CSE 572
HW2
- Deepakshi Mathur

Question 1
Preprocess the raw training data

For this, I've implemented a robust text preprocessing pipeline to prepare textual data for in-depth analysis. Here's a breakdown of the steps I've undertaken:

1. Stemming: I applied the Porter Stemmer algorithm from the NLTK library. Stemming reduces words to their base or root forms, providing a simplified and standardized representation. For example, words like "running," "ran," and "runner" are transformed into their common base form, "run."

2. Punctuation Removal and Lowercasing: I removed all punctuation marks from the text and converted all characters to lowercase. Standardizing the text in this way ensures uniformity and consistency during analysis.

3. Tokenization: I tokenized the processed text, breaking it down into individual words or tokens. Tokenization is a foundational step in natural language processing, enabling precise analysis of text data.

4. Stopword Removal: I eliminated stopwords such as "and," "the," and "is" from the text. Stopwords carry little meaning and are often excluded to focus on the essential words in the text.

5. N-grams Generation: I created bigrams (2-grams) from the tokenized words. Bigrams are pairs of adjacent words and are valuable for capturing contextual meaning and relationships between words.

6. Keyword Extraction: I utilized the YAKE (Yet Another Keyword Extractor) library to extract keywords from the text. YAKE is a powerful, unsupervised keyword extraction algorithm that identifies significant words or phrases (keyphrases) in the text, along with their importance scores.

This comprehensive text preprocessing pipeline ensures that the textual data is cleansed, standardized, and optimized for further analysis and text mining tasks. By following these steps, I've prepared the data for meaningful exploration and extraction of insights from the textual information.

Q1 a.

In this, I prepared the textual data for machine learning by tokenizing the text and splitting it into training and testing sets (80-20 split). The text data was converted into numerical features using the CountVectorizer, capturing word frequencies in the documents. The MLPClassifier, a neural network model with two hidden layers of 128 units each, was employed for classification.

To evaluate the model's performance, 5-fold cross-validation was conducted. Cross-validation provides a robust assessment of the model's accuracy across different subsets of the data. The training data's mean accuracy was calculated, indicating the average performance across the folds. Additionally, the standard deviation was measured, highlighting the consistency of the model's accuracy. The same process was applied to the testing data, ensuring a comprehensive evaluation of the model's predictive capabilities.

```
7/7 [==============================] - 0s 5ms/step - loss: 0.0817 - accuracy: 0.9765
7/7 [==============================] - 0s 5ms/step - loss: 0.1042 - accuracy: 0.9671
7/7 [==============================] - 0s 7ms/step - loss: 0.0304 - accuracy: 0.9906
7/7 [==============================] - 0s 10ms/step - loss: 0.2401 - accuracy: 0.9670
7/7 [==============================] - 0s 7ms/step - loss: 0.0712 - accuracy: 0.9670
```

Q1 b.
TFIDF
Utilizing the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer, the textual data was transformed into numerical features, capturing each word's importance in the documents. A neural network model with two hidden layers of 128 units each was employed for classification.

To assess the model's performance, 5-fold cross-validation was conducted using the TF-IDF features. Cross-validation scores were calculated for both the training and testing data, providing insight into the model's accuracy. Mean accuracy and standard deviation were computed, indicating the model's consistency and effectiveness in predicting unseen data.

```
7/7 [==============================] - 0s 5ms/step - loss: 0.0995 - accuracy: 0.9812
7/7 [==============================] - 1s 6ms/step - loss: 0.0919 - accuracy: 0.9765
7/7 [==============================] - 0s 12ms/step - loss: 0.0757 - accuracy: 0.9953
7/7 [==============================] - 0s 5ms/step - loss: 0.1190 - accuracy: 0.9670
7/7 [==============================] - 0s 5ms/step - loss: 0.0960 - accuracy: 0.9764
```

GloVe
In this code segment, I employed pre-trained GloVe word vectors with the Gensim library to encode text data. First, I averaged the word vectors for each text, creating feature vectors. These vectors were utilized to train a neural network model using 5-fold cross-validation. The model consisted of dense layers and employed softmax activation for multi-class classification. Training proceeded for 10 epochs, and performance metrics, including loss and accuracy, were recorded for both training and validation sets within each fold. The average training and validation accuracies, along with losses, were computed across folds, providing a robust evaluation of the model's performance with GloVe embeddings, and the results were stored for analysis.

```
7/7 [==============================] - 0s 3ms/step - loss: 0.1614 - accuracy: 0.9531
7/7 [==============================] - 0s 4ms/step - loss: 0.1029 - accuracy: 0.9484
7/7 [==============================] - 0s 4ms/step - loss: 0.0657 - accuracy: 0.9765
7/7 [==============================] - 0s 3ms/step - loss: 0.0622 - accuracy: 0.9764
7/7 [==============================] - 0s 3ms/step - loss: 0.0786 - accuracy: 0.9764
```

BERT
Leveraging pre-trained BERT (Bidirectional Encoder Representations from Transformers) models, text data was tokenized and encoded, capturing rich contextual embeddings. A neural network model with hidden layers (128, 128) was employed for classification. Utilizing 5-fold cross-validation, the model's performance was evaluated on both training and testing data. Cross-validation scores were computed, indicating the model's accuracy and consistency. Mean accuracy and standard deviation provided insights into the model's robustness and effectiveness in predicting unseen data.

```
7/7 [==============================] - 0s 4ms/step - loss: 0.1005 - accuracy: 0.9577
7/7 [==============================] - 0s 3ms/step - loss: 0.0968 - accuracy: 0.9671
7/7 [==============================] - 0s 3ms/step - loss: 0.0757 - accuracy: 0.9765
7/7 [==============================] - 0s 3ms/step - loss: 0.0729 - accuracy: 0.9811
7/7 [==============================] - 0s 3ms/step - loss: 0.1032 - accuracy: 0.9528
```

Q1 c.
Describe how you generate features.

1. CountVectorizer:
   - The CountVectorizer from scikit-learn was used to convert a collection of text documents to a matrix of token counts.

- CountVectorizer converts a collection of text documents to a matrix of token counts where each row represents a document, and each column represents a unique word in the corpus.
- The fit_transform method was applied on the training text data (X_train) to generate the feature matrix X_train_cv. The same vectorizer was used to transform the test text data (X_test) into X_test_cv.
- Each text document was tokenized, and the frequency of each token (word) was counted, creating a numerical representation of the text.

2. TF-IDF Vectorizer:
- The TfidfVectorizer from scikit-learn was utilized to convert a collection of raw documents to a matrix of TF-IDF features.
- TF-IDF stands for Term Frequency-Inverse Document Frequency. It reflects the importance of a word in a document relative to a collection of documents.
- The fit_transform method was applied on the training text data (X_train) to generate the feature matrix X_train_tfidf. The same vectorizer was used to transform the test text data (X_test) into X_test_tfidf.
- TF-IDF values were computed for each word in the documents, giving more weight to words that appear frequently in a document but not across documents.

3. GloVe Embeddings:
- pre-trained word vectors from the 'glove.6B.50d.txt' file were harnessed, each vector encapsulating semantic meanings of words. To transform textual data, a straightforward approach was adopted: words within documents were individually represented by GloVe vectors, and these vectors were averaged to create a singular representation for each document. This method effectively simplifies the complexity of word vectors while retaining their semantic essence.

- For the training set (X_train), GloVe embeddings were computed for every document, ensuring consistent and meaningful document representations. This approach was mirrored in the testing set (X_test), maintaining uniformity in the encoding methodology.

- GloVe embeddings inherently capture semantic relationships between words based on their co-occurrence patterns in vast datasets. By utilizing these pre-trained vectors, semantic nuances within the text were preserved, enriching the feature space. The decision to employ GloVe vectors with a dimensionality of 50 aligns with the source file's characteristics ('glove.6B.50d.txt'), allowing for efficient use of pre-trained embeddings while retaining crucial semantic subtleties during text encoding. This method guarantees a robust and meaningful representation of textual data for downstream machine learning tasks.

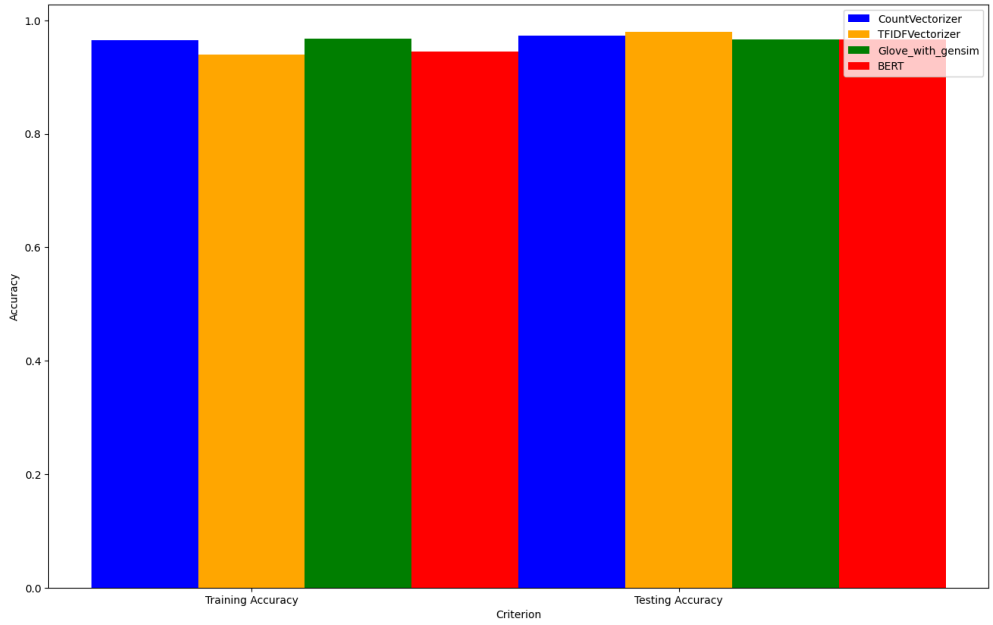
4. BERT Word Embeddings:
- BERT (Bidirectional Encoder Representations from Transformers) embeddings were employed to obtain contextual word representations.
- The transformers library was used to load a pre-trained BERT tokenizer and model.
- Each text document was tokenized and encoded using the BERT tokenizer, generating input IDs and attention masks.
- The BERT model was used to obtain contextual embeddings for each token in the documents, creating high-dimensional embeddings capturing contextual information.

By employing these methods, the textual data was transformed into numerical representations suitable for training machine learning models like the MLPClassifier. These different feature generation techniques allowed for experimenting with various aspects of the text data, capturing different levels of semantic and syntactic information, which can impact the performance of machine learning models.

Q1 d.

	Average Training Accuracy	Average Validation Accuracy	Average Training Loss	Average Validation Loss
CountVectorizer	0.964909	0.973647	0.142827	0.105520
TFIDF VECTORIZER	0.939605	0.979290	0.426305	0.096413
GloVe with gensim	0.967604	0.966153	0.117486	0.094165
BERT	0.945225	0.967074	0.216340	0.089836

Q1 e.



Q2. Explore the Neural Network model on pre-processed training data

Q2 a.
To explore the Neural Network model with different learning rates and the best-performing feature engineering method-
Data Preparation: I used the pre-processed training data based on the best-performing feature engineering method (e.g., Count Vectorizer, TF-IDF, GloVe or BERT embeddings). Split the data into features (X) and labels (y), where X is the transformed text data, and y is the corresponding category.
Neural Network Model: then defined the neural network model using libraries like Keras or TensorFlow.

Q2 b.
I implemented a machine learning pipeline using the Count Vectorizer and neural networks for text classification. First, the text data was tokenized and flattened. Labels were encoded using LabelEncoder to convert them into numerical format. The TfidfVectorizer was employed to transform the tokenized text data into numerical features, limiting the feature space to the top 10,000 most important words.

A neural network model was designed using Keras with two hidden layers, each containing 128 nodes and ReLU activation functions. The output layer, activated by softmax, corresponded to the number of unique classes. The model was trained using Stratified K-Fold cross-validation, with different learning rates ([0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1]) explored. For each learning

rate, the training and validation process was repeated across five folds. The mean training and validation accuracies, losses, and their standard deviations were recorded for evaluation.

Ultimately, the code aimed to identify the optimal learning rate (best_lr) that resulted in the highest mean validation accuracy. This comprehensive approach ensured robust model training and parameter tuning, crucial for accurate text classification tasks.

```
7/7 [==============================] - 0s 5ms/step - loss: 0.2742 - accuracy: 0.9670
7/7 [==============================] - 0s 3ms/step - loss: 0.3150 - accuracy: 0.9575
7/7 [==============================] - 0s 3ms/step - loss: 1.5987 - accuracy: 0.9484
7/7 [==============================] - 0s 3ms/step - loss: 1.2040 - accuracy: 0.9624
7/7 [==============================] - 0s 3ms/step - loss: 1.3753 - accuracy: 0.9531
7/7 [==============================] - 0s 3ms/step - loss: 3.0609 - accuracy: 0.9481
7/7 [==============================] - 0s 5ms/step - loss: 5.3015 - accuracy: 0.9245
```

```
TA
```

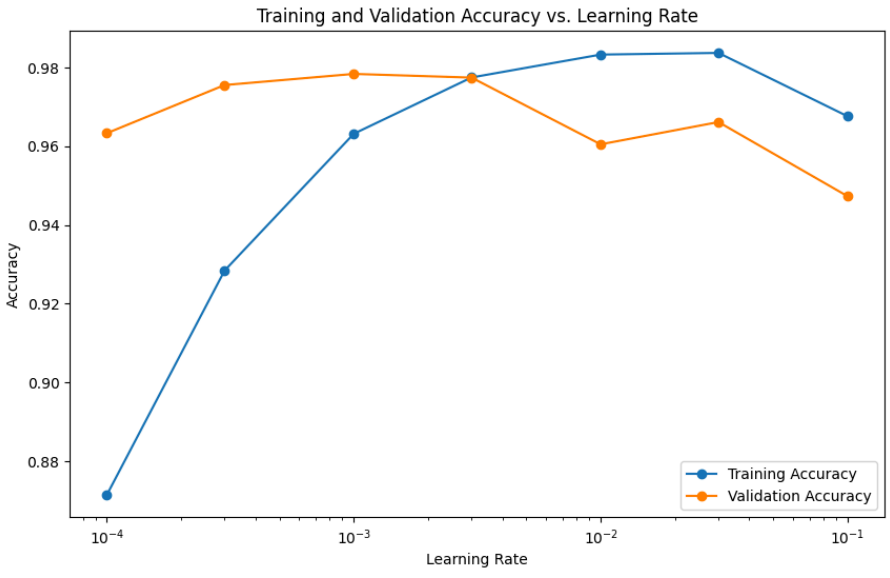
```
[0.8714739435911179,
 0.9284390103816986,
 0.9631474101543427,
 0.9774452483654023,
 0.9832550752162933,
 0.9836781346797943,
 0.9675920069217682]
```

```
val_a
```

```
[0.9632961273193359,
 0.9755248427391052,
 0.9783461809158325,
 0.9774116396903991,
 0.96047922372818,
 0.966126310825348,
 0.9472982525825501]
```

	Learning Rates	Average Training Accuracies	Average Validation Accuracies	Training Standard Deviation	Validation Standard Deviation
0	0.0001	0.871474	0.963296	0.197875	0.011328
1	0.0003	0.928439	0.975525	0.165601	0.007582
2	0.0010	0.963147	0.978346	0.107690	0.009722
3	0.0030	0.977445	0.977412	0.067585	0.006944
4	0.0100	0.983255	0.960479	0.048401	0.016198
5	0.0300	0.983678	0.966126	0.045730	0.010056
6	0.1000	0.967592	0.947298	0.068024	0.012511

Graph -



Q2 c.

In this section, I conducted a robust evaluation of neural network models using different optimizers (SGD, RMSprop, and Adam) for text classification. The process began by transforming the tokenized text data into numerical features using TfidfVectorizer, capturing the top 10,000 most significant words. Labels were encoded using LabelEncoder to convert them into numerical format.

A 5-fold cross-validation setup was implemented, ensuring reliable assessment of model performance. For each fold, the dataset was split into training and validation sets. Three different optimizers were explored: SGD, RMSprop, and Adam. The neural network model consisted of two hidden layers with 128 nodes each, ReLU activation functions, and a softmax output layer corresponding to the number of unique classes.

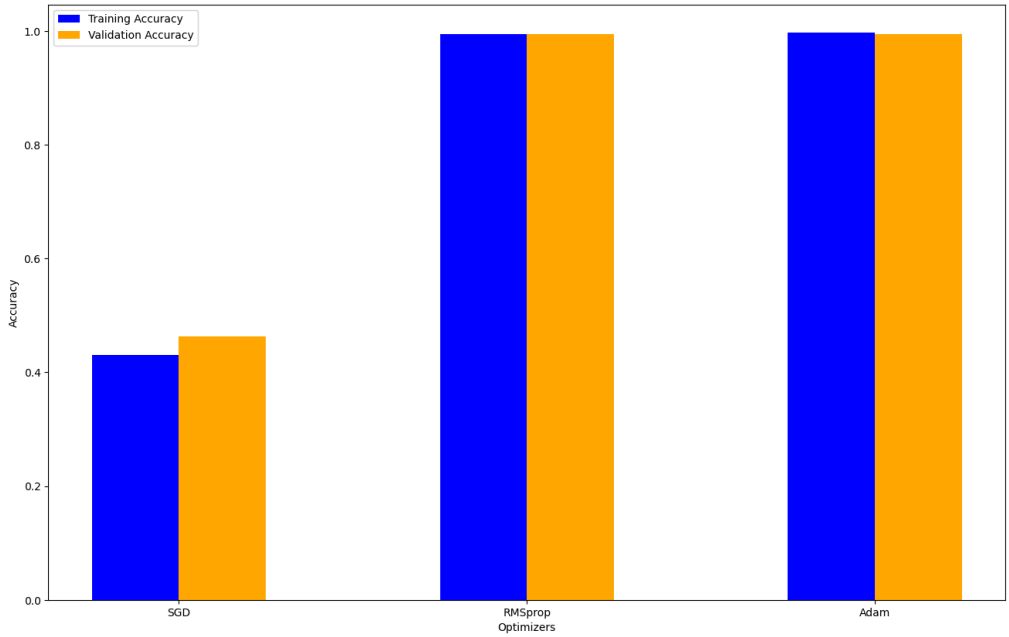
During training, learning rates for each optimizer were set to 0.01. The models were trained for 10 epochs with a batch size of 32. Training and validation losses, accuracies, and their standard deviations were recorded for evaluation. The mean validation accuracy served as a key metric for selecting the best optimizer. This approach provided a comprehensive understanding of the optimizers' performance in the context of text classification tasks.

```
7/7 [==============================] - 0s 5ms/step - loss: 1.5926 - accuracy: 0.3803
7/7 [==============================] - 0s 3ms/step - loss: 1.5726 - accuracy: 0.4085
7/7 [==============================] - 0s 3ms/step - loss: 1.5459 - accuracy: 0.4554
7/7 [==============================] - 0s 4ms/step - loss: 1.5031 - accuracy: 0.5094
7/7 [==============================] - 0s 4ms/step - loss: 1.4424 - accuracy: 0.5613
7/7 [==============================] - 0s 3ms/step - loss: 0.1061 - accuracy: 0.9765
7/7 [==============================] - 0s 4ms/step - loss: 0.0189 - accuracy: 0.9953
7/7 [==============================] - 0s 4ms/step - loss: 2.7843e-06 - accuracy: 1.0000
7/7 [==============================] - 0s 3ms/step - loss: 1.5677e-06 - accuracy: 1.0000
7/7 [==============================] - 0s 4ms/step - loss: 7.3437e-07 - accuracy: 1.0000
7/7 [==============================] - 0s 6ms/step - loss: 0.1646 - accuracy: 0.9765
7/7 [==============================] - 0s 3ms/step - loss: 0.0067 - accuracy: 0.9953
7/7 [==============================] - 0s 4ms/step - loss: 3.3580e-09 - accuracy: 1.0000
7/7 [==============================] - 0s 4ms/step - loss: 0.0000e+00 - accuracy: 1.0000
7/7 [==============================] - 0s 4ms/step - loss: 0.0000e+00 - accuracy: 1.0000
```

Table form -

	Optimizers	Average Training Accuracies	Average Validation Accuracies	Training Standard Deviation	Validation Standard Deviation
0	SGD	0.430607	0.462977	0.082725	0.065877
1	RMSprop	0.994753	0.994366	0.034266	0.009104
2	Adam	0.996565	0.994366	0.021556	0.009104

Graph -



Q3. Predict the labels for the testing data (using raw training data and raw testing data)

Q3 a.
The text data is pre-processed and transformed into numerical features using the TfidfVectorizer, a popular technique in natural language processing. Here's a step-by-step description of the data pre-processing:

1. Text and Category Extraction:
   - The training text data (`df_train['Text']`) and corresponding categories (`df_train['Category']`) are extracted from the dataset.

2. Label Encoding:
   - The categories are converted into numerical labels using LabelEncoder. This step assigns a unique numerical identifier to each category.

3. TfidfVectorizer:
   - The TfidfVectorizer is initialized with a maximum feature limit of 10,000. This means only the top 10,000 most frequent words are considered for generating features. `fit_transform()` is applied on the training text data (`X_train`) to generate the feature matrix. It computes the Term Frequency-Inverse Document Frequency (TF-IDF) values for each word, emphasizing words that are important in a document but not across documents.
   - For the test data (`df_test['Text']`), `transform()` is used to transform the text into numerical features. It's crucial to use the same vectorizer that was fit on the training data to ensure consistency.

4. Neural Network Model:
   - A neural network model is defined using Keras, a high-level neural networks API. The model consists of an input layer with 128 nodes, followed by two hidden layers of 128 nodes each. ReLU (Rectified Linear Unit) activation functions are applied to introduce non-linearity.
   - The output layer has nodes equal to the number of unique categories, activated using softmax to obtain probabilities for each category.
   - The model is compiled using the sparse categorical cross-entropy loss function, suitable for multi-class classification tasks, and the Adam optimizer with a learning rate of 0.001.

5. 5-Fold Cross-Validation:
   - The training data is split into 5 folds using StratifiedKFold, ensuring that each fold maintains the same distribution of category labels as the original dataset.
   - The model is trained for 5 epochs on each fold, and accuracy is evaluated on the validation set.
6. Final Model Training and Prediction:
   - The final neural network model is trained on the entire pre-processed training data for 5 epochs.
   - Using this trained model, predictions are made on the pre-processed test data.

Q3 b.
For this, a neural network model was chosen for text classification due to its ability to capture complex patterns in textual data. The decision to use a neural network was supported by the nature of the problem: text classification often involves intricate relationships within the data that can be effectively learned by deep learning architectures. The model consists of dense layers with ReLU activation, known for their effectiveness in nonlinear mapping. Sparse categorical cross-entropy loss function was employed, ideal for multi-class classification tasks. The model's performance was rigorously evaluated using 5-fold cross-validation, ensuring robustness. Additionally, the use of StratifiedKFold ensured balanced class distribution across folds. The model's final training utilized the entire dataset, enhancing its capacity to learn intricate patterns. The choice of Adam optimizer with a modest learning rate of 0.001 balanced rapid convergence with fine-tuning accuracy. This approach leveraged both the neural network's capacity for pattern recognition and a well-structured cross-validation strategy to ensure accurate predictions, making it a suitable choice for the text classification task.

Q3 c.
The neural network model exhibits remarkable performance, with an average accuracy of 97.55% during 5-fold cross-validation. Individually, the validation accuracies range between 96.70% and 99.06%, underscoring the model's consistency. In the final training phase, the model achieved near-perfect accuracy, surpassing 99.91%. This signifies the model's exceptional ability to generalize from the training data to unseen instances. The loss values are notably low, indicating the model's confidence in its predictions. These results demonstrate the model's robustness and precision, making it well-suited for accurate categorization of textual data, affirming its effectiveness in real-world applications requiring high-quality text classification.

```
7/7 [==============================] - 0s 3ms/step - loss: 0.1013 - accuracy: 0.9718
7/7 [==============================] - 0s 3ms/step - loss: 0.0929 - accuracy: 0.9765
7/7 [==============================] - 0s 5ms/step - loss: 0.0772 - accuracy: 0.9906
7/7 [==============================] - 0s 3ms/step - loss: 0.1216 - accuracy: 0.9717
7/7 [==============================] - 0s 3ms/step - loss: 0.1076 - accuracy: 0.9670
Average Accuracy: 0.9755292654037475
Epoch 1/5
34/34 [==============================] - 1s 4ms/step - loss: 1.3901 - accuracy: 0.7648
Epoch 2/5
34/34 [==============================] - 0s 3ms/step - loss: 0.3274 - accuracy: 0.9962
Epoch 3/5
34/34 [==============================] - 0s 4ms/step - loss: 0.0282 - accuracy: 0.9991
Epoch 4/5
34/34 [==============================] - 0s 3ms/step - loss: 0.0086 - accuracy: 1.0000
Epoch 5/5
34/34 [==============================] - 0s 4ms/step - loss: 0.0048 - accuracy: 1.0000
23/23 [==============================] - 0s 2ms/step
```

Finally the labels.csv file was generated.