# Deep learning for NLP Mid Term Project

# 1. Problem Definition

We have taken Comparative Study of CNN and RNN for Natural Language Processing as base paper which systematically evaluates CNNs, GRUs, and LSTMs on NLP tasks, including **SemEval-2010 Task 8: Relation Classification**. They got best result for this task was **68.56% F1-score** using a GRU model.

**Problem Statement**

The goal is to **improve the F1-score on SemEval-2010 Task 8 beyond the base paper's 68.56%**

# 1. Dataset - SemEval-2010 Task 8

This is a benchmark dataset for relation classification, which focusses on identifying semantic relationships between pairs of different nominals (nouns/noun phrases) which are present in sentences. Below is a detailed breakdown of the dataset we chose for this:

---

**1. Dataset Overview**

- **Key Features**:
  - **Directionality**: Relations are directional (e.g., Cause-Effect(e1,e2) ≠ Cause-Effect(e2,e1)).
  - **Fine-grained Labels**: The dataset includes relations like *Entity-Origin*, *Content-Container*, and *Instrument-Agency* which are nuanced.
  - **Adversarial Examples**: Many sentences labeled as *"Other"* contain co-occurring entities without a valid relation.

---

**2. Dataset Statistics**

| Split | Sentences | Relation Types |
|---|---|---|
| **Training** | 8,000 | 19 (9 relations + inverses + "Other") |
| **Test** | 2,717 | Same as training |

- **Relation Types**:
  1. **Cause-Effect** (e.g., *"Smoking <e1>causes</e1> <e2>cancer</e2>"*)
  2. **Instrument-Agency** (e.g., *"<e1>Knife</e1> cuts <e2>bread</e2>"*)
  3. **Product-Producer** (e.g., *"<e1>Microsoft</e1> developed <e2>Windows</e2>"*)
  4. **Entity-Origin** (e.g., *"<e1>Wine</e1> from <e2>France</e2>"*)
  5. **Entity-Destination** (e.g., *"<e1>Passengers</e1> boarded the <e2>plane</e2>"*)
  6. **Component-Whole** (e.g., *"<e1>Wheel</e1> of a <e2>car</e2>"*)
  7. **Member-Collection** (e.g., *"<e1>Player</e1> on the <e2>team</e2>"*)

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

8. **Message-Topic** (e.g., *"<e1>Article</e1> about <e2>climate</e2>"*)
9. **Content-Container** (e.g., *"<e1>Water</e1> in the <e2>bottle</e2>"*)
10. **"Other"**: No valid relation.

---

**3. Data Preprocessing**

- **Text Cleaning:**
  We standardized the input by lowercasing and retaining only letters, numbers, and specific symbols.
- **Tokenization & Padding:**
  We used Keras Tokenizer converts sentences into sequences of integers (with a defined maximum vocabulary size) and pads these sequences to a fixed maximum length.
- **Label Encoding:**
  We encoded labels using scikit-learn's LabelEncoder and then transformed into one-hot encoding vectors.

# 4. Embedding Strategies

We have performed multiple experiments with different types of embedding:

- **GloVe Embeddings:**
  Some experiments loads pre-trained GloVe embeddings from a file (e.g., glove.6B.100d.txt). An embedding matrix is built where each word in the vocabulary is mapped to its corresponding vector.
- **Word2Vec & FastText:**
  We load Word2Vec (using gensim's KeyedVectors) and FastText embeddings for out-of-vocabulary (OOV) words we initializes random vectors.
- **Random Embeddings**:
  Some experiments use embeddings that are randomly initialized and learned during training, without using any pre-trained weights.

# 3. Model Architectures

We implemented several architectures, each designed to compare different strategies. Below, we list all the models used in our experiments.

## List of Models

1. **CNN+Bi-GRU**
    - A hybrid model combining **CNN** (Convolutional Neural Network) and **Bidirectional GRU** (Gated Recurrent Unit).

| Layer (Type) | Output Shape | Param # | Description |
|---|---|---|---|
| **Input Layer (InputLayer)** | (None, 100) | 0 | Accepts integer-encoded sequences of fixed length 100. |
| **Embedding (Embedding)** | (None, 100, 100) | 2,762,000 | Maps each word index to a 100-dimensional dense vector; vocabulary size inferred from the training data. |
| **Spatial Dropout (SpatialDropout1D)** | (None, 100, 100) | 0 | Drops entire 1D feature maps to reduce overfitting, operating on the embedding outputs. |
| **Bidirectional GRU (Bidirectional(GRU))** | (None, 100, 256) | 176,640 | Processes the sequence bidirectionally with 128 units per direction, outputting 256 features per time step. |
| **Conv1D (Kernel Size 3)** | (None, 98, 256) | 77,056 | Convolves over the embeddings with a kernel size of 3 and 256 filters; output length = 100 - 3 + 1 = 98. |
| **Conv1D (Kernel Size 4)** | (None, 97, 256) | 102,656 | Convolves over the embeddings with a kernel size of 4 and 256 filters; output length = 100 - 4 + 1 = 97. |
| **Conv1D (Kernel Size 5)** | (None, 96, 256) | 128,256 | Convolves over the embeddings with a kernel size of 5 and 256 filters; output length = 100 - 5 + 1 = 96. |
| **Bidirectional GRU (Second GRU)** | (None, 100, 256) | 296,448 | A second bidirectional GRU layer (128 units per direction) processing the output of the first GRU branch. |
| **GlobalMaxPooling1D (on Conv1D, kernel size 3)** | (None, 256) | 0 | Applies max pooling over time on the Conv1D (kernel 3) output. |
| **GlobalMaxPooling1D (on Conv1D, kernel size 4)** | (None, 256) | 0 | Applies max pooling over time on the Conv1D (kernel 4) output. |
| **GlobalMaxPooling1D (on Conv1D, kernel size 5)** | (None, 256) | 0 | Applies max pooling over time on the Conv1D (kernel 5) output. |
| **GlobalMaxPooling1D (on 2nd Bidirectional GRU)** | (None, 256) | 0 | Pools the output of the second bidirectional GRU layer across all time steps. |
| **Attention Layer** | (None, 256) | 356 | Computes a weighted sum over the GRU outputs using a trainable attention mechanism. |
| **Concatenate (CNN branch)** | (None, 768) | 0 | Concatenates the three global max pooled outputs from the three Conv1D layers (256×3 = 768 features). |
| **Concatenate (GRU branch)** | (None, 512) | 0 | Concatenates the pooled GRU output (256) with the attention output (256) to form a 512-dimensional feature vector. |
| **Concatenate (Final)** | (None, 1280) | 0 | Merges the CNN branch (768 features) and the GRU branch (512 features) to produce 1280 total features. |
| **Batch Normalization** | (None, 1280) | 5,120 | Normalizes the combined feature vector to stabilize training and speed up convergence. |
| **Dense** | (None, 128) | 163,968 | A fully connected layer that reduces the dimensionality from 1280 to 128 using ReLU activation. |
| **Dropout** | (None, 128) | 0 | Randomly drops 50% of the neurons in the dense layer to help prevent overfitting. |
| **Output Dense (Softmax)** | (None, 10) | 1,290 | Final classification layer that produces probability distributions over 10 classes using softmax activation. |

2. **Bidirectional GRU**
    - A standalone **Bidirectional GRU** model.

| Layer (Type) | Output Shape | Param # | Description |
|---|---|---|---|
| **input_39 (InputLayer)** | (None, 100) | 0 | Input sequence of length 100. |
| **embedding_38 (Embedding)** | (None, 100, 100) | 2,762,000 | Maps indices to 100D vectors. |
| **spatial_dropout1d_41 (SpatialDropout1D)** | (None, 100, 100) | 0 | Applies dropout to entire embedding maps. |
| **bidirectional_55 (Bidirectional GRU)** | (None, 100, 256) | 176,640 | Processes sequences in both directions. |
| **global_max_pooling1d_85 (GlobalMaxPooling1D)** | (None, 256) | 0 | Pools maximum value over time steps. |
| **dense_82 (Dense)** | (None, 128) | 32,896 | Fully connected layer, reduces features to 128. |
| **dropout_37 (Dropout)** | (None, 128) | 0 | Regularizes by dropping units. |
| **dense_83 (Dense)** | (None, 10) | 1,290 | Output layer with 10 units (e.g., softmax classifier). |

3. **CNN**
    - A standalone **Convolutional Neural Network** model.

| Layer (Type) | Output Shape | Param # | Connected To | Description |
|---|---|---|---|---|
| **Input (InputLayer)** | (None, 100) | 0 | – | Input sequences. |
| **Embedding (Embedding)** | (None, 100, 100) | 2,762,000 | Input | Maps indices to 100D vectors. |
| **SpatialDropout (SpatialDropout1D)** | (None, 100, 100) | 0 | Embedding | Drops entire feature maps. |
| **Conv1D (kernel=3)** | (None, 98, 256) | 77,056 | SpatialDropout | 1D conv with kernel size 3. |
| **Conv1D (kernel=4)** | (None, 97, 256) | 102,656 | SpatialDropout | 1D conv with kernel size 4. |
| **Conv1D (kernel=5)** | (None, 96, 256) | 128,256 | SpatialDropout | 1D conv with kernel size 5. |
| **GlobalMaxPool (from conv, k=3)** | (None, 256) | 0 | Conv1D (kernel=3) | Max-pooling over time. |
| **GlobalMaxPool (from conv, k=4)** | (None, 256) | 0 | Conv1D (kernel=4) | Max-pooling over time. |
| **GlobalMaxPool (from conv, k=5)** | (None, 256) | 0 | Conv1D (kernel=5) | Max-pooling over time. |
| **Concatenate** | (None, 768) | 0 | All GlobalMaxPools | Merges three pooled outputs (256×3). |
| **BatchNorm (BatchNormalization)** | (None, 768) | 3,072 | Concatenate | Normalizes features. |
| **Dense** | (None, 128) | 98,432 | BatchNorm | Fully connected, reduces dim to 128. |
| **Dropout** | (None, 128) | 0 | Dense | Regularizes by dropping units. |
| **Output Dense (Softmax)** | (None, 10) | 1,290 | Dropout | Final classification layer (10 classes). |

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

4. **Bidirectional GRU**
    - A standalone **Gated Recurrent Unit** model. We added a batch normalization layer because it will normalize the gru output which will stabalize parameters in training and help us converge faster. We removed Global max pooling which reduces dimensionality by selecting dominant features.

| Layer (Type) | Output Shape | Param # | Description |
|---|---|---|---|
| **input_42 (InputLayer)** | (None, 100) | 0 | Input sequence of length 100. |
| **embedding_41 (Embedding)** | (None, 100, 100) | 2,762,000 | Maps word indices to 100D vectors. |
| **spatial_dropout1d_44 (SpatialDropout1D)** | (None, 100, 100) | 0 | Drops entire embedding maps for regularization. |
| **bidirectional_58 (Bidirectional GRU)** | (None, 256) | 176,640 | Processes sequence bidirectionally; outputs 256 features. |
| **batch_normalization_31 (BatchNorm)** | (None, 256) | 1,024 | Normalizes the GRU output. |
| **dense_88 (Dense)** | (None, 128) | 32,896 | Fully connected layer reducing features to 128. |
| **dropout_40 (Dropout)** | (None, 128) | 0 | Applies dropout for regularization. |
| **dense_89 (Dense)** | (None, 10) | 1,290 | Output layer (10 classes, softmax activation). |

5. **LSTM**
    - A standalone **Long Short-Term Memory** model.

| Layer (Type) | Output Shape | Param # | Description |
|---|---|---|---|
| **input_43 (InputLayer)** | (None, 100) | 0 | Input sequence of length 100. |
| **embedding_42 (Embedding)** | (None, 100, 100) | 2,762,000 | Maps indices to 100D embeddings. |
| **spatial_dropout1d_45 (SpatialDropout1D)** | (None, 100, 100) | 0 | Drops entire feature maps for regularization. |
| **bidirectional_59 (Bidirectional GRU)** | (None, 256) | 234,496 | Processes sequence bidirectionally; outputs 256 features. |
| **batch_normalization_32 (BatchNorm)** | (None, 256) | 1,024 | Normalizes GRU outputs. |
| **dense_90 (Dense)** | (None, 128) | 32,896 | Fully connected layer reducing features to 128. |
| **dropout_41 (Dropout)** | (None, 128) | 0 | Applies dropout for further regularization. |
| **dense_91 (Dense)** | (None, 10) | 1,290 | Output layer with 10 units (e.g., for classification). |

# 3. Training and Optimization

**Loss and Optimizer:**

- **Loss Function:**
  We used **categorical crossentropy** loss, which is well-suited for multi-class classification problems.
- **Optimizer:**
  We used **Adam optimizer** for its adaptive learning rate and efficiency in training deep networks.

**Training Hyperparameters:**

- **Hyperparameters:**
  - **MAX_VOCAB:** 20,000
    Limits the vocabulary size used in tokenization.
  - **MAX_LENGTH:** 100
    Sets the fixed length for input sequences via padding/truncation.
  - **EMBEDDING_DIM:** 100
    Defines the dimensionality of the GloVe word embeddings.
- **Batch Size & Epochs:**
  The model is trained using a batch size of **32** over **20 epochs**.
- **Validation:**
  The dataset is split into training and validation sets (approximately 80/20) to monitor performance and tune hyperparameters during training.

**Regularization and Callbacks:**

- **Early Stopping:**
  An early stopping callback monitors the validation loss and stops training if no improvement is observed for **3 consecutive epochs**, restoring the best model weights.
- **Learning Rate Reduction:**
  A ReduceLROnPlateau callback reduces the learning rate by a factor of **0.5** if the validation loss plateaus for **2 epochs**, with a minimum learning rate set to **1e-6**.
- **Class Weighting:**
  Class weights are computed using scikit-learn's compute_class_weight function to handle class imbalance, ensuring that the model does not bias toward majority classes.

# 4. Results

Highest Accuracy: The CNN + stacked Bidirectional GRU + Attention with GloVe embedding model achieved 75.30% test accuracy.

Models that used pre-trained embeddings consistently outperformed those that used random embeddings.

We tested four different embeddings. Three were pre-trained (GloVe, Word2Vec, FastText) and one was random. Different architectures were used, including CNN, RNN (GRU and LSTM), and a hybrid CNN+RNN model. All models were tested on the SemEval 2010 Task 8 dataset. The final results we observed are as follows:

1. CNN

- GloVe: 74.05%
- Random: 69.89%
- Frozen pretrained GloVe: 73.02%

2. GRU

- GloVe: 74.46%
- Random: 50.31%

3. LSTM

- GloVe: 69.67%
- Random: 53.48%

4. Hybrid Model: CNN + BiGRU + Attention

- GloVe: 75.30% (highest single-run figure recorded)
- Word2Ve: 70.78%
- FastText: 72.14%

5. Ensemble: CNN+LSTM+GRU

- 74%

Fine-Tuning vs. Freezing Embeddings

We also tested the effects of freezing vs. fine-tuning embeddings:

- Fine-tuned GloVe with CNN sometimes improved from 73.02% to 74.00% compared to freeing.

Trends in the Results

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

- Large Performance Gap Between Pre-trained and Random Embeddings: CNN and RNN models with random embeddings performed between 55-60%, while models with pre-trained embeddings scored between 70-75%. This 10-15% difference shows how much better pre-trained embeddings performed.
- GloVe Outperformed Other Pre-trained Embeddings: When comparing embeddings, GloVe was better by 1-2% over Word2Vec and FastText.
- Hybrid Model Achieved the Best Accuracy: The CNN + BiGRU + Attention approach consistently gave the highest accuracy (75.30% with GloVe).

Comparison with the Reference Paper

The reference paper found that no single architecture was definitively better. They all had its own merits and limitations:

- CNN performed well on short sentences since it captures local patterns efficiently.
- RNN models (GRU, LSTM) performed better on longer, more complex sentences because they capture context over a sequence.
- The variation between CNN vs. RNN depends heavily on sentence structure. This was observed in our results.

| Model | Embedding | Accuracy |
|---|---|---|
| **CNN** | GloVe | 74.05% |
| **CNN** | Random | 69.89% |
| **CNN** | Frozen Pretrained GloVe | 73.02% |
| **GRU** | GloVe | 74.46% |
| **GRU** | Random | 50.31% |
| **LSTM** | GloVe | 69.67% |
| **LSTM** | Random | 53.48% |
| **Hybrid (CNN + BiGRU + Attn)** | GloVe | 75.30% |
| **Hybrid (CNN + BiGRU + Attn)** | Word2Vec | 70.78% |
| **Hybrid (CNN + BiGRU + Attn)** | FastText | 72.14% |
| **Ensemble (CNN + LSTM + GRU)** | Mixed | 74.00% |

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

# 5. Experiments

## Experiment 1: Comparing Pre-trained vs. Random Embeddings

We ran our model using CNN, GRU, and LSTM architectures on the SemEval 2010 Task 8 dataset while using four different types of word embeddings:

- **GloVe** (a pre-trained embedding)
- **Word2Vec** (a pre-trained embedding)
- **FastText** (a pre-trained embedding)
- **Random** (embeddings initialized randomly, without pre-training)

The goal of the experiment is to understand if pretrained word embeddings are actually important and if they will outperform the random embedding and by how much. Pre-trained embeddings are built from large amounts of text and already capture language knowledge. Random embeddings learn this information from scratch from the data given. We wanted to see whether this prior knowledge makes a significant difference in model performance.

**Results**

Models which used pre-trained embeddings scored between 70% and 75% accuracy. Random embeddings scored 55-60% except for GloVe which performed a bit better. There is a difference of 10-15 percentage. This is because pre-trained embeddings already contain rich language knowledge. They have been trained on large data already and so they capture word meanings.

Random embeddings start with no information prior. The model has to learn the task and the word representations from scratch. It uses the limited data available in SemEval 2010. This makes it much harder for the model to achieve high accuracy.

## Experiment 2: Evaluating Different Pre-trained Embeddings

We compared three types of pre-trained word embeddings -GloVe, Word2Vec, and FastText- using the same models (CNN, GRU, and LSTM) on the SemEval 2010 Task 8 dataset. The goal was to see which pre-trained embedding gives the best performance.

Pre-trained embeddings are already built with a lot of language knowledge. However, different embeddings are trained on different data and use different methods. We wanted to know if one embedding consistently performed better and by how much do they differ.

**Results:**

**GloVe Outperformed the Others.** Across all three models, **GloVe** achieved about 1-2 percentage higher than Word2Vec or FastText. This probably means that for this dataset and task GloVe captures word meanings and relationships in a better way.

**Word2Vec and FastText** also performed almost well but not as good as GloVe which was more by a small margin. This showed that the word embedding used matters.

## Experiment 3: Comparing Architectures (CNN vs. RNN vs. Hybrid)

We tested three different types of model architectures on the same dataset (SemEval 2010 Task 8):

- A **CNN** model that focuses on extracting local patterns (like key phrases).
- An **RNN** model (using GRU or LSTM) that processes the sentence word by word to capture overall context.
- A **Hybrid model (CNN + BiGRU + Attention)** that combines the strengths of both CNN and RNN, and uses attention to focus on the most important parts of the sentence.

**Results:**

- **CNN: 72.60%**
- **GRU: 74.60%**
- **LSTM: 73.40%**
- **Hybrid (CNN + BiGRU + Attention): 75.30%** (This was the highest).

We wanted to see which type of model architecture works best for our task. CNN is great for learning local clues and RNN is better at understanding the full sentence context. The hybrid model was tested to check if combining these strengths could improve performance even more.

The Hybrid model performed the best. it uses CNN to pick up key phrases and RNN with attention to focus on the most informative parts of longer sentences. This made it get the highest accuracy 75.30%.

## Experiment 4: Freezing vs. Fine-Tuning Embeddings

We took the GloVe embeddings as it was the best embedding out of the three and ran experiments on the SemEval 2010 Task 8 dataset. We used 2 settings.

- **Freezing the embeddings:** The GloVe weights remained unchanged during training.
- **Fine-tuning the embeddings:** The GloVe weights were updated along with the rest of the model during training.

We conducted these tests on both the CNN models.

**Results**

**For the CNN model:**

- **Frozen GloVe:** 73.02%
- **Fine-tuned GloVe:** 74.00%

**For the CNN:** Fine-tuning provided an improvement of about 1–2 percentage over freezing.

We wanted to see if allowing the pre-trained embeddings to adjust to fine-tuning could improve the model's performance compared to keeping them fixed (freezing).

Fine-tuning helps the word embeddings understand our specific task better. This means the words get a little more meaning based on our dataset. This makes the model perform slightly better. Because of this, the accuracy improved by about 1–2% when we fine-tuned the embeddings instead of keeping them fixed.

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

CNN + GRU with Attention) using GloVe word embeddings

```
Classification Report:
                    precision    recall  f1-score   support

      Cause-Effect       0.88      0.87      0.88       328
   Component-Whole       0.85      0.67      0.75       312
 Content-Container       0.76      0.84      0.80       192
Entity-Destination       0.72      0.92      0.81       292
     Entity-Origin       0.79      0.83      0.81       258
 Instrument-Agency       0.70      0.58      0.63       156
 Member-Collection       0.72      0.91      0.80       233
     Message-Topic       0.75      0.85      0.80       261
             Other       0.45      0.31      0.37       454
  Product-Producer       0.67      0.73      0.70       231

          accuracy                           0.73      2717
         macro avg       0.73      0.75      0.73      2717
      weighted avg       0.72      0.73      0.71      2717
```

Bidirectional GRU-based RNN model with GloVe embeddings

```
Classification Report:
                    precision    recall  f1-score   support

      Cause-Effect       0.90      0.92      0.91       328
   Component-Whole       0.78      0.72      0.75       312
 Content-Container       0.81      0.85      0.83       192
Entity-Destination       0.78      0.93      0.85       292
     Entity-Origin       0.79      0.88      0.83       258
 Instrument-Agency       0.65      0.53      0.58       156
 Member-Collection       0.77      0.87      0.82       233
     Message-Topic       0.82      0.74      0.78       261
             Other       0.48      0.41      0.44       454
  Product-Producer       0.69      0.74      0.71       231

          accuracy                           0.74      2717
         macro avg       0.75      0.76      0.75      2717
      weighted avg       0.74      0.74      0.74      2717
```

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

Hybrid CNN + BiGRU model with Attention using Word2Vec

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.83 | 0.89 | 0.86 | 328 |
| Component-Whole | 0.80 | 0.66 | 0.72 | 312 |
| Content-Container | 0.92 | 0.68 | 0.78 | 192 |
| Entity-Destination | 0.74 | 0.91 | 0.82 | 292 |
| Entity-Origin | 0.74 | 0.86 | 0.79 | 258 |
| Instrument-Agency | 0.46 | 0.83 | 0.59 | 156 |
| Member-Collection | 0.69 | 0.85 | 0.76 | 233 |
| Message-Topic | 0.78 | 0.79 | 0.78 | 261 |
| Other | 0.47 | 0.27 | 0.34 | 454 |
| Product-Producer | 0.72 | 0.65 | 0.69 | 231 |
| accuracy | | | 0.71 | 2717 |
| macro avg | 0.71 | 0.74 | 0.71 | 2717 |
| weighted avg | 0.71 | 0.71 | 0.69 | 2717 |

Hybrid CNN + BiGRU model with Attention using FastText word embeddings

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.88 | 0.89 | 0.88 | 328 |
| Component-Whole | 0.74 | 0.75 | 0.75 | 312 |
| Content-Container | 0.71 | 0.88 | 0.78 | 192 |
| Entity-Destination | 0.76 | 0.92 | 0.83 | 292 |
| Entity-Origin | 0.75 | 0.83 | 0.79 | 258 |
| Instrument-Agency | 0.70 | 0.59 | 0.64 | 156 |
| Member-Collection | 0.82 | 0.80 | 0.81 | 233 |
| Message-Topic | 0.73 | 0.85 | 0.79 | 261 |
| Other | 0.40 | 0.29 | 0.34 | 454 |
| Product-Producer | 0.76 | 0.65 | 0.70 | 231 |
| accuracy | | | 0.72 | 2717 |
| macro avg | 0.72 | 0.74 | 0.73 | 2717 |
| weighted avg | 0.71 | 0.72 | 0.71 | 2717 |

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

Seperate using Glove for CNN LSTM and GRU

LSTM Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.77 | 0.93 | 0.84 | 328 |
| Component-Whole | 0.74 | 0.66 | 0.70 | 312 |
| Content-Container | 0.81 | 0.79 | 0.80 | 192 |
| Entity-Destination | 0.81 | 0.82 | 0.82 | 292 |
| Entity-Origin | 0.81 | 0.79 | 0.80 | 258 |
| Instrument-Agency | 0.50 | 0.72 | 0.59 | 156 |
| Member-Collection | 0.68 | 0.90 | 0.77 | 233 |
| Message-Topic | 0.76 | 0.78 | 0.77 | 261 |
| Other | 0.40 | 0.28 | 0.33 | 454 |
| Product-Producer | 0.73 | 0.56 | 0.64 | 231 |
| accuracy | | | 0.70 | 2717 |
| macro avg | 0.70 | 0.72 | 0.71 | 2717 |
| weighted avg | 0.69 | 0.70 | 0.69 | 2717 |

Rndom embedding for all the 3

LSTM (Random Embeddings) Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.84 | 0.80 | 0.82 | 328 |
| Component-Whole | 0.85 | 0.18 | 0.30 | 312 |
| Content-Container | 0.67 | 0.77 | 0.72 | 192 |
| Entity-Destination | 0.66 | 0.86 | 0.74 | 292 |
| Entity-Origin | 0.42 | 0.84 | 0.57 | 258 |
| Instrument-Agency | 0.54 | 0.38 | 0.44 | 156 |
| Member-Collection | 0.49 | 0.64 | 0.56 | 233 |
| Message-Topic | 0.80 | 0.30 | 0.43 | 261 |
| Other | 0.24 | 0.28 | 0.26 | 454 |
| Product-Producer | 0.53 | 0.47 | 0.50 | 231 |
| accuracy | | | 0.53 | 2717 |
| macro avg | 0.60 | 0.55 | 0.53 | 2717 |
| weighted avg | 0.59 | 0.53 | 0.52 | 2717 |

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

GRU (Random Embeddings) Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.56 | 0.90 | 0.69 | 328 |
| Component-Whole | 0.33 | 0.79 | 0.46 | 312 |
| Content-Container | 0.85 | 0.65 | 0.74 | 192 |
| Entity-Destination | 0.77 | 0.74 | 0.75 | 292 |
| Entity-Origin | 0.94 | 0.12 | 0.21 | 258 |
| Instrument-Agency | 0.35 | 0.45 | 0.39 | 156 |
| Member-Collection | 0.42 | 0.64 | 0.51 | 233 |
| Message-Topic | 0.59 | 0.64 | 0.61 | 261 |
| Other | 0.34 | 0.04 | 0.08 | 454 |
| Product-Producer | 0.62 | 0.21 | 0.31 | 231 |
| accuracy | | | 0.50 | 2717 |
| macro avg | 0.58 | 0.52 | 0.48 | 2717 |
| weighted avg | 0.56 | 0.50 | 0.45 | 2717 |

CNN (Random Embeddings) Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.91 | 0.87 | 0.89 | 328 |
| Component-Whole | 0.71 | 0.65 | 0.68 | 312 |
| Content-Container | 0.75 | 0.88 | 0.81 | 192 |
| Entity-Destination | 0.82 | 0.87 | 0.84 | 292 |
| Entity-Origin | 0.82 | 0.81 | 0.82 | 258 |
| Instrument-Agency | 0.44 | 0.69 | 0.54 | 156 |
| Member-Collection | 0.74 | 0.85 | 0.79 | 233 |
| Message-Topic | 0.64 | 0.84 | 0.72 | 261 |
| Other | 0.44 | 0.23 | 0.30 | 454 |
| Product-Producer | 0.64 | 0.65 | 0.64 | 231 |
| accuracy | | | 0.70 | 2717 |
| macro avg | 0.69 | 0.73 | 0.70 | 2717 |
| weighted avg | 0.69 | 0.70 | 0.68 | 2717 |

CNN with Fine-Tuned Pre-trained (GloVe) Embeddings

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.80 | 0.94 | 0.86 | 328 |
| Component-Whole | 0.76 | 0.77 | 0.76 | 312 |
| Content-Container | 0.81 | 0.87 | 0.84 | 192 |
| Entity-Destination | 0.77 | 0.91 | 0.84 | 292 |
| Entity-Origin | 0.85 | 0.82 | 0.83 | 258 |
| Instrument-Agency | 0.63 | 0.69 | 0.66 | 156 |
| Member-Collection | 0.76 | 0.86 | 0.81 | 233 |
| Message-Topic | 0.77 | 0.84 | 0.80 | 261 |
| Other | 0.48 | 0.30 | 0.37 | 454 |
| Product-Producer | 0.75 | 0.68 | 0.71 | 231 |
| accuracy | | | 0.74 | 2717 |
| macro avg | 0.74 | 0.77 | 0.75 | 2717 |
| weighted avg | 0.72 | 0.74 | 0.73 | 2717 |

Ensemble of Multiple Models (CNN, GRU, LSTM)

Ensemble Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.86 | 0.91 | 0.89 | 328 |
| Component-Whole | 0.74 | 0.78 | 0.76 | 312 |
| Content-Container | 0.76 | 0.88 | 0.81 | 192 |
| Entity-Destination | 0.80 | 0.89 | 0.84 | 292 |
| Entity-Origin | 0.80 | 0.87 | 0.83 | 258 |
| Instrument-Agency | 0.60 | 0.73 | 0.66 | 156 |
| Member-Collection | 0.73 | 0.91 | 0.81 | 233 |
| Message-Topic | 0.76 | 0.89 | 0.82 | 261 |
| Other | 0.53 | 0.24 | 0.33 | 454 |
| Product-Producer | 0.73 | 0.69 | 0.71 | 231 |
| accuracy | | | 0.74 | 2717 |
| macro avg | 0.73 | 0.78 | 0.75 | 2717 |
| weighted avg | 0.72 | 0.74 | 0.72 | 2717 |

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

CNN with Frozen Pre-trained (GloVe) Embeddings

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.87 | 0.90 | 0.88 | 328 |
| Component-Whole | 0.78 | 0.72 | 0.75 | 312 |
| Content-Container | 0.75 | 0.86 | 0.80 | 192 |
| Entity-Destination | 0.79 | 0.88 | 0.83 | 292 |
| Entity-Origin | 0.77 | 0.82 | 0.80 | 258 |
| Instrument-Agency | 0.58 | 0.70 | 0.64 | 156 |
| Member-Collection | 0.75 | 0.89 | 0.81 | 233 |
| Message-Topic | 0.78 | 0.83 | 0.80 | 261 |
| Other | 0.46 | 0.30 | 0.36 | 454 |
| Product-Producer | 0.69 | 0.69 | 0.69 | 231 |
| accuracy | | | 0.73 | 2717 |
| macro avg | 0.72 | 0.76 | 0.74 | 2717 |
| weighted avg | 0.71 | 0.73 | 0.72 | 2717 |

Final trying to improve

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Cause-Effect | 0.90 | 0.92 | 0.91 | 328 |
| Component-Whole | 0.75 | 0.77 | 0.76 | 312 |
| Content-Container | 0.76 | 0.89 | 0.82 | 192 |
| Entity-Destination | 0.82 | 0.89 | 0.85 | 292 |
| Entity-Origin | 0.79 | 0.88 | 0.83 | 258 |
| Instrument-Agency | 0.60 | 0.79 | 0.68 | 156 |
| Member-Collection | 0.74 | 0.88 | 0.81 | 233 |
| Message-Topic | 0.78 | 0.89 | 0.83 | 261 |
| Other | 0.52 | 0.26 | 0.35 | 454 |
| Product-Producer | 0.74 | 0.75 | 0.74 | 231 |
| accuracy | | | 0.75 | 2717 |
| macro avg | 0.74 | 0.79 | 0.76 | 2717 |
| weighted avg | 0.73 | 0.75 | 0.73 | 2717 |

Name: Goutham Deepak (gfd5210@psu.edu) Vedant Sawant (vzs5407@psu.edu )

# 6. Lessons & experience we learned

1. **Embeddings is useful**
   **-** Pre-trained vs. Random: Pre-trained embeddings (GloVe/Word2Vec/FastText) outperformed randomly initialized ones.
2. **Hybrid Architecture Work**
   - CNN + RNN work better than standalone networks.
3. **Training & Optimization Practices**
   -Early Stopping: Saved training time and prevented overfitting (patience=3).
4. **Debugging Experience**
   - Fix code error like errors in shape of attention like (None, 100, 256) vs. (None, 256))
   - Reducing memory usage by lowering BATCH_SIZE and simplifying architectures when training on large embeddings (e.g., Word2Vec 300d).
   - Hyperparameter tuning for overfitting. Added Earlystopping for that. We also added dropout layers.
5. **Learned about Ensemble strategy**
   - We learned about soft voting ensemble.
6. **Library dependencies experience**
   - In our studies, we also learned about the compatibility of library versions for TensorFlow, Keras, PyTorch, OpenCV, and scikit-learn.