Research Papers dataset link:: https://www.kaggle.com/datasets/spsayakpaul/arxiv-paper-abstracts/data

1 Section:

Loading tools and dataset

In [1]:
```
from tensorflow.keras import layers
from tensorflow import keras
import tensorflow as tf

from sklearn.model_selection import train_test_split

from ast import literal_eval
# is used for safely evaluating strings containing Python literals or conta
# (e.g., lists, dictionaries) to their corresponding Python objects.

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
c:\Users\HP\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarnin
g: A NumPy version >=1.18.5 and <1.25.0 is required for this version of Sc
iPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

WARNING:tensorflow:From c:\Users\HP\anaconda3\lib\site-packages\keras\src
\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprec
ated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```
arxiv_data = pd.read_csv("arxiv_data_210930-054931.csv")
```

In [3]:
```
arxiv_data.head()
```

Out[3]:

	terms	titles	abstracts
0	['cs.LG']	Multi-Level Attention Pooling for Graph Neural...	Graph neural networks (GNNs) have been widely ...
1	['cs.LG', 'cs.AI']	Decision Forests vs. Deep Networks: Conceptual...	Deep networks and decision forests (such as ra...
2	['cs.LG', 'cs.CR', 'stat.ML']	Power up! Robust Graph Convolutional Network v...	Graph convolutional networks (GCNs) are powerf...
3	['cs.LG', 'cs.CR']	Releasing Graph Neural Networks with Different...	With the increasing popularity of Graph Neural...
4	['cs.LG']	Recurrence-Aware Long-Term Cognitive Network f...	Machine learning solutions for pattern classif...

Data Cleaning and Preprocessing

In [4]:
```
arxiv_data.shape
```

Out[4]:
```
(56181, 3)
```

In [5]:
```
arxiv_data.isnull().sum()
```

Out[5]:
```
terms        0
titles       0
abstracts    0
dtype: int64
```

In [6]:
```
arxiv_data.duplicated().sum()
```

Out[6]:
```
15054
```

In [7]:
```
# getting unique labels
labels_column = arxiv_data['terms'].apply(literal_eval)
labels = labels_column.explode().unique()
print("labels :",labels)
print("lenght :",len(labels))
```

```
labels : ['cs.LG' 'cs.AI' 'cs.CR' ... 'D.1.3; G.4; I.2.8; I.2.11; I.5.3;
J.3'
 '68T07, 68T45, 68T10, 68T50, 68U35' 'I.2.0; G.3']
lenght : 1177
```

In [8]:
```
# remove duplicate entries based on the "titles" (terms) column
# This filters the DataFrame, keeping only the rows where the titles are no
arxiv_data = arxiv_data[~arxiv_data['titles'].duplicated()]
print(f"There are {len(arxiv_data)} rows in the deduplicated dataset.")
# There are some terms with occurrence as low as 1.
print(sum(arxiv_data['terms'].value_counts()==1))
# how many unique terms
print(arxiv_data['terms'].nunique())
```

```
There are 41105 rows in the deduplicated dataset.
2503
3401
```

In [9]:
```
# Filtering the rare terms. (it keeps only those rows where the "terms" val
arxiv_data_filtered = arxiv_data.groupby('terms').filter(lambda x: len(x) >
arxiv_data_filtered.shape
```

Out[9]:
```
(38602, 3)
```

In [10]:
```
# It evaluates the given string containing a Python literal or container di
arxiv_data_filtered['terms'] = arxiv_data_filtered['terms'].apply(lambda x:
arxiv_data_filtered['terms'].values[:3]
```

Out[10]:
```
array([list(['cs.LG']), list(['cs.LG', 'cs.AI']),
       list(['cs.LG', 'cs.CR', 'stat.ML'])], dtype=object)
```

train and test split.

In [12]:
```
test_split = 0.1

# Initial train and test split.
# The stratify parameter ensures that the splitting is done in a way that p
train_df, test_df = train_test_split(arxiv_data_filtered,test_size=test_spl

# Splitting the test set further into validation
# and new test sets.
val_df = test_df.sample(frac=0.5)
test_df.drop(val_df.index, inplace=True)

print(f"Number of rows in training set: {len(train_df)}")
print(f"Number of rows in validation set: {len(val_df)}")
print(f"Number of rows in test set: {len(test_df)}")
```

```
Number of rows in training set: 34741
Number of rows in validation set: 1930
Number of rows in test set: 1931
```

In [13]:
```
# creates a TensorFlow RaggedTensor (terms) from the values in the "terms"
terms = tf.ragged.constant(train_df['terms'].values)
# This line creates a StringLookup layer in TensorFlow. The purpose of this
lookup = tf.keras.layers.StringLookup(output_mode='multi_hot')
# This step adapts the StringLookup layer to the unique values in the "term
lookup.adapt(terms)
# retrieve vocabulary
vocab = lookup.get_vocabulary()

print("Vocabulary:\n")
print(vocab)
```

```
WARNING:tensorflow:From c:\Users\HP\anaconda3\lib\site-packages\keras\src
\backend.py:873: The name tf.get_default_graph is deprecated. Please use t
f.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\HP\anaconda3\lib\site-packages\keras\src
\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecate
d. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

Vocabulary:

['[UNK]', 'cs.CV', 'cs.LG', 'stat.ML', 'cs.AI', 'eess.IV', 'cs.RO', 'cs.C
L', 'cs.NE', 'cs.GR', 'cs.CR', 'math.OC', 'eess.SP', 'cs.SI', 'cs.MM', 'c
s.SY', 'cs.IR', 'eess.SY', 'cs.MA', 'cs.HC', 'math.IT', 'cs.IT', 'cs.DC',
'stat.AP', 'cs.CY', 'stat.ME', 'stat.TH', 'math.ST', 'eess.AS', 'cs.SD',
'cs.DS', 'q-bio.QM', 'q-bio.NC', 'cs.CG', 'stat.CO', 'cs.GT', 'cs.NI', 'ma
th.NA', 'cs.SE', 'cs.NA', 'I.2.6', 'physics.chem-ph', 'cs.DB', 'physics.co
mp-ph', 'cond-mat.dis-nn', 'q-bio.BM', 'cs.PL', 'cs.LO', 'math.PR', '68T4
5', 'cs.AR', 'physics.data-an', 'quant-ph', 'I.2.10', 'cs.CE', 'cond-mat.s
tat-mech', 'q-fin.ST', 'math.DS', 'I.4.6', 'physics.ao-ph', 'cs.CC', '68T0
5', 'physics.soc-ph', 'physics.med-ph', 'cs.PF', 'cs.DM', 'q-bio.GN', 'eco
n.EM', 'I.4.8', 'astro-ph.IM', 'physics.flu-dyn', 'math.AT', 'hep-ex', 'I.
4', '68U10', 'q-fin.TR', 'physics.geo-ph', 'cs.FL', 'I.5.4', 'I.2', 'cond-
mat.mtrl-sci', 'I.4.9', '68T10', 'physics.optics', 'I.4; I.5', '68T07', 'q
-fin.CP', 'math.CO', 'math.AP', 'I.2.6; I.2.8', '65D19', 'q-bio.PE', 'phys
ics.app-ph', 'nlin.CD', 'cs.MS', 'I.4.5', 'I.2.6; I.5.1', 'I.2.10; I.4; I.
5', 'I.2.0; I.2.6', '68U01', '68T01', 'q-fin.GN', 'hep-ph', 'cs.SC', 'cs.E
T', 'K.3.2', 'I.2.8', '68T30', 'q-fin.EC', 'q-bio.MN', 'econ.GN', 'I.4.9;
I.5.4', 'I.4.0', 'I.2; I.5', 'I.2; I.4; I.5', 'I.2.6; I.2.7', 'I.2.10; I.
4.8', '68T99', '68Q32', '68', '62H30', 'q-fin.RM', 'q-fin.PM', 'q-bio.TO',
'q-bio.OT', 'physics.plasm-ph', 'physics.class-ph', 'physics.bio-ph', 'nli
n.AO', 'math.SP', 'math.MP', 'math.LO', 'math.FA', 'math-ph', 'cs.DL', 'co
nd-mat.soft', 'I.5.2', 'I.4.6; I.4.8', 'I.4.4', 'I.4.3', 'I.4.1', 'I.3.7',
'I.2; J.2', 'I.2; I.2.6; I.2.7', 'I.2.7', 'I.2.6; I.5.4', 'I.2.6; I.2.9',
'I.2.6; I.2.7; H.3.1; H.3.3', 'I.2.6; I.2.10', 'I.2.6, I.5.4', 'I.2.1; J.
3', 'I.2.10; I.5.1; I.4.8', 'I.2.10; I.4.8; I.5.4', 'I.2.10; I.2.6', 'I.2.
1', 'H.3.1; I.2.6; I.2.7', 'H.3.1; H.3.3; I.2.6; I.2.7', 'G.3', 'F.2.2; I.
2.7', 'E.5; E.4; E.2; H.1.1; F.1.1; F.1.3', '68Txx', '62H99', '62H35', '60
L10, 60L20', '14J60 (Primary) 14F05, 14J26 (Secondary)']
```

In [14]:
```
sample_label = train_df["terms"].iloc[0]
print(f"Original label: {sample_label}")

label_binarized = lookup([sample_label])
print(f"Label-binarized representation: {label_binarized}")
```

```
Original label: ['cs.CV']
Label-binarized representation: [[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

In [15]:
```
# following lines::
# which is used for automatic adjustment of resource usage by TensorFlow's

#max_seqlen: Maximum sequence length. It indicates the maximum length allow
max_seqlen = 150
#batch_size: Batch size. It specifies the number of samples to use in each
batch_size = 128
#padding_token: A token used for padding sequences.
padding_token = "<pad>"
#auto = tf.data.AUTOTUNE: auto is assigned the value tf.data.AUTOTUNE,
auto = tf.data.AUTOTUNE

def make_dataset(dataframe, is_train=True):
    # creating sequences of labesls
    labels = tf.ragged.constant(dataframe["terms"].values)
    #This line uses the previously defined lookup layer to convert the ragg
    label_binarized = lookup(labels).numpy()
    # creating sequences of text.
    dataset = tf.data.Dataset.from_tensor_slices((dataframe["abstracts"].va
    # shuffling data basis on condition
    dataset = dataset.shuffle(batch_size * 10) if is_train else dataset
    return dataset.batch(batch_size)

"""
In summary, the make_dataset function is designed to create a
dataset suitable for training a model. It takes a dataframe as input,
assumes it has "abstracts" and "terms" columns, and creates a dataset of
batches where each batch consists of abstract
def invert_multi_hot(encoded_labels):
    """Reverse a single multi-hot encoded label to a tuple of vocab terms."
    hot_indices = np.argwhere(encoded_labels == 1.0)[..., 0]
    return np.take(loaded_vocab, hot_indices)sequences and their correspond
"""
```

Out[15]:
```
'\nIn summary, the make_dataset function is designed to create a \ndataset
suitable for training a model. It takes a dataframe as input, \nassumes it
has "abstracts" and "terms" columns, and creates a dataset of \nbatches wh
ere each batch consists of abstract \nsequences and their corresponding bi
narized label sequences. \n'
```

In [16]:
```
train_dataset = make_dataset(train_df, is_train=True)
validation_dataset = make_dataset(val_df, is_train=False)
test_dataset = make_dataset(test_df, is_train=False)
```

In [19]:
```
def invert_multi_hot(encoded_labels):
    """Reverse a single multi-hot encoded label to a tuple of vocab terms."
    hot_indices = np.argwhere(encoded_labels == 1.0)[..., 0]
    return np.take(vocab, hot_indices)


# This code snippet is iterating through batches of the training dataset an
text_batch, label_batch = next(iter(train_dataset))
for i, text in enumerate(text_batch[:5]):
    label = label_batch[i].numpy()[None, ...]
    print(f"Abstract: {text}")
    print(f"Label(s): {invert_multi_hot(label[0])}")
    print(" ")
```

```
Abstract: b'In this work, we present the Text Conditioned Auxiliary Classi
fier Generative\nAdversarial Network, (TAC-GAN) a text to image Generative
Adversarial Network\n(GAN) for synthesizing images from their text descrip
tions. Former approaches\nhave tried to condition the generative process o
n the textual data; but allying\nit to the usage of class information, kno
wn to diversify the generated samples\nand improve their structural cohere
nce, has not been explored. We trained the\npresented TAC-GAN model on the
Oxford-102 dataset of flowers, and evaluated the\ndiscriminability of the
generated images with Inception-Score, as well as their\ndiversity using t
he Multi-Scale Structural Similarity Index (MS-SSIM). Our\napproach outper
forms the state-of-the-art models, i.e., its inception score is\n3.45, cor
responding to a relative increase of 7.8% compared to the recently\nintrod
uced StackGan. A comparison of the mean MS-SSIM scores of the training\nan
d generated samples per class shows that our approach is able to generate
\nhighly diverse images with an average MS-SSIM of 0.14 over all generated
\nclasses.'
Label(s): ['cs.CV']

Abstract: b'Similarity learning has gained a lot of attention from researc
hes in recent\nyears and tons of successful approaches have been recently
proposed. However,\nthe majority of the state-of-the-art similarity learni
ng methods consider only\na binary similarity. In this paper we introduce
a new loss function called\nContinuous Histogram Loss (CHL) which generali
zes recently proposed Histogram\nloss to multiple-valued similarities, i.
e. allowing the acceptable values of\nsimilarity to be continuously distri
buted within some range. The novel loss\nfunction is computed by aggregati
ng pairwise distances and similarities into 2D\nhistograms in a differenti
able manner and then computing the probability of\ncondition that pairwise
distances will not decrease as the similarities\nincrease. The novel loss
is capable of solving a wider range of tasks including\nsimilarity learnin
g, representation learning and data visualization.'
Label(s): ['cs.LG' 'stat.ML']

Abstract: b'We propose to restore old photos that suffer from severe degra
dation through\na deep learning approach. Unlike conventional restoration
tasks that can be\nsolved through supervised learning, the degradation in
real photos is complex\nand the domain gap between synthetic images and re
al old photos makes the\nnetwork fail to generalize. Therefore, we propose
a novel triplet domain\ntranslation network by leveraging real photos alon
g with massive synthetic\nimage pairs. Specifically, we train two variatio
nal autoencoders (VAEs) to\nrespectively transform old photos and clean ph
otos into two latent spaces. And\nthe translation between these two latent
spaces is learned with synthetic\npaired data. This translation generalize
s well to real photos because the\ndomain gap is closed in the compact lat
ent space. Besides, to address multiple\ndegradations mixed in one old pho
to, we design a global branch with a partial\nnonlocal block targeting to
the structured defects, such as scratches and dust\nspots, and a local bra
nch targeting to the unstructured defects, such as noises\nand blurriness.
Two branches are fused in the latent space, leading to improved\ncapabilit
y to restore old photos from multiple defects. The proposed method\noutper
forms state-of-the-art methods in terms of visual quality for old photos\n
restoration.'
Label(s): ['cs.CV' 'eess.IV' 'cs.GR']

Abstract: b'In this study, we present a multi-class graphical Bayesian pre
dictive\nclassifier that incorporates the uncertainty in the model selecti
on into the\nstandard Bayesian formalism. For each class, the dependence s
tructure\nunderlying the observed features is represented by a set of deco
mposable\nGaussian graphical models. Emphasis is then placed on the Bayesi
an model\naveraging which takes full account of the class-specific model u
ncertainty by\naveraging over the posterior graph model probabilities. An
```

```
explicit evaluation\nof the model probabilities is well known to be infeas
ible. To address this\nissue, we consider the particle Gibbs strategy of O
lsson et al. (2018b) for\nposterior sampling from decomposable graphical m
odels which utilizes the\nChristmas tree algorithm of Olsson et al. (2018
a) as proposal kernel. We also\nderive a strong hyper Markov law which we
call the hyper normal Wishart law\nthat allow to perform the resultant Bay
esian calculations locally. The proposed\npredictive graphical classifier
reveals superior performance compared to the\nordinary Bayesian predictive
rule that does not account for the model\nuncertainty, as well as to a num
ber of out-of-the-box classifiers.'
Label(s): ['stat.ML']

Abstract: b'We present a learning model that makes full use of boundary in
formation for\nsalient object segmentation. Specifically, we come up with
a novel loss\nfunction, i.e., Contour Loss, which leverages object contour
s to guide models\nto perceive salient object boundaries. Such a boundary-
aware network can learn\nboundary-wise distinctions between salient object
s and background, hence\neffectively facilitating the saliency detection.
Yet the Contour Loss\nemphasizes on the local saliency. We further propose
the hierarchical global\nattention module (HGAM), which forces the model h
ierarchically attend to global\ncontexts, thus captures the global visual
saliency. Comprehensive experiments\non six benchmark datasets show that o
ur method achieves superior performance\nover state-of-the-art ones. Moreo
ver, our model has a real-time speed of 26 fps\non a TITAN X GPU.'
Label(s): ['cs.CV' '65D19']
```

In [20]:

```
# This code calculates the size of the vocabulary in the "abstracts" column

# Creating vocabulary with uniques words
vocabulary = set()
train_df["abstracts"].str.lower().str.split().apply(vocabulary.update)
vocabulary_size = len(vocabulary)
print(vocabulary_size)
```

```
159227
```

Text Vectorization

In [21]:

```
# Initializes a TextVectorization layer
text_vectorizer = layers.TextVectorization(max_tokens=vocabulary_size,ngram
# `TextVectorization` layer needs to be adapted as per the vocabulary from
# training set.
text_vectorizer.adapt(train_dataset.map(lambda text, label: text))
```

In [22]:
```
"""
Mapping Vectorization to Datasets: The code maps the text vectorization ope
each element of the training, validation, and test datasets. This ensures t
data in each dataset is transformed into numerical vectors using the adapte
The num_parallel_calls parameter is used to parallelize the mapping process
applied to prefetch data batches
for better performance.
"""
train_dataset = train_dataset.map(lambda text, label: (text_vectorizer(text
validation_dataset = validation_dataset.map(lambda text, label: (text_vecto
test_dataset = test_dataset.map(lambda text, label: (text_vectorizer(text),
```

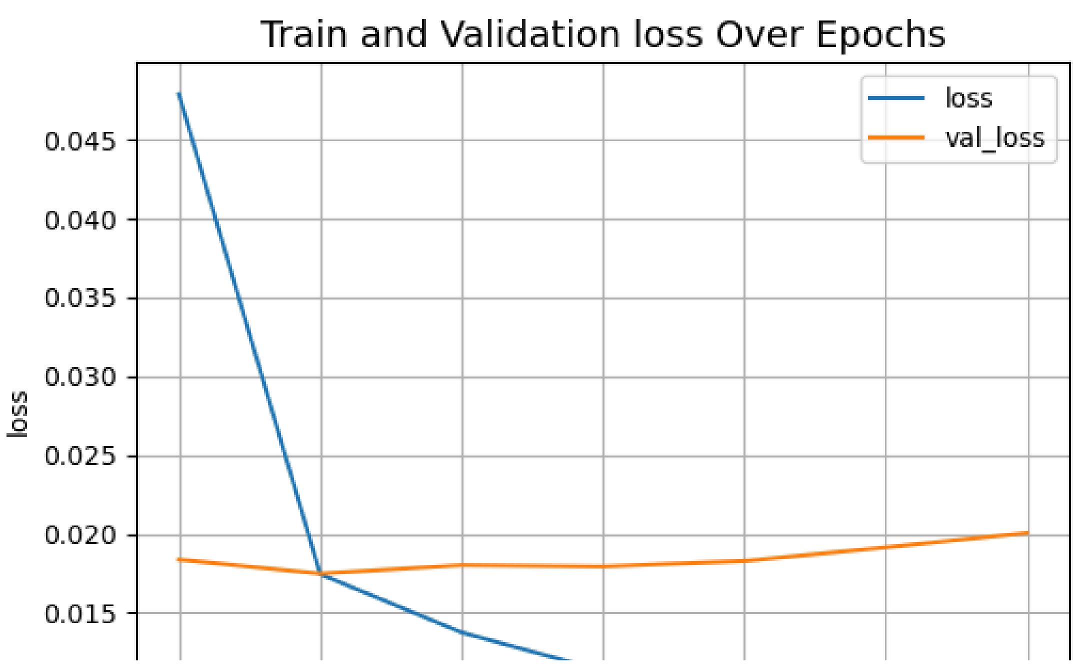
model training

In [26]:

```
# creating shallow_mlp_model  (MLP)
from tensorflow.keras.callbacks import EarlyStopping

# Creating shallow_mlp_model (MLP) with dropout layers
model1 = keras.Sequential([
    # First hidden layer: 512 neurons, ReLU activation function, with dropo
    layers.Dense(512, activation="relu"),
    layers.Dropout(0.5),  # Adding dropout for regularization.

    # Second hidden layer: 256 neurons, ReLU activation function, with drop
    layers.Dense(256, activation="relu"),
    layers.Dropout(0.5),  # Adding dropout for regularization.

    # Output layer: The number of neurons equals the vocabulary size (outpu
    layers.Dense(lookup.vocabulary_size(), activation='sigmoid')
])

# Compile the model
model1.compile(loss="binary_crossentropy", optimizer='adam', metrics=['bina

# Add early stopping
# Number of epochs with no improvement after which training will be stopped
# Restore weights from the epoch with the best value of the monitored quant
early_stopping = EarlyStopping(patience=5,restore_best_weights=True)

# Train the model
# Add early stopping callback.verbose=1
history = model1.fit(train_dataset,validation_data=validation_dataset,epoch
```

```
Epoch 1/20
272/272 [==============================] - 281s 998ms/step - loss: 0.04
79 - binary_accuracy: 0.9836 - val_loss: 0.0184 - val_binary_accuracy:
0.9946
Epoch 2/20
272/272 [==============================] - 279s 1s/step - loss: 0.0174
- binary_accuracy: 0.9950 - val_loss: 0.0175 - val_binary_accuracy: 0.9
949
Epoch 3/20
272/272 [==============================] - 288s 1s/step - loss: 0.0138
- binary_accuracy: 0.9959 - val_loss: 0.0180 - val_binary_accuracy: 0.9
948
Epoch 4/20
272/272 [==============================] - 292s 1s/step - loss: 0.0113
- binary_accuracy: 0.9966 - val_loss: 0.0179 - val_binary_accuracy: 0.9
947
Epoch 5/20
272/272 [==============================] - 312s 1s/step - loss: 0.0097
- binary_accuracy: 0.9971 - val_loss: 0.0183 - val_binary_accuracy: 0.9
```

In [27]:
```
# plotting loss
def plot_result(item):
    plt.plot(history.history[item], label=item)
    plt.plot(history.history["val_" + item], label="val_" + item)
    plt.xlabel("Epochs")
    plt.ylabel(item)
    plt.title("Train and Validation {} Over Epochs".format(item), fontsize=
    plt.legend()
    plt.grid()
    plt.show()


plot_result("loss")
plot_result("binary_accuracy")
```



Model Evaluation

In [28]:
```
# model evaltuation on test and val dataset
_, binary_acc1 = model1.evaluate(test_dataset)
_, binary_acc2 = model1.evaluate(validation_dataset)

print(f"Categorical accuracy on the test set: {round(binary_acc1 * 100, 2)}
print(f"Categorical accuracy on the validation set: {round(binary_acc2 * 10
```

```
16/16 [==============================] - 4s 189ms/step - loss: 0.0182 - bi
nary_accuracy: 0.9946
16/16 [==============================] - 3s 159ms/step - loss: 0.0175 - bi
nary_accuracy: 0.9949
Categorical accuracy on the test set: 99.46%.
Categorical accuracy on the validation set: 99.49%.
```

Save Model and Text Vectorizer:

In [30]:
```
import pickle

# Save the model
model1.save("models/model.h5")

# Save the configuration of the text vectorizer
saved_text_vectorizer_config = text_vectorizer.get_config()
with open("models/text_vectorizer_config.pkl", "wb") as f:
    pickle.dump(saved_text_vectorizer_config, f)


# Save the vocabulary
with open("models/vocab.pkl", "wb") as f:
    pickle.dump(vocab, f)
```

```
c:\Users\HP\anaconda3\lib\site-packages\keras\src\engine\training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Load Model and Text Vectorizer:

In [37]:
```
# from tensorflow import keras
# import pickle
import pickle
# Load the model
loaded_model = keras.models.load_model("models/model.h5")

# from tensorflow.keras.layers import TextVectorization

# Load the configuration of the text vectorizer
with open("text_vectorizer_config.pkl", "rb") as f:
    saved_text_vectorizer_config = pickle.load(f)

# Create a new TextVectorization layer with the saved configuration
loaded_text_vectorizer = text_vectorizer.from_config(saved_text_vectorizer_

# Load the saved weights into the new TextVectorization layer
with open("text_vectorizer_weights.pkl", "rb") as f:
    weights = pickle.load(f)
    loaded_text_vectorizer.set_weights(weights)

with open("vocab.pkl","rb") as f:
    loaded_vocab = pickle.load(f)
```

In [38]:
```
# Load the vocabulary
with open("models/vocab.pkl", "rb") as f:
    loaded_vocab = pickle.load(f)
```

In [39]:
```
def invert_multi_hot(encoded_labels):
    """Reverse a single multi-hot encoded label to a tuple of vocab terms."
    hot_indices = np.argwhere(encoded_labels == 1.0)[..., 0]
    return np.take(loaded_vocab, hot_indices)
```

In [40]:
```
def predict_category(abstract, model, vectorizer, label_lookup):
    # Preprocess the abstract using the loaded text vectorizer
    preprocessed_abstract = vectorizer([abstract])

    # Make predictions using the loaded model
    predictions = model.predict(preprocessed_abstract)

    # Convert predictions to human-readable labels
    predicted_labels = label_lookup(np.round(predictions).astype(int)[0])

    return predicted_labels
```

In [45]:
```
# Example usage
new_abstract = 'Deep networks and decision forests (such as random forests
predicted_categories = predict_category(new_abstract, loaded_model, loaded_
print("Predicted Categories:", predicted_categories)
```

```
1/1 [==============================] - 0s 459ms/step
Predicted Categories: ['cs.LG' 'cs.AI']
```

In [39]:
```
# great resutls.................................
```

=======Section 2========

2 Recommendation System

In [43]:
```
arxiv_data.drop(columns = ["terms","abstracts"], inplace = True)
```

In [44]:
```
arxiv_data.drop_duplicates(inplace= True)
arxiv_data.reset_index(drop= True,inplace = True)
```

In [45]:
```
pd.set_option('display.max_colwidth', None)
arxiv_data
```

Out[45]:

	titles
0	Multi-Level Attention Pooling for Graph Neural Networks: Unifying Graph Representations with Multiple Localities
1	Decision Forests vs. Deep Networks: Conceptual Similarities and Empirical Differences at Small Sample Sizes
2	Power up! Robust Graph Convolutional Network via Graph Powering
3	Releasing Graph Neural Networks with Differential Privacy Guarantees
4	Recurrence-Aware Long-Term Cognitive Network for Explainable Pattern Classification
...	...
41100	An experimental study of graph-based semi-supervised classification with additional node information
41101	Bayesian Differential Privacy through Posterior Sampling
41102	Mining Spatio-temporal Data on Industrialization from Historical Registries
41103	Wav2Letter: an End-to-End ConvNet-based Speech Recognition System
41104	Generalized Low Rank Models

41105 rows × 1 columns

Sentence Transformers

In [47]:
```
!pip install -U -q sentence-transformers
```

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000020D095B3A60>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/sentence-transformers/
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000020D095B3D90>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/sentence-transformers/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000020D095B3DC0>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/sentence-transformers/
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000020D095E3280>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/sentence-transformers/
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000020D095E3430>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/sentence-transformers/
ERROR: Could not find a version that satisfies the requirement sentence-tr
ansformers (from versions: none)
ERROR: No matching distribution found for sentence-transformers
```

In [49]:
```
# This imports the SentenceTransformer class from the Sentence Transformers
from sentence_transformers import SentenceTransformer, util
# we load all-MiniLM-L6-v2, which is a MiniLM model fine tuned on a large d
# 1 billion training pairs.
#This initializes the 'all-MiniLM-L6-v2' model from Sentence Transformers.
# This model is capable of encoding sentences into fixed-size vectors (embe
model = SentenceTransformer('all-MiniLM-L6-v2')
#Our sentences we like to encode
sentences = arxiv_data['titles']
#Sentences are encoded by calling model.encode()
embeddings = model.encode(sentences)

"""
The embeddings can be used for various natural language processing (NLP) ta
such as similarity search, clustering
"""
```

```
modules.json:   0%|          | 0.00/349 [00:00<?, ?B/s]

c:\Users\HP\anaconda3\lib\site-packages\huggingface_hub\file_download.py:1
48: UserWarning: `huggingface_hub` cache-system uses symlinks by default t
o efficiently store duplicated files but your machine does not support the
m in C:\Users\HP\.cache\huggingface\hub\models--sentence-transformers--all
-MiniLM-L6-v2. Caching files will still work but in a degraded version tha
t might require more space on your disk. This warning can be disabled by s
etting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For mor
e details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#li
mitations. (https://huggingface.co/docs/huggingface_hub/how-to-cache#limit
ations.)
To support symlinks on Windows, you either need to activate Developer Mode
or to run Python as an administrator. In order to see activate developer m
ode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-s
tarted/enable-your-device-for-development (https://docs.microsoft.com/en-u
s/windows/apps/get-started/enable-your-device-for-development)
  warnings.warn(message)

config_sentence_transformers.json:   0%|          | 0.00/116 [00:00<?, ?B/
s]

README.md:   0%|          | 0.00/10.7k [00:00<?, ?B/s]

sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/612 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/90.9M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/350 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]

1_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
```

Out[49]:
```
'\nThe embeddings can be used for various natural language processing (NL
P) tasks, \nsuch as similarity search, clustering\n'
```

In [50]:
```
embeddings
```

Out[50]:
```
array([[ 0.06643412, -0.04954597,  0.06388088, ...,  0.00106303,
        -0.1215638 , -0.06962783],
       [ 0.09212257, -0.07606944,  0.06572863, ..., -0.08565164,
        -0.09266546,  0.00725291],
       [-0.08162688,  0.02428937,  0.01888741, ...,  0.00806161,
        -0.05129534, -0.05874001],
       ...,
       [ 0.01227978, -0.08568835, -0.02782774, ..., -0.05257973,
        -0.10806682,  0.07843313],
       [-0.07258195, -0.12690923, -0.0053555 , ...,  0.03597708,
        -0.03986151, -0.05971032],
       [ 0.00768869, -0.10124185,  0.08909854, ..., -0.08199864,
        -0.05649742,  0.09007055]], dtype=float32)
```

Type *Markdown* and LaTeX: α^2

Print the embeddings

In [51]:
```
c = 0
#This loop iterates over pairs of sentences and their corresponding embeddi
#zip is used to iterate over both lists simultaneously.
for sentence, embedding in zip(sentences, embeddings):
    print("Sentence:", sentence)
    print("Embedding length:", len(embedding)) # list of floats
    print("")
    # Breaks out of the loop after printing information for the first 5 sen
    if c >=5:
        break
    c +=1
```

```
Sentence: Multi-Level Attention Pooling for Graph Neural Networks: Unifyin
g Graph Representations with Multiple Localities
Embedding length: 384

Sentence: Decision Forests vs. Deep Networks: Conceptual Similarities and
Empirical Differences at Small Sample Sizes
Embedding length: 384

Sentence: Power up! Robust Graph Convolutional Network via Graph Powering
Embedding length: 384

Sentence: Releasing Graph Neural Networks with Differential Privacy Guaran
tees
Embedding length: 384

Sentence: Recurrence-Aware Long-Term Cognitive Network for Explainable Pat
tern Classification
Embedding length: 384

Sentence: Lifelong Graph Learning
Embedding length: 384
```

Save files

In [52]:
```
import pickle
# Saving sentences and corresponding embeddings
with open('embeddings.pkl', 'wb') as f:
    pickle.dump(embeddings, f)

with open('sentences.pkl', 'wb') as f:
    pickle.dump(sentences, f)

with open('rec_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

Recommendation for similar papers

In [54]:
```
# load save files
embeddings = pickle.load(open('embeddings.pkl','rb'))
sentences = pickle.load(open('sentences.pkl','rb'))
rec_model = pickle.load(open('rec_model.pkl','rb'))
```

In [55]:
```
import torch

def recommendation(input_paper):
    # Calculate cosine similarity scores between the embeddings of input_pa
    cosine_scores = util.cos_sim(embeddings, rec_model.encode(input_paper))

    # Get the indices of the top-k most similar papers based on cosine simi
    top_similar_papers = torch.topk(cosine_scores, dim=0, k=5, sorted=True)

    # Retrieve the titles of the top similar papers.
    papers_list = []
    for i in top_similar_papers.indices:
        papers_list.append(sentences[i.item()])

    return papers_list
```

In [ ]:
```
# exampel usage 2: (use this paper as input (BERT: Pre-training of Deep Bid
input_paper = input("Enter the title of any paper you like")
recommend_papers = recommendation(input_paper)


print("We recommend to read this paper............")
print("=============================================")
for paper in recommend_papers:
    print(paper)
```

In [ ]:

```
# exampel usage 3: (use this paper as input (Review of deep learning: conce
input_paper = input("Enter the title of any paper you like")
recommend_papers = recommendation(input_paper)


print("We recommend to read this paper............")
print("=============================================")
for paper in recommend_papers:
    print(paper)
```

In [57]:

```
# install this versions
import sentence_transformers
import tensorflow
import torch
print(torch.__version__)
print(sentence_transformers.__version__)
print(tensorflow.__version__)
```

```
2.2.2+cpu
2.7.0
2.15.0
```

In [ ]: