Deep Learning Final Project Report

Problem Statement:

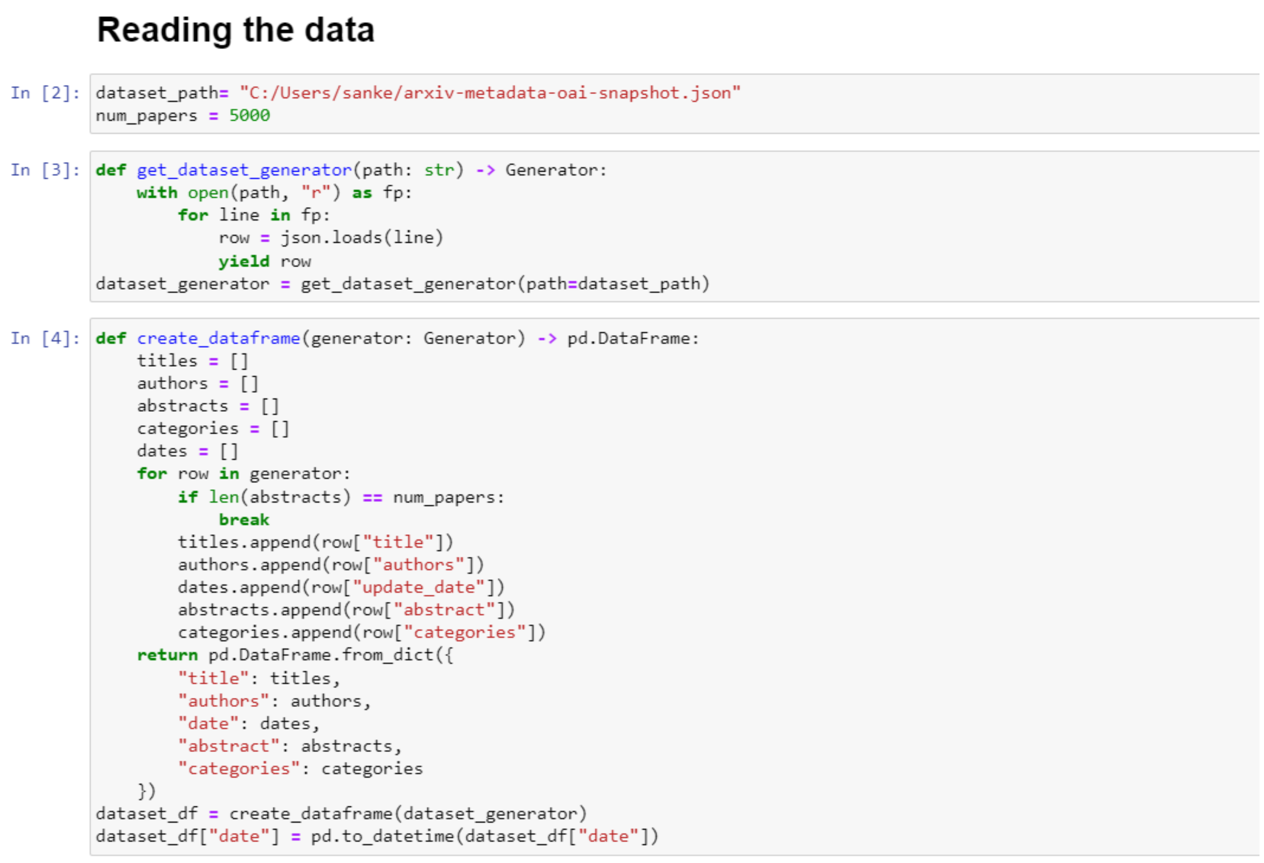
Generating research paper titles based on past titles and research progress. Creating a model to generate research paper titles by embedding various deep learning techniques, natural language processing, and different language models.

Dataset Description:

The dataset is from ArXiv, which is an open access platform that collects data about scientific research papers related to physics, computer science, and other subfields of science. The dataset used in this project is of JSON type and consists of data related to 1.7 million articles. It consists of relevant features like title, author, category, abstract and date.

Data Preprocessing:

To keep the process simple, the dataset which is in JSON format is transformed into pandas data frame. And only 5000 records are used for the initial model building.

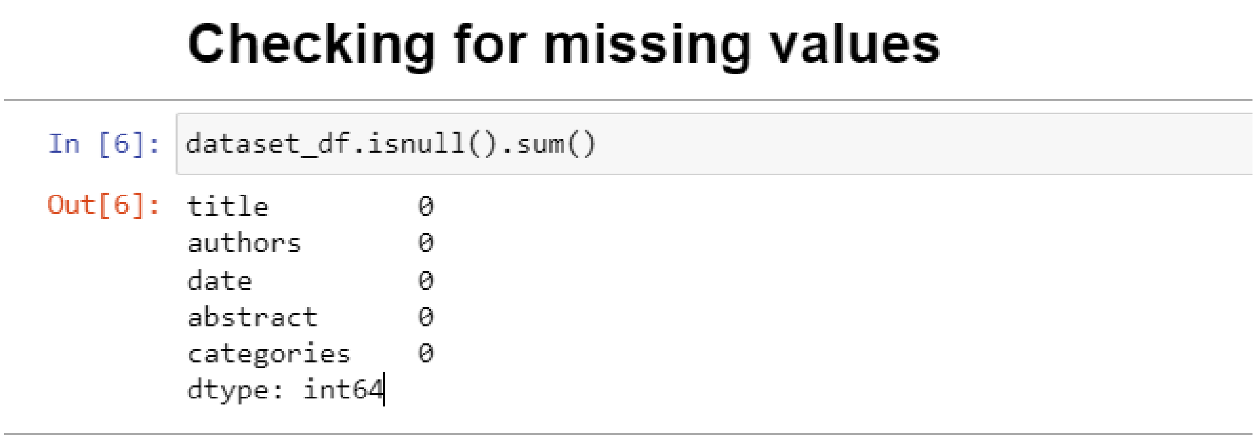


Reading the data

```
In [2]: dataset_path= "C:/Users/sanke/arxiv-metadata-oai-snapshot.json"
        num_papers = 5000
```

```
In [3]: def get_dataset_generator(path: str) -> Generator:
            with open(path, "r") as fp:
                for line in fp:
                    row = json.loads(line)
                    yield row
        dataset_generator = get_dataset_generator(path=dataset_path)
```

```
In [4]: def create_dataframe(generator: Generator) -> pd.DataFrame:
            titles = []
            authors = []
            abstracts = []
            categories = []
            dates = []
            for row in generator:
                if len(abstracts) == num_papers:
                    break
                titles.append(row["title"])
                authors.append(row["authors"])
                dates.append(row["update_date"])
                abstracts.append(row["abstract"])
                categories.append(row["categories"])
            return pd.DataFrame.from_dict({
                "title": titles,
                "authors": authors,
                "date": dates,
                "abstract": abstracts,
                "categories": categories
            })
        dataset_df = create_dataframe(dataset_generator)
        dataset_df["date"] = pd.to_datetime(dataset_df["date"])
```



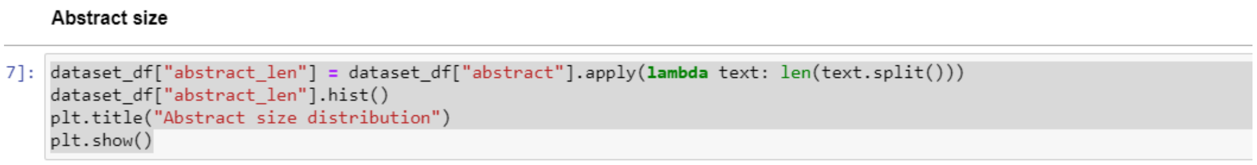
Checking for missing values

```
In [6]: dataset_df.isnull().sum()

Out[6]: title         0
        authors       0
        date          0
        abstract      0
        categories    0
        dtype: int64
```

No missing values

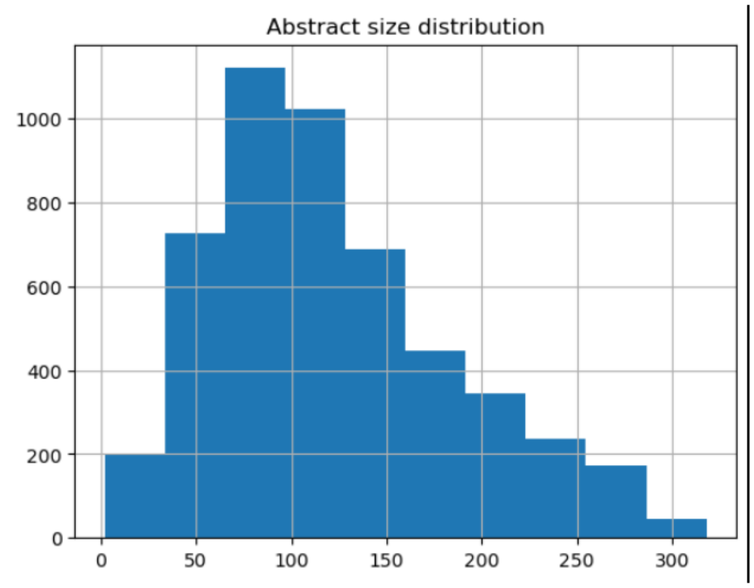
Null values were not found in the dataset, which removes the process of replacing or filling the null values.

Visualization:



Abstract size

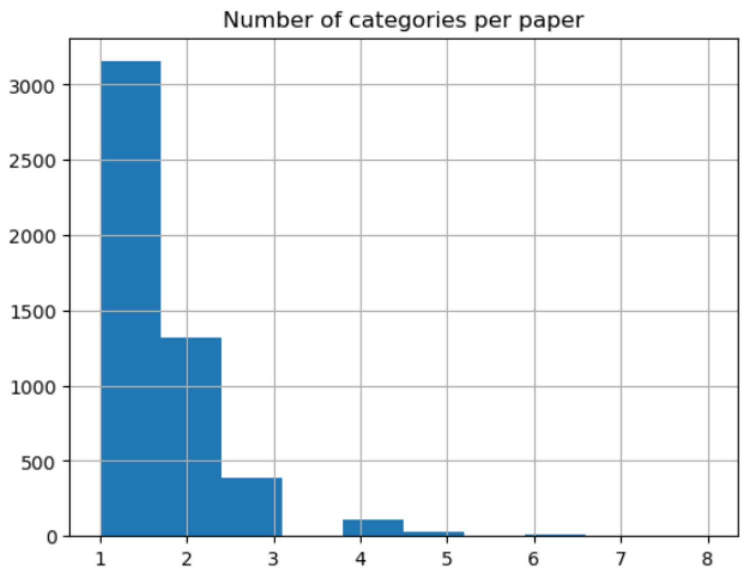
```
7]: dataset_df["abstract_len"] = dataset_df["abstract"].apply(lambda text: len(text.split()))
    dataset_df["abstract_len"].hist()
    plt.title("Abstract size distribution")
    plt.show()
```



The above plot depicts the distribution of abstract size in the research article. We can infer that there are more than 1000 words in the abstract in many articles. We can also infer that the variance in this histogram is wider, by which we can say that there is great diversity in abstract length.

Number of categories per paper

```
In [8]: dataset_df["categories"] = dataset_df["categories"].apply(lambda text: tuple(text.split()))
        dataset_df["num_categories"] = dataset_df["categories"].apply(lambda cats: len(cats))
        dataset_df["num_categories"].hist()
        plt.title("Number of categories per paper")
        plt.show()
```



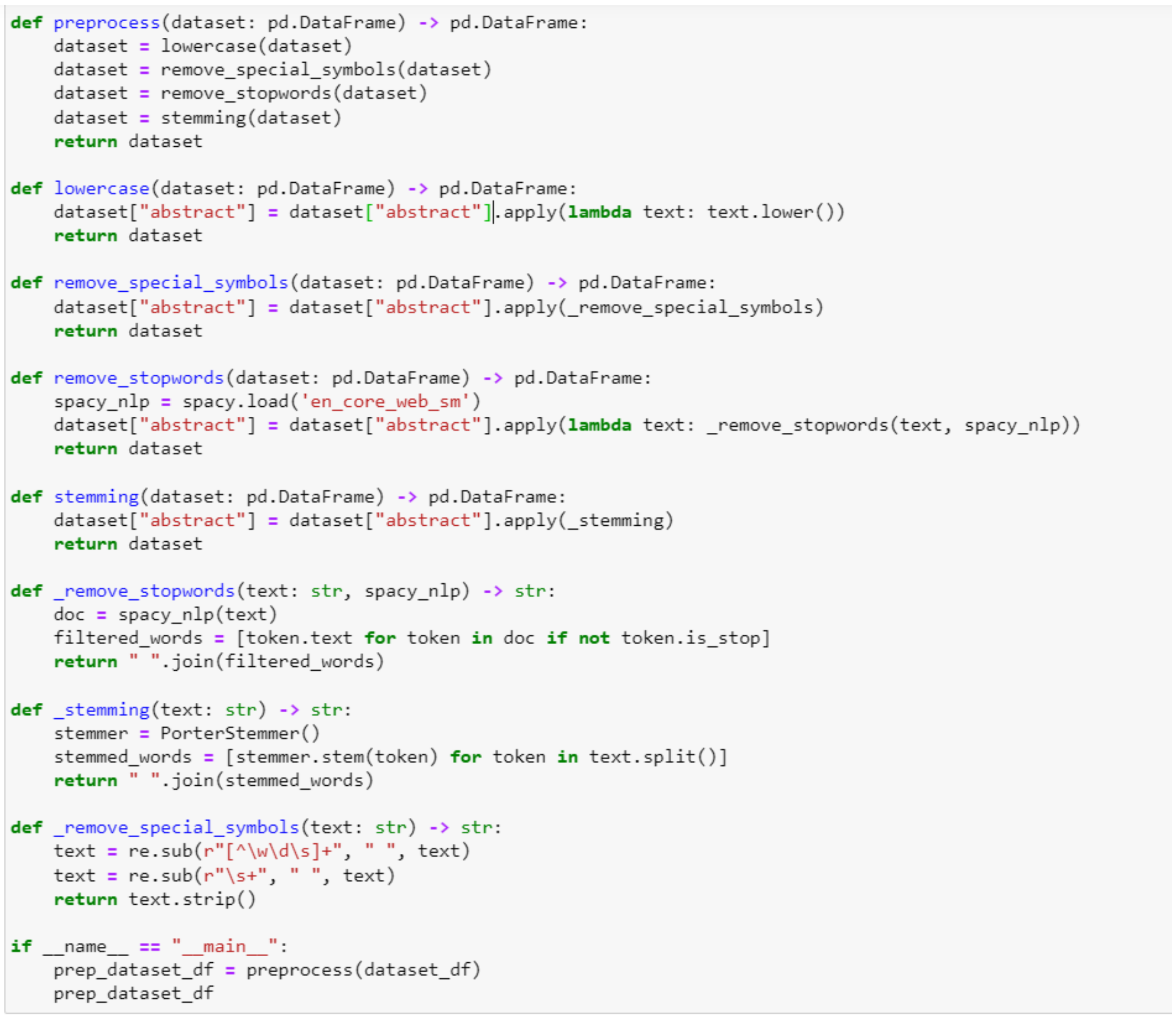
There are very few articles with more than one category.

Number of unique papers

```
|: print(f"Num. papers: {len(dataset_df)}")
   dataset_df = dataset_df[~dataset_df[["title", "authors"]].duplicated()]
   print(f"Num. unique papers: {len(dataset_df)}")
```

```
Num. papers: 5000
Num. unique papers: 4998
```

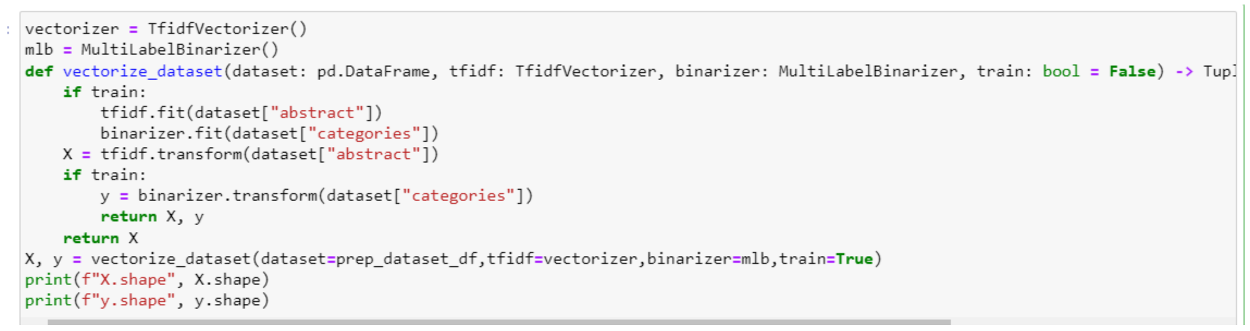
Two papers do not have a unique title and author name.



```python
def preprocess(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset = lowercase(dataset)
    dataset = remove_special_symbols(dataset)
    dataset = remove_stopwords(dataset)
    dataset = stemming(dataset)
    return dataset

def lowercase(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset["abstract"] = dataset["abstract"].apply(lambda text: text.lower())
    return dataset

def remove_special_symbols(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset["abstract"] = dataset["abstract"].apply(_remove_special_symbols)
    return dataset

def remove_stopwords(dataset: pd.DataFrame) -> pd.DataFrame:
    spacy_nlp = spacy.load('en_core_web_sm')
    dataset["abstract"] = dataset["abstract"].apply(lambda text: _remove_stopwords(text, spacy_nlp))
    return dataset

def stemming(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset["abstract"] = dataset["abstract"].apply(_stemming)
    return dataset

def _remove_stopwords(text: str, spacy_nlp) -> str:
    doc = spacy_nlp(text)
    filtered_words = [token.text for token in doc if not token.is_stop]
    return " ".join(filtered_words)

def _stemming(text: str) -> str:
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(token) for token in text.split()]
    return " ".join(stemmed_words)

def _remove_special_symbols(text: str) -> str:
    text = re.sub(r"[^\w\d\s]+", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()

if __name__ == "__main__":
    prep_dataset_df = preprocess(dataset_df)
    prep_dataset_df
```

The above code acts as a pipeline in cleaning and preparing the text data for training the model. It applies lowercase conversion, removal of special symbols and stop words, and word stemming.

Then the text data is vectorized using TfidVectorizer() to quantify the importance of words in the document.



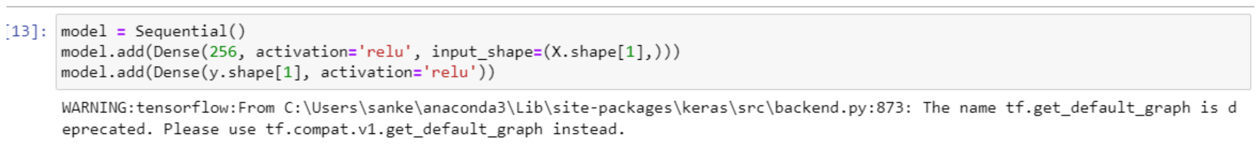
```python
vectorizer = TfidfVectorizer()
mlb = MultiLabelBinarizer()
def vectorize_dataset(dataset: pd.DataFrame, tfidf: TfidfVectorizer, binarizer: MultiLabelBinarizer, train: bool = False) -> Tupl
    if train:
        tfidf.fit(dataset["abstract"])
        binarizer.fit(dataset["categories"])
    X = tfidf.transform(dataset["abstract"])
    if train:
        y = binarizer.transform(dataset["categories"])
        return X, y
    return X
X, y = vectorize_dataset(dataset=prep_dataset_df,tfidf=vectorizer,binarizer=mlb,train=True)
print(f"X.shape", X.shape)
print(f"y.shape", y.shape)
```

This code segment prepares the text data for multi-label classification by converting text features into weighted vectors (X) and labels into binary matrix representation (y).

Then the dataset is split into training and validation sets.

Model setup:

Model building



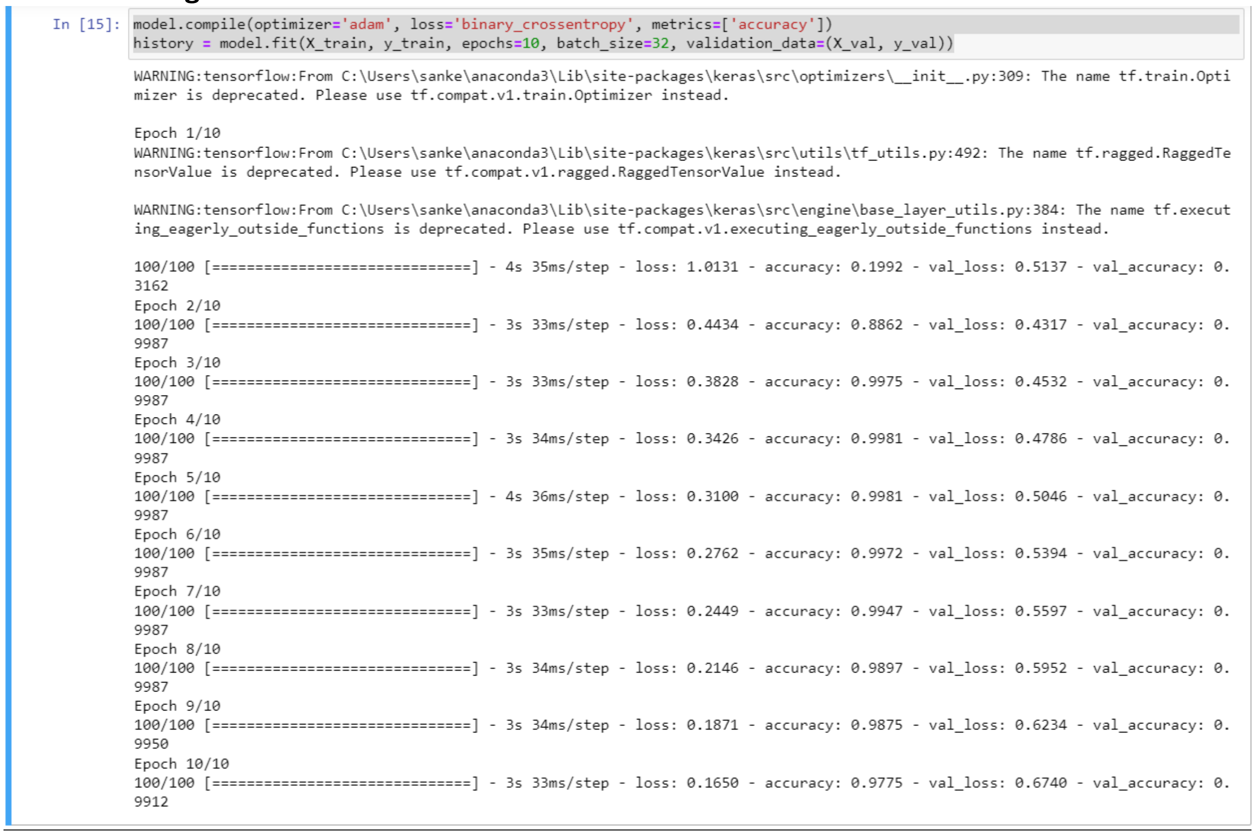
```
[13]: model = Sequential()
      model.add(Dense(256, activation='relu', input_shape=(X.shape[1],)))
      model.add(Dense(y.shape[1], activation='relu'))
```

```
WARNING:tensorflow:From C:\Users\sanke\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is d
eprecated. Please use tf.compat.v1.get_default_graph instead.
```

The model consists of two Dense Layers, the first layer has a reLu activation function and the second layer has a sigmoid activation function.

The hidden layer has 256 neurons, and the final dense layer has the number of neurons equal to the shape of the Y feature which is 38.

Model Training:



```
In [15]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
         history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

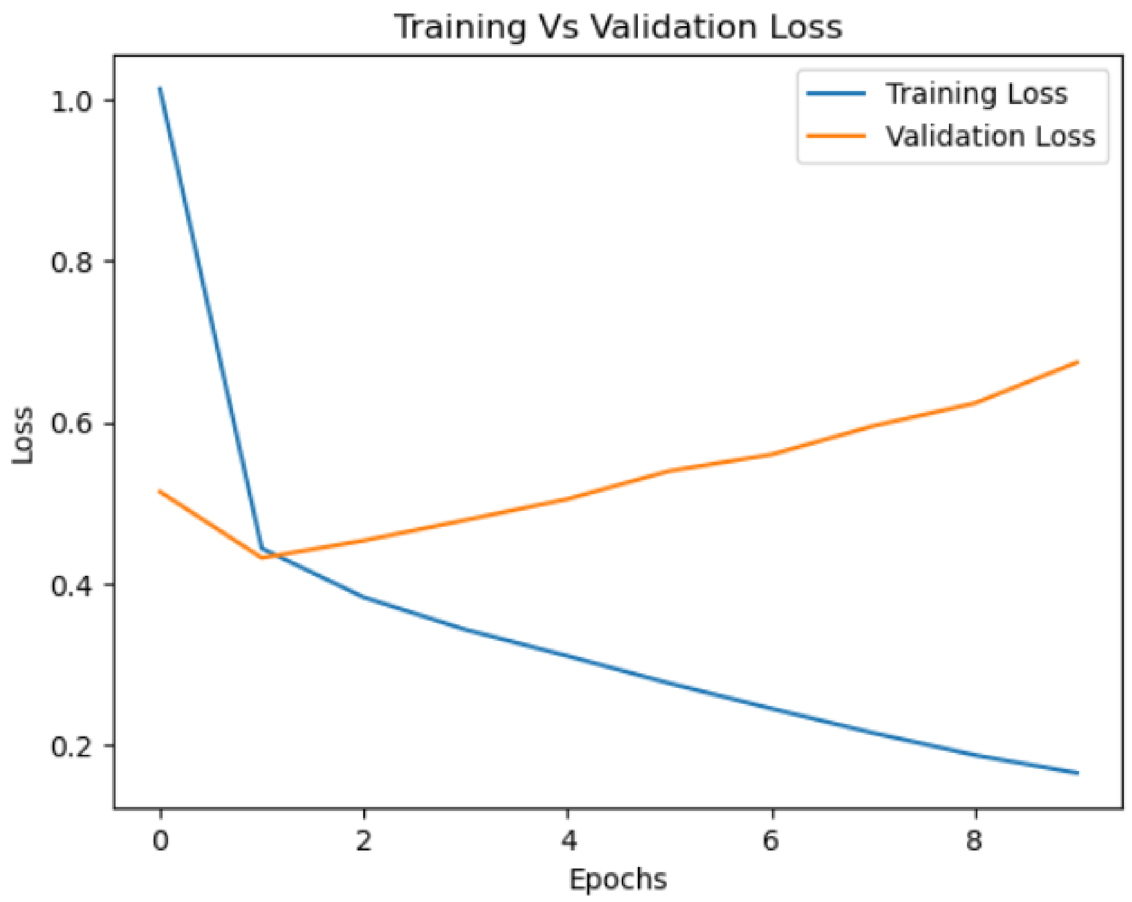
```
WARNING:tensorflow:From C:\Users\sanke\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Opti
mizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/10
WARNING:tensorflow:From C:\Users\sanke\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTe
nsorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\sanke\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.execut
ing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

100/100 [==============================] - 4s 35ms/step - loss: 1.0131 - accuracy: 0.1992 - val_loss: 0.5137 - val_accuracy: 0.
3162
Epoch 2/10
100/100 [==============================] - 3s 33ms/step - loss: 0.4434 - accuracy: 0.8862 - val_loss: 0.4317 - val_accuracy: 0.
9987
Epoch 3/10
100/100 [==============================] - 3s 33ms/step - loss: 0.3828 - accuracy: 0.9975 - val_loss: 0.4532 - val_accuracy: 0.
9987
Epoch 4/10
100/100 [==============================] - 3s 34ms/step - loss: 0.3426 - accuracy: 0.9981 - val_loss: 0.4786 - val_accuracy: 0.
9987
Epoch 5/10
100/100 [==============================] - 4s 36ms/step - loss: 0.3100 - accuracy: 0.9981 - val_loss: 0.5046 - val_accuracy: 0.
9987
Epoch 6/10
100/100 [==============================] - 3s 35ms/step - loss: 0.2762 - accuracy: 0.9972 - val_loss: 0.5394 - val_accuracy: 0.
9987
Epoch 7/10
100/100 [==============================] - 3s 33ms/step - loss: 0.2449 - accuracy: 0.9947 - val_loss: 0.5597 - val_accuracy: 0.
9987
Epoch 8/10
100/100 [==============================] - 3s 34ms/step - loss: 0.2146 - accuracy: 0.9897 - val_loss: 0.5952 - val_accuracy: 0.
9987
Epoch 9/10
100/100 [==============================] - 3s 34ms/step - loss: 0.1871 - accuracy: 0.9875 - val_loss: 0.6234 - val_accuracy: 0.
9950
Epoch 10/10
100/100 [==============================] - 3s 33ms/step - loss: 0.1650 - accuracy: 0.9775 - val_loss: 0.6740 - val_accuracy: 0.
9912
```

The model is trained for 10 epochs with Adam optimizer.

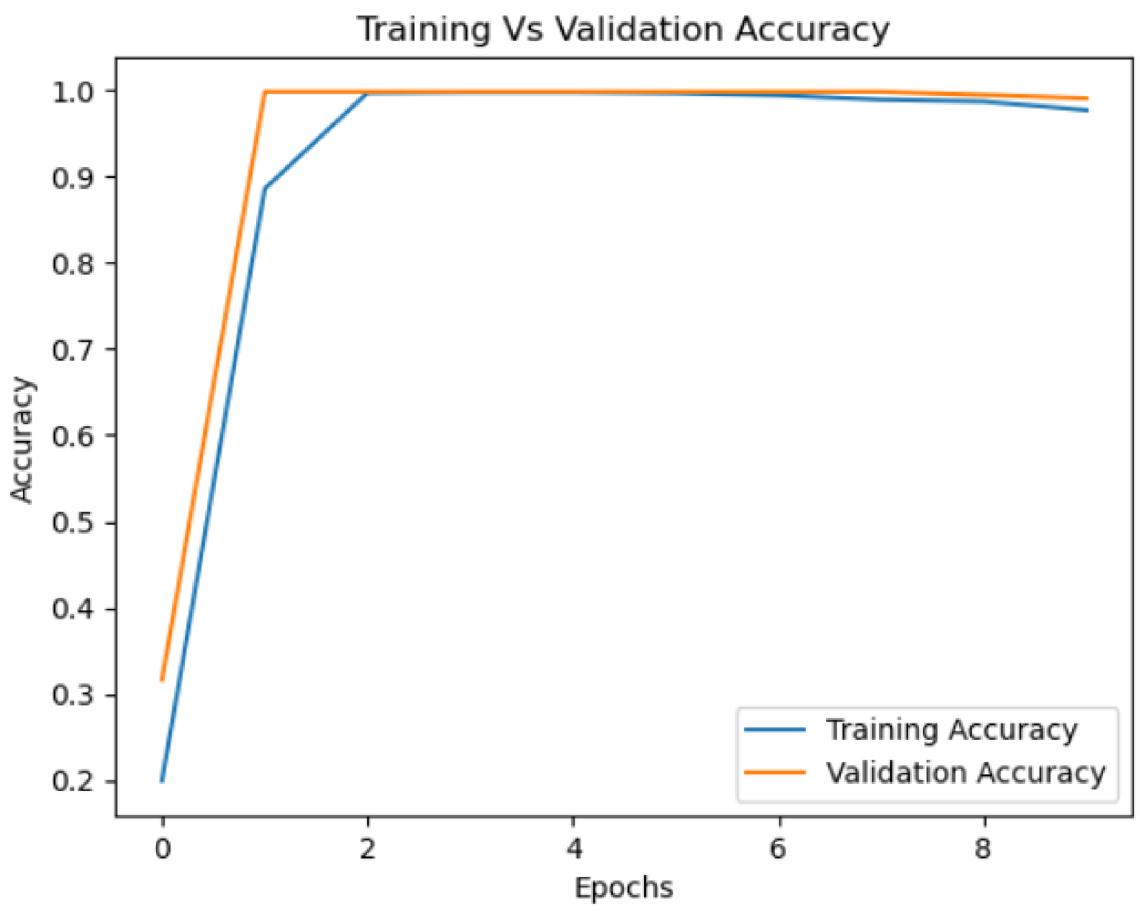
Accuracy and loss are chosen as the evaluation metrics for this model.

After 10 epochs of training the model received a trained accuracy of 0.9775 and validation accuracy of 0.9912.

The trained was able to perform well in the testing set with an accuracy rate of 0.995 and a loss of 0.69



Training loss decreased over the epochs, however we can observe a slight increase in validation loss over the epochs.



Both training and validation accuracy increased over the epochs.

The next phase of this project involves exploring various innovative model architectures to enhance the application's performance and capabilities.

Project By:

Rohith Dinakaran
Sankeerth Sridhar Narayan
Ssneha Balasubramanium