminishing returns as embedding size increases.

When tested on out-of-distribution data from 2023, all models exhibited performance degradation, with cs.LG experiencing the most significant decline due to evolving research trends and terminology shifts. GloVe200CNN demonstrated the most stable generalization, while GloVe50CNN and GloVe100CNN struggled more, particularly in underrepresented categories. In contrast, well-represented categories such as cs.CL, cs.CR, and cs.CV remained relatively robust, highlighting the critical role of class distribution in maintaining model performance over time.

Several avenues for future work could further enhance the system's performance and granularity. First, training custom domain-specific word embeddings could improve semantic representation by capturing domain-specific nuances. Second, exploring hybrid architectures that combine CNNs and RNNs could leverage the strengths of both approaches, such as the local feature extraction capabilities of CNNs and the sequential dependency modeling of RNNs. Finally, integrating large language models (LLMs) into the pipeline could significantly enhance classification granularity and accuracy by leveraging their advanced contextual understanding and generalization capabilities. These improvements could pave the way for more robust and scalable solutions in academic paper classification.

## References


[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

[2] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

[5] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 137–142, 1998.

[6] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[7] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[8] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard 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*, pages 1480–1489, 2016.

[9] Cornell University. arxiv metadata dataset. Kaggle, 2021.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.