Deep Learning
(CSL 312)

Lab Practical Report

Faculty name: Ms. Srishti Vashishtha

Student Name: Chayan Gulati

Roll No.: 18csu054

Semester: 6th

Group: DS VI- A2

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2020-21

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India

Session 2020-21

INDEX

8.	Image Classification on MNSIT Dataset	39	24-02-2021		2	
9.	To Create a CNN model with dataset containing images of cats and dogs for image classification	47	3-03-2021		2	
10.	To Create Le-Net model on MNSIT Dataset	50	10-03-2021		2	
11.	To Create Alex-Net model on CIFAR10 Dataset	53	17-03-2021		2	
12.	To Build an image classifier with Keras and Convolutional Neural Networks for the fashion MNIST dataset	57	17-03-2021		2	
13.	To Train a CNN model to classify images from CIFAR10 Database.	60	31-03-2021		2	
14.	To implement autoencoders for dimensionality reduction	63	28-04-2021		3	
15.	Using MNIST dataset, improve autoencoder's performance using convolutional layers	68	28-04-2021		3	
16.	To create a recurrent neural model on alcohol sales dataset	72	5-05-2021		3	
17.	To create a RNN model and predict miles travelled by vehicles	83	5-05-2021		3	

18.	To perform tokenisation and stemming on text data using NLTK	92	7-04-2021		4	
19.	To perform lemmatization and remove stopwords on text data using NLTK	95	7-04-2021		4	
20.	To explore following components of Partof-Speech (POS) tagging: a) Extract nouns from text b) Extract verbs from text	102	7-04-2021		4	
21.	To create Bag-ofWords(BOW) and Bag-og-n-grams using the following a) Bag-of-words Using the Count Vectorizer b)Bag-of-n-grams Using the Count Vectorizer c) Bag-of-words Using the Tf-Idf Vectorizer	108	14-04-2021		4	
22.	To build a NLP model for Spam Detection using TFIDF Vectorizer.	120	14-04-2021		4	

EXPERIMENT NO. 1

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 27-1-2021
Faculty Signature:
Marks:

Objective(s): Exploring Features of Tensorflow and Keras packages.
Outcome: Getting Familiar with some of the deep learning Packages.
Problem Statement: To explore the basic features of Tensorflow and Keras packages.
Background Study: TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms, Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

Student Work Area

Code/Sample Outputs:

```python
# Importing the libraries
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13]
y = dataset.iloc[:, -1]

X = X.values
y = y.values

```

```python
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])


from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("Country", OneHotEncoder(),[1])], remainder="passthrough") # The last arg ([0]) is the list of colu
X = ct.fit_transform(X)
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

```python
# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import tensorflow as tf
print ("TensorFlow version: " + tf.__version__)



from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer (2 layers are added by this one function call)
# units = 6 comes from average of input layer + output layer (11 + 1)/2
classifier.add(Dense(input_dim = 11, units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
# For a logistic regression, we dont use the sum of squared residual loss function
# We use binary cross entropy if we have 1 binary outcome and
# categorical cross entropy if we have more than one categorical outcome
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

```

```
800/800 [==============================] - 1s 1ms/step - loss: 0.3920 - accuracy: 0.8405
Epoch 94/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3912 - accuracy: 0.8411
Epoch 95/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3921 - accuracy: 0.8397
Epoch 96/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3916 - accuracy: 0.8429
Epoch 97/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3920 - accuracy: 0.8394
Epoch 98/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3918 - accuracy: 0.8406
Epoch 99/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3919 - accuracy: 0.8415
Epoch 100/100
800/800 [==============================] - 1s 907us/step - loss: 0.3915 - accuracy: 0.8401

<tensorflow.python.keras.callbacks.History at 0x154466b73c8>
```

```
# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

```

```
cm
```

```
array([[1492,   80],
       [ 287,  141]], dtype=int64)
```

EXPERIMENT NO. 2

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 27-1-2021
Faculty Signature:
Marks:

Objective(s): Exploring Features of Tensorflow 2.0
Outcome: Learning how different features effects the training of a model.
Problem Statement: To explore the features of TensorFlow 2.0 using Fashion MNIST data.
Background Study: There are multiple changes in TensorFlow 2.0 to make TensorFlow users more productive. TensorFlow 2.0 removes redundant APIs, makes APIs more consistent (Unified RNNs, Unified Optimizers), and better integrates with the Python runtime with Eager execution Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct **dropin replacement** for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Student Work Area

Code/Sample Outputs:

Import the relevant packages

```
import numpy as np
import tensorflow as tf

import tensorflow_datasets as tfds
```

Data

That's where we load and preprocess our data.

```
mnist_dataset, mnist_info = tfds.load(name='mnist', with_info=True, as_supervised=True)
mnist_train, mnist_test = mnist_dataset['train'], mnist_dataset['test']

num_validation_samples = 0.1 * mnist_info.splits['train'].num_examples
num_validation_samples = tf.cast(num_validation_samples, tf.int64)

num_test_samples = tf.cast(num_test_samples, tf.int64)

def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255.

    return image, label

scaled_train_and_validation_data = mnist_train.map(scale)

test_data = mnist_test.map(scale)


BUFFER_SIZE = 10000
shuffled_train_and_validation_data = scaled_train_and_validation_data.shuffle(BUFFER_SIZE)

validation_data = shuffled_train_and_validation_data.take(num_validation_samples)

train_data = shuffled_train_and_validation_data.skip(num_validation_samples)

BATCH_SIZE = 100

train_data = train_data.batch(BATCH_SIZE)

validation_data = validation_data.batch(num_validation_samples)

test_data = test_data.batch(num_test_samples)

validation_inputs, validation_targets = next(iter(validation_data))
```

Model

Outline the model

When thinking about a deep learning algorithm, we mostly imagine building the model. So, let's do it :)

```
input_size = 784
output_size = 10
hidden_layer_size = 205

model = tf.keras.Sequential([

    tf.keras.layers.Flatten(input_shape=(28, 28, 1)), # input layer

    tf.keras.layers.Dense(hidden_layer_size, activation='relu'), # 1st hidden layer
    tf.keras.layers.Dense(hidden_layer_size, activation='relu'), # 2nd hidden layer

    tf.keras.layers.Dense(output_size, activation='softmax') # output layer
])
```

Choose the optimizer and the loss function

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Training

That's where we train the model we have built.

```
NUM_EPOCHS = 5

model.fit(train_data, epochs=NUM_EPOCHS, validation_data=(validation_inputs, validation_targets), validation_steps=1, verl
```

```
Epoch 1/5
537/540 [============================>.] - ETA: 0s - loss: 0.4988 - accuracy: 0.8579WARNING:tensorflow:5 out of the last 13 ca
ls to <function Model.make_test_function.<locals>.test_function at 0x7f1d7a1540d0> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing ten
ors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of
he loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessa
y retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
w.org/api_docs/python/tf/function for  more details.

WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_test_function.<locals>.test_function at 0x7f1d7a1540d0>
riggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.fun
tion repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1),
lease define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relax
s argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#con
rolling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.

540/540 [==============================] - 10s 15ms/step - loss: 0.4971 - accuracy: 0.8584 - val_loss: 0.1399 - val_accuracy:
0.9575
Epoch 2/5
540/540 [==============================] - 7s 12ms/step - loss: 0.1152 - accuracy: 0.9643 - val_loss: 0.0956 - val_accuracy: 0
9727
Epoch 3/5
540/540 [==============================] - 6s 11ms/step - loss: 0.0732 - accuracy: 0.9779 - val_loss: 0.0757 - val_accuracy: 0
9777
Epoch 4/5
540/540 [==============================] - 5s 9ms/step - loss: 0.0561 - accuracy: 0.9829 - val_loss: 0.0559 - val_accuracy: 0.
833
Epoch 5/5
540/540 [==============================] - 6s 9ms/step - loss: 0.0457 - accuracy: 0.9858 - val_loss: 0.0474 - val_accuracy: 0.
```

Test the model

As we discussed in the lectures, after training on the training data and validating on the validation data, we test the final prediction power of our model by running it on the test dataset that the algorithm has NEVER seen before.

It is very important to realize that fiddling with the hyperparameters overfits the validation dataset.

The test is the absolute final instance. You should not test before you are completely done with adjusting your model.

If you adjust your model after testing, you will start overfitting the test dataset, which will defeat its purpose.

```
test_loss, test_accuracy = model.evaluate(test_data)
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.0738 - accuracy: 0.9771
```

```
# We can apply some nice formatting if we want to
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accuracy*100.))
```

```
Test loss: 0.07. Test accuracy: 97.71%
```

Using the initial model and hyperparameters given in this notebook, the final test accuracy should be roughly around 97%.

Each time the code is rerun, we get a different accuracy as the batches are shuffled, the weights are initialized in a different way, etc.

Finally, we have intentionally reached a suboptimal solution, so you can have space to build on it.

Results after Tuning Hyperparameters:

1. Changing the hidden layer to 200 gives the validation accuracy of 98.5 and it took 30 sec to train the algorithm.
2. After adding another layer doesn't change the accuracy that much it goes down to 98.4% and it took 32 sec to train the algorithm.
3. Even after adding 5 more layers the accuracy doesn't change that much it goes to 98.7% and it took 33 sec to train the algorithm.
4. After applying sigmoid transformation to both layers. The Accuracy goes down to 96.9% and it took 33 sec to train.
5. After applying a ReLu to the first hidden layer and tanh to the second one. the accuracy increases to 98.9% and it took 31 sec to train.
6. After Changing the batch size to 10000. it took 20 sec to train and gives us the accuracy of 90.7%.
7. Changingbatch size to 1 gives the accuracy of 97.5% and take 9 min and 35 sec to train.
8. Changing the learning rate to 0.0001 gives us the accuracy of 95.7% and took 33 seconds.
9. Changing the learning rate to 0.02 gives us the accuracy of 95.4% and took 26 seconds.
10. Taking hidden layer size of 205, applying a ReLu to the first hidden layer and tanh to the second one, using default learning rate gives us the accuracy 98.9 taking to 31 sec rain.

EXPERIMENT NO. 3

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 3-2-2021
Faculty Signature:
Marks:

Objective(s): Implement Ann Model.
Outcome: Learning to implement simple ANN Model.
Problem Statement: To build an ANN Model to convert temperature in degree Celsius to Fahrenheit.
Background Study: Artificial **Neural Networks** (**ANN**) are multi-layer fully-connected neural nets that look like the figure below. They consist of an input layer, multiple hidden layers, and an output layer. Training this **deep neural network** means **learning** the weights associated with all the edges

Student Work Area

Code/Sample Outputs:

IMPORT LIBRARIES

```
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

IMPORT DATASETS

```
Temperature_df = pd.read_csv('Celsius to Fahrenheit.csv')
Temperature_df.reset_index(drop=True, inplace=True)
```

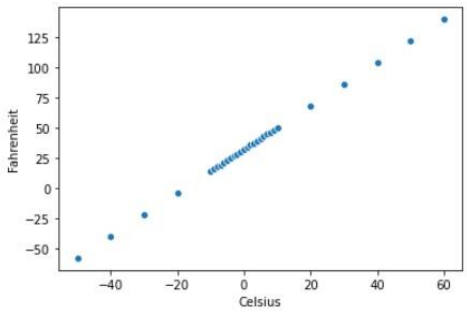
```
Temperature_df
```

	Celsius	Fahrenheit
0	-50	-58.0
1	-40	-40.0
2	-30	-22.0
3	-20	-4.0
4	-10	14.0
5	-9	15.8
6	-8	17.6
7	-7	19.4
8	-6	21.2
9	-5	23.0
10	-4	24.8
11	-3	26.6
12	-2	28.4

VISUALIZE DATASET

```
sns.scatterplot(Temperature_df['Celsius'], Temperature_df['Fahrenheit'])
```

```
<AxesSubplot:xlabel='Celsius', ylabel='Fahrenheit'>
```



CREATE TESTING AND TRAINING DATASET

```
X_train = Temperature_df['Celsius']
y_train = Temperature_df['Fahrenheit']
```

BUILD AND TRAIN THE MODEL

```
X_train.shape
```

```
(30,)
```

```
y_train.shape
```

```
(30,)
```

This will model a simple linear equation.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(units=1, input_shape=[1]))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.5), loss='mean_squared_error')
```

```
epochs_hist = model.fit(X_train, y_train, epochs = 100)
```

```
1/1 [==============================] - 0s 998us/step - loss: 4.0202
Epoch 92/100
1/1 [==============================] - 0s 0s/step - loss: 3.6177
Epoch 93/100
1/1 [==============================] - 0s 996us/step - loss: 3.2457
Epoch 94/100
1/1 [==============================] - 0s 998us/step - loss: 2.9099
Epoch 95/100
1/1 [==============================] - 0s 993us/step - loss: 2.6088
Epoch 96/100
1/1 [==============================] - 0s 996us/step - loss: 2.3331
Epoch 97/100
1/1 [==============================] - 0s 1ms/step - loss: 2.0792
Epoch 98/100
1/1 [==============================] - 0s 999us/step - loss: 1.8508
Epoch 99/100
1/1 [==============================] - 0s 998us/step - loss: 1.6475
Epoch 100/100
1/1 [==============================] - 0s 1ms/step - loss: 1.4630
```

EVALUATING THE MODEL

```
epochs_hist.history.keys()
```

```
dict_keys(['loss'])
```

```
plt.plot(epochs_hist.history['loss'])
plt.title('Model Loss Progress During Training')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.legend(['Training Loss'])
```

```
<matplotlib.legend.Legend at 0x14958af9710>
```


```
model.get_weights()
```

```
[array([[1.8059913]], dtype=float32), array([30.85845], dtype=float32)]
```

```
# Use the trained model to perform predictions

Temp_C = 0
Temp_F = model.predict([Temp_C])
print('Temperature in degF Using Trained ANN =', Temp_F)

```

```
Temperature in degF Using Trained ANN = [[30.85845]]
```

```
# Let's confirm this Using the equation:
Temp_F = 9/5 * Temp_C + 32
print('Temperature in degF Using Equation =', Temp_F)

```

```
Temperature in degF Using Equation = 32.0
```

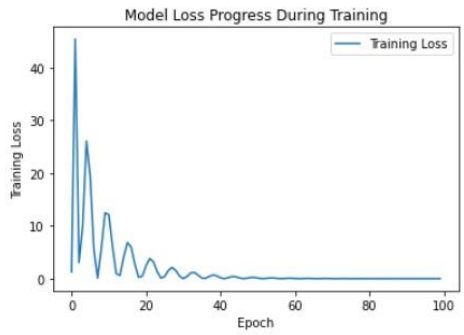
Result After Tuning Hyperparameters:

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.3), loss='mean_squared_error')
epochs_hist = model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 1/100
1/1 [==============================] - 0s 0s/step - loss: 1.2939
Epoch 2/100
1/1 [==============================] - 0s 2ms/step - loss: 45.4081
Epoch 3/100
1/1 [==============================] - 0s 1ms/step - loss: 3.0536
Epoch 4/100
1/1 [==============================] - 0s 2ms/step - loss: 10.0025
Epoch 5/100
1/1 [==============================] - 0s 1ms/step - loss: 26.0823
Epoch 6/100
1/1 [==============================] - 0s 998us/step - loss: 19.5287
Epoch 7/100
1/1 [==============================] - 0s 998us/step - loss: 5.5449
Epoch 8/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1060
Epoch 9/100
1/1 [==============================] - 0s 997us/step - loss: 5.6845
Epoch 10/100
```

```
plt.plot(epochs_hist.history['loss'])
plt.title('Model Loss Progress During Training')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.legend(['Training Loss'])
```

```
<matplotlib.legend.Legend at 0x14958b4d470>
```



EXPERIMENT NO. 4

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 3-2-2021
Faculty Signature:
Marks:

Objective(s): Implement regression model.
Outcome: Learning to make regression model in deep learning.
Problem Statement: To build an ANN model for regression problem on house predication dataset.
Background Study: *Regression* is a predictive *modeling* task that involves predicting a numerical output given some input. It is different from classification tasks that involve predicting a class label. Typically, a *regression* task involves predicting a single numeric value.

Student Work Area

Code/Sample Outputs:

Keras Regression Code Along Project

Let's now apply our knowledge to a more realistic data set. Here we will also focus on feature engineering and cleaning our data!

The Data

We will be using data from a Kaggle data set:

https://www.kaggle.com/harlfoxem/housesalesprediction

Feature Columns

- id - Unique ID for each home sold
- date - Date of the home sale
- price - Price of each home sold
- bedrooms - Number of bedrooms
- bathrooms - Number of bathrooms, where .5 accounts for a room with a toilet but no shower
- sqft_living - Square footage of the apartments interior living space
- sqft_lot - Square footage of the land space
- floors - Number of floors
- waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not
- view - An index from 0 to 4 of how good the view of the property was
- condition - An index from 1 to 5 on the condition of the apartment,
- grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 ha a high quality level of construction and design.
- sqft_above - The square footage of the interior housing space that is above ground level
- sqft_basement - The square footage of the interior housing space that is below ground level
- yr_built - The year the house was initially built
- yr_renovated - The year of the house's last renovation
- zipcode - What zipcode area the house is in
- lat - Lattitude
- long - Longitude
- sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
- sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

```
1 df = pd.read_csv('kc_house_data.csv')
```

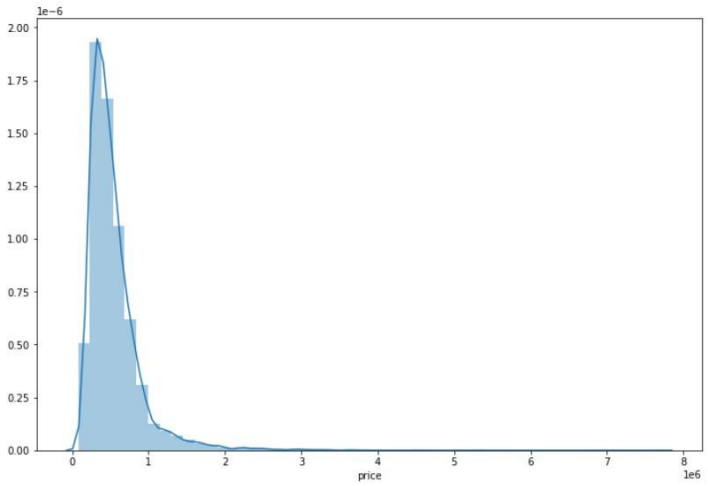
Exploratory Data Analysis

```
1 df.isnull().sum()
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

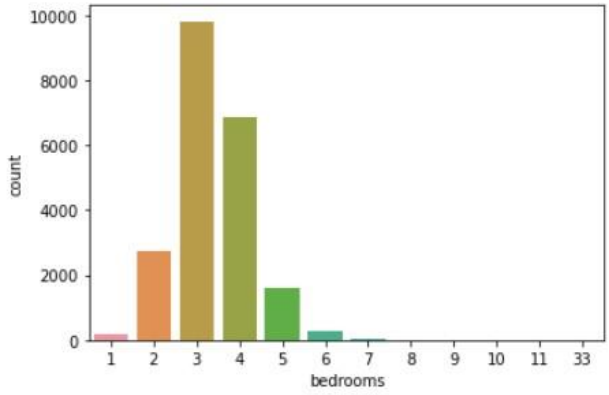
```
plt.figure(figsize=(12,8))
sns.distplot(df['price'])
```

```
<AxesSubplot:xlabel='price'>
```



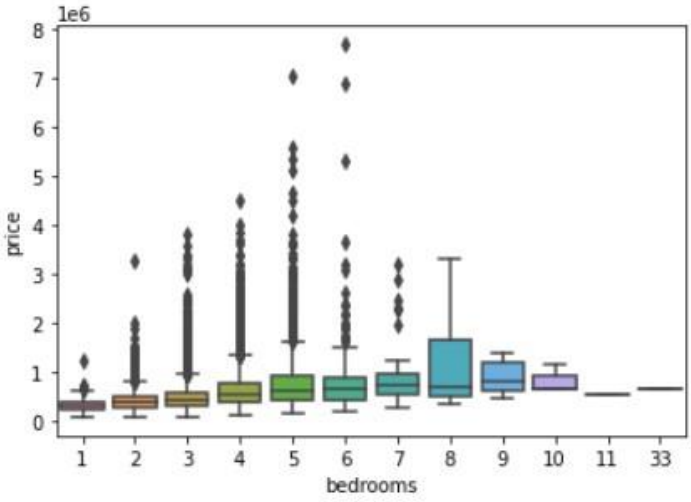

```
sns.countplot(df['bedrooms'])
```

```
<AxesSubplot:xlabel='bedrooms', ylabel='count'>
```



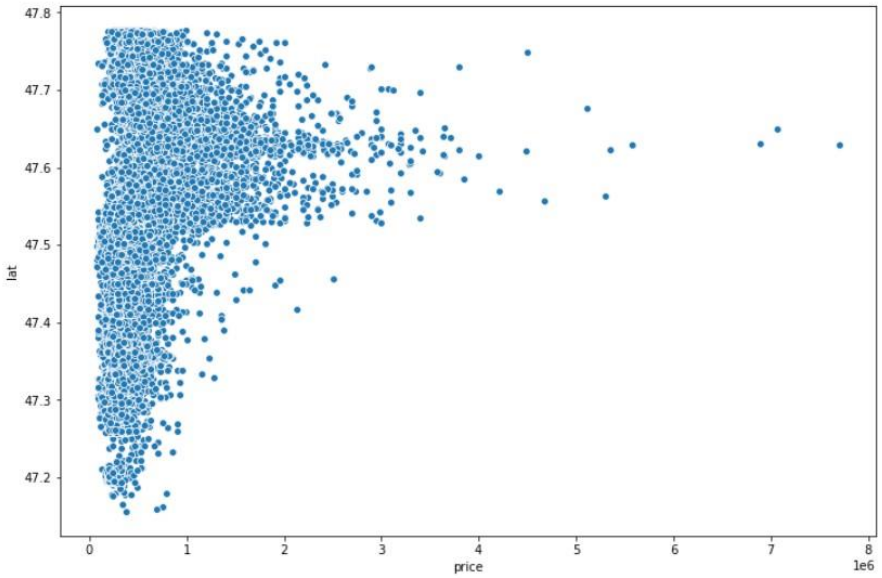

```
sns.boxplot(x='bedrooms',y='price',data=df)
```

```
<AxesSubplot:xlabel='bedrooms', ylabel='price'>
```



```
plt.figure(figsize=(12,8))
sns.scatterplot(x='price',y='lat',data=df)
```

```
<AxesSubplot:xlabel='price', ylabel='lat'>
```




```
df.sort_values('price',ascending=False).head(20)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
7245	6762700020	10/13/2014	7700000.0	6	8.00	12050	27600	2.5	0	3	...	13	8570	3480	1910
3910	9808700762	6/11/2014	7060000.0	5	4.50	10040	37325	2.0	1	2	...	11	7680	2360	1940
9245	9208900037	9/19/2014	6890000.0	6	7.75	9890	31374	2.0	0	4	...	13	8860	1030	2001
4407	2470100110	8/4/2014	5570000.0	5	5.75	9200	35069	2.0	0	0	...	13	6200	3000	2001
1446	8907500070	4/13/2015	5350000.0	5	5.00	8000	23985	2.0	0	4	...	12	6720	1280	2009
1313	7558700030	4/13/2015	5300000.0	6	6.00	7390	24829	2.0	1	4	...	12	5000	2390	1991
1162	1247600105	10/20/2014	5110000.0	5	5.25	8010	45517	2.0	1	4	...	12	5990	2020	1999
8085	1924059029	6/17/2014	4670000.0	5	6.75	9640	13068	1.0	1	4	...	12	4820	4820	1983
2624	7738500731	8/15/2014	4500000.0	5	5.50	6640	40014	2.0	1	4	...	12	6350	290	2004
8629	3835500195	6/18/2014	4490000.0	4	3.00	6430	27517	2.0	0	0	...	12	6430	0	2001
12358	6065300370	5/6/2015	4210000.0	5	6.00	7440	21540	2.0	0	0	...	12	5550	1890	2003
4145	6447300265	10/14/2014	4000000.0	4	5.50	7080	16573	2.0	0	0	...	12	5760	1320	2008
2083	8106100105	11/14/2014	3850000.0	4	4.25	5770	21300	2.0	1	4	...	11	5770	0	1980
7028	853200010	7/1/2014	3800000.0	5	5.50	7050	42840	1.0	0	2	...	13	4320	2730	1978
19002	2303900100	9/11/2014	3800000.0	3	4.25	5510	35000	2.0	0	4	...	13	4910	600	1997
16288	7397300170	5/30/2014	3710000.0	4	3.50	5550	28078	2.0	0	2	...	12	3350	2200	2000
18467	4389201095	5/11/2015	3650000.0	5	3.75	5020	8694	2.0	0	1	...	12	3970	1050	2007
6502	4217402115	4/21/2015	3650000.0	6	4.75	5480	19401	1.5	1	4	...	11	3910	1570	1936
15241	2425049063	9/11/2014	3640000.0	4	3.25	4830	22257	2.0	1	4	...	11	4830	0	1990
19133	3625049042	10/11/2014	3640000.0	5	6.00	5490	19897	2.0	0	0	...	12	5490	0	2005

20 rows × 21 columns

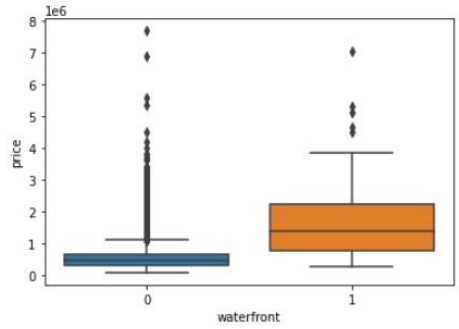
```
len(df)*(0.01)
```

```
215.97
```

Other Features

```
1 sns.boxplot(x='waterfront',y='price',data=df)
```

```
<AxesSubplot:xlabel='waterfront', ylabel='price'>
```



Working with Feature Data

```
1 df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	y
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  object
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     21597 non-null  int64
 9   view           21597 non-null  int64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  int64
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  int64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 non-null  int64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
1 df = df.drop('id',axis=1)
```

```
1 df.head()
```

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renc
0	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1955	
1	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1951	

Feature Engineering from Date

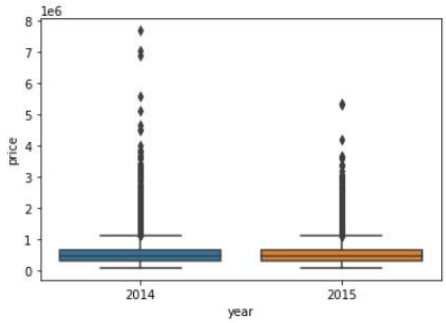
```
df['date'] = pd.to_datetime(df['date'])
```

```
df['month'] = df['date'].apply(lambda date:date.month)
```

```
df['year'] = df['date'].apply(lambda date:date.year)
```

```
sns.boxplot(x='year',y='price',data=df)
```

```
<AxesSubplot:xlabel='year', ylabel='price'>
```




```
# could make sense due to scaling, higher should correlate to more value
df['yr_renovated'].value_counts()
```

```
0       20683
2014       91
2013       37
2003       36
2000       35
        ...
1934        1
1959        1
1951        1
1948        1
1944        1
Name: yr_renovated, Length: 70, dtype: int64
```

```
df['sqft_basement'].value_counts()
```

```
0       13110
600       221
700       218
500       214
800       206
        ...
792         1
2590        1
935         1
2390        1
248         1
Name: sqft_basement, Length: 306, dtype: int64
```

Scaling and Train Test Split

```
X = df.drop('price',axis=1)
y = df['price']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=101)
```

Scaling

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
X_train= scaler.fit_transform(X_train)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:334: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

```
X_test = scaler.transform(X_test)
```

```
X_train.shape
```

```
(15117, 19)
```

```
X_test.shape
```

```
(6480, 19)
```

Creating a Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam
```

```
model = Sequential()

model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam',loss='mse')
```

Training the Model

```
%%time
model.fit(x=X_train,y=y_train.values,
          validation_data=(X_test,y_test.values),
          batch_size=128,epochs=400)
```

```
Epoch 1/400
119/119 [==============================] - 0s 2ms/step - loss: 430226112512.0000 - val_loss: 418838282240.0000
Epoch 2/400
119/119 [==============================] - 0s 2ms/step - loss: 428275073024.0000 - val_loss: 412261974016.0000
Epoch 3/400
119/119 [==============================] - 0s 2ms/step - loss: 403122749440.0000 - val_loss: 356132913152.0000
Epoch 4/400
119/119 [==============================] - 0s 2ms/step - loss: 292535402496.0000 - val_loss: 193961377792.0000
Epoch 5/400
119/119 [==============================] - 0s 2ms/step - loss: 136423989248.0000 - val_loss: 97235525632.0000
Epoch 6/400
119/119 [==============================] - 0s 2ms/step - loss: 98144493568.0000 - val_loss: 93608591360.0000
Epoch 7/400
119/119 [==============================] - 0s 2ms/step - loss: 95825108992.0000 - val_loss: 91933966336.0000
Epoch 8/400
```

Evaluation on Test Data

https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,explained_variance_score
```

Predicting on Brand New Data

```
1 X_test
```

```
array([[0.1       , 0.08      , 0.04239917, ..., 0.00887725, 0.63636364,
        0.        ],
       [0.3       , 0.36      , 0.17269907, ..., 0.00993734, 0.81818182,
        0.        ],
       [0.2       , 0.24      , 0.12512927, ..., 0.00547073, 0.90909091,
        0.        ],
       ...,
       [0.1       , 0.08      , 0.05584281, ..., 0.00506255, 1.        ,
        0.        ],
       [0.3       , 0.2       , 0.22233713, ..., 0.00774485, 0.09090909,
        1.        ],
       [0.3       , 0.32      , 0.27611169, ..., 0.0196531 , 0.45454545,
        0.        ]])
```

```
1 predictions = model.predict(X_test)
```

```
1 mean_absolute_error(y_test,predictions)
```

```
101295.51837504822
```

```
1 np.sqrt(mean_squared_error(y_test,predictions))
```

```
162912.388559915
```

```
1 explained_variance_score(y_test,predictions)
```

```
0.7998974512784672
```

```
1 df['price'].mean()
```

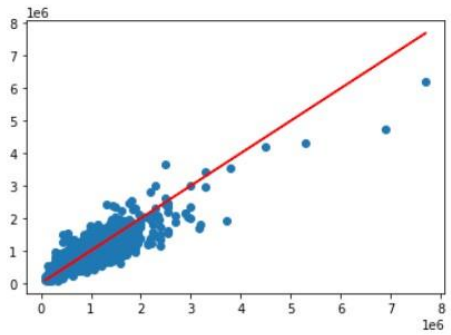
```
540296.5735055795
```

```
1 df['price'].median()
```

```
450000.0
```

```
1 # Our predictions
2 plt.scatter(y_test,predictions)
3
4 # Perfect predictions
5 plt.plot(y_test,y_test,'r')
```

```
[<matplotlib.lines.Line2D at 0x2249de3e518>]
```



Predicting on a brand new house

```
single_house = df.drop('price',axis=1).iloc[0]
```

```
single_house = scaler.transform(single_house.values.reshape(-1, 19))
```

```
single_house
```

```
array([[0.2       , 0.08      , 0.08376422, 0.00310751, 0.        ,
        0.        , 0.        , 0.5       , 0.4       , 0.10785619,
        0.        , 0.47826087, 0.        , 0.57149751, 0.21760797,
        0.16193426, 0.00582059, 0.81818182, 0.        ]])
```

```
model.predict(single_house)
```

```
array([[276145.28]], dtype=float32)
```

```
df.iloc[0]
```

```
price            221900.0000
bedrooms              3.0000
bathrooms             1.0000
sqft_living        1180.0000
sqft_lot           5650.0000
floors                1.0000
waterfront            0.0000
view                  0.0000
condition             3.0000
grade                 7.0000
sqft_above         1180.0000
sqft_basement         0.0000
yr_built           1955.0000
yr_renovated          0.0000
lat                  47.5112
long               -122.2570
sqft_living15      1340.0000
sqft_lot15         5650.0000
month                10.0000
```

EXPERIMENT NO. 5

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 10-2-2021
Faculty Signature:
Marks:

Objective(s): Implement classification model.
Outcome: Learning to make classification model using deep learning.
Problem Statement: To build an ANN model for classification problem on breast cancer classification.
Background Study: classification refers to a predictive modeling problem where a class label is predicted for a given example of input data. Examples of classification problems include: Given an example, classify if it is spam or not. Given a handwritten character, classify it as one of the known characters.

Student Work Area

Code/Sample Outputs:

The Data

Breast cancer wisconsin (diagnostic) dataset

Data Set Characteristics:

```
    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.
```

```
1 import pandas as pd
2 import numpy as np
```

```
1 df = pd.read_csv('cancer_classification.csv')
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
```

```
df.describe().transpose()
```

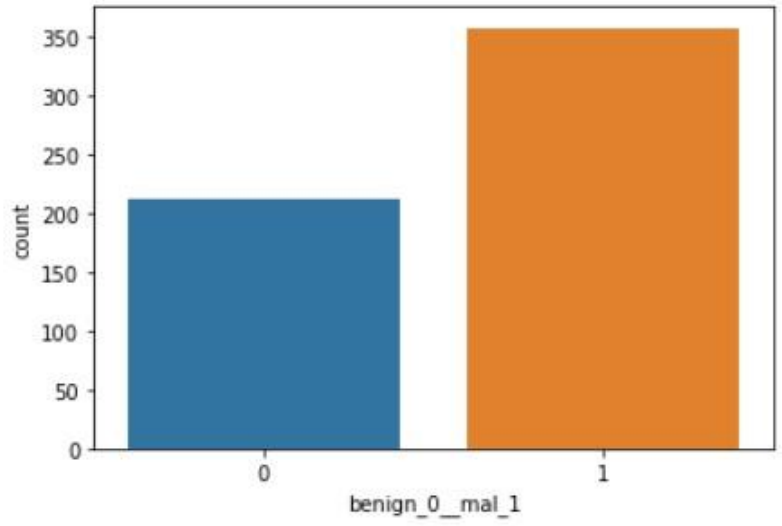
	count	mean	std	min	25%	50%	75%	max
mean radius	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
mean texture	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
mean perimeter	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
mean area	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
mean smoothness	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
mean compactness	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
mean concavity	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
mean concave points	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
mean symmetry	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
mean fractal dimension	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius error	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture error	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter error	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area error	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness error	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness error	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity error	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points error	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry error	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal dimension error	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
worst radius	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
worst texture	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
worst perimeter	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
worst area	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
worst smoothness	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22260

EDA

```
import seaborn as sns
import matplotlib.pyplot as plt
```

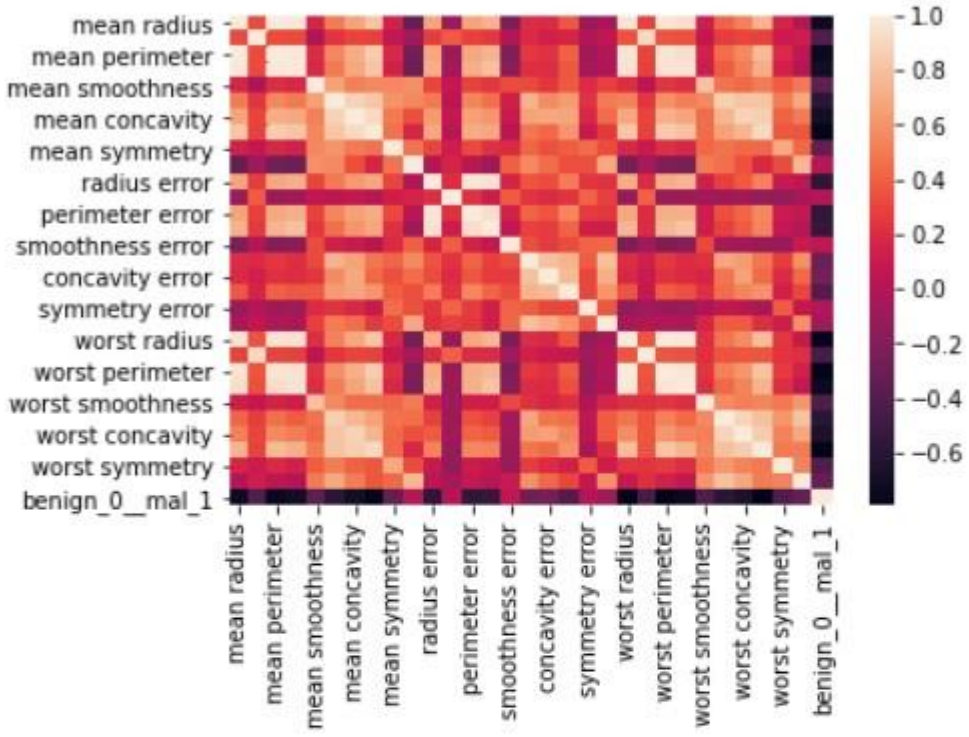
```
sns.countplot(x='benign_0__mal_1',data=df)
```

```
<AxesSubplot:xlabel='benign_0__mal_1', ylabel='count'>
```



```
1 sns.heatmap(df.corr())
```

```
<AxesSubplot:>
```




```
1 df.corr()['benign_0__mal_1'].sort_values()
```

```
worst concave points      -0.793566
worst perimeter           -0.782914
mean concave points       -0.776614
worst radius              -0.776454
mean perimeter            -0.742636
worst area                -0.733825
mean radius               -0.730029
mean area                 -0.708984
mean concavity            -0.696360
worst concavity           -0.659610
mean compactness          -0.596534
worst compactness         -0.590998
radius error              -0.567134
perimeter error           -0.556141
```

Train Test Split

```
1 X = df.drop('benign_0__mal_1',axis=1).values
2 y = df['benign_0__mal_1'].values
```

```
1 from sklearn.model_selection import train_test_split
```

```
1 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=101)
```

Scaling Data

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
scaler.fit(X_train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Creating the Model

```
# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout
```

```
X_train.shape
```

```
(426, 30)
```

```
model = Sequential()

# https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-

model.add(Dense(units=30,activation='relu'))

model.add(Dense(units=15,activation='relu'))


model.add(Dense(units=1,activation='sigmoid'))

# For a binary classification problem
model.compile(loss='binary_crossentropy', optimizer='adam')
```

Training the Model

Example One: Choosing too many epochs and overfitting!

```
%%time
# https://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network
# https://datascience.stackexchange.com/questions/18414/are-there-any-rules-for-choosing-the-size-of-a-mini-batch

model.fit(x=X_train,
          y=y_train,
          epochs=600,
          validation_data=(X_test, y_test), verbose=1
          )
```

```
Epoch 1/600
14/14 [==============================] - 0s 8ms/step - loss: 0.6847 - val_loss: 0.6727
Epoch 2/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6582 - val_loss: 0.6482
Epoch 3/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6299 - val_loss: 0.6185
Epoch 4/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5992 - val_loss: 0.5861
Epoch 5/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5597 - val_loss: 0.5433
Epoch 6/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5153 - val_loss: 0.4979
Epoch 7/600
14/14 [==============================] - 0s 3ms/step - loss: 0.4699 - val_loss: 0.4515
Epoch 8/600
14/14 [==============================] - 0s 3ms/step - loss: 0.4284 - val_loss: 0.4080
Epoch 9/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3850 - val_loss: 0.3675
Epoch 10/600
```

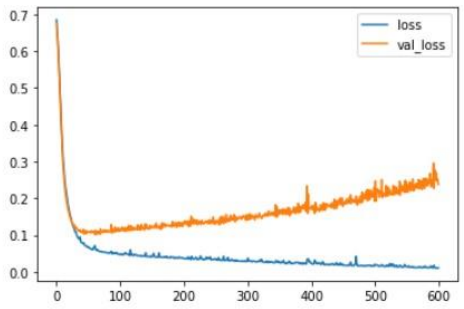
```
# model.history.history
```

```
model_loss = pd.DataFrame(model.history.history)
```

```
# model_loss
```

```
model_loss.plot()
```

```
<AxesSubplot:>
```



EXPERIMENT NO. 6

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 27-2-2021
Faculty Signature:
Marks:

Objective(s): Implement Early stopping and dropout techniaques.
Outcome: learning to implement early stopping and dropout techniques.
Problem Statement: To build an ANN model for classification problem on breast cancer classification to see the effect of: **a.** Early Stopping **b.** Dropouts
Background Study: **Early stopping:** It is a method that allows you to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on the validation dataset. **Dropout:** Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "*dropped out*." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer.

Student Work Area

Code/Sample Outputs:

Example Two: Early Stopping

We obviously trained too much! Let's use early stopping to track the val_loss and stop training once it begins increasing too much!

```
model = Sequential()
model.add(Dense(units=30,activation='relu'))
model.add(Dense(units=15,activation='relu'))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

Stop training when a monitored quantity has stopped improving.

```
Arguments:
    monitor: Quantity to be monitored.
    min_delta: Minimum change in the monitored quantity
        to qualify as an improvement, i.e. an absolute
        change of less than min_delta, will count as no
        improvement.
    patience: Number of epochs with no improvement
        after which training will be stopped.
    verbose: verbosity mode.
    mode: One of `{"auto", "min", "max"}`. In `min` mode,
        training will stop when the quantity
        monitored has stopped decreasing; in `max`
        mode it will stop when the quantity
        monitored has stopped increasing; in `auto`
        mode, the direction is automatically inferred
        from the name of the monitored quantity.
```

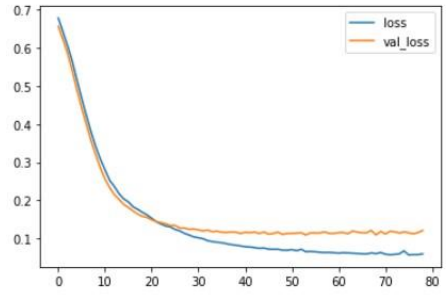
```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)
```

```
%%time
model.fit(x=X_train,
          y=y_train,
          epochs=600,
          validation_data=(X_test, y_test), verbose=1,
          callbacks=[early_stop]
          )
```

```
Epoch 1/600
14/14 [==============================] - 0s 6ms/step - loss: 0.6785 - val_loss: 0.6565
Epoch 2/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6424 - val_loss: 0.6224
Epoch 3/600
14/14 [==============================] - 0s 2ms/step - loss: 0.6054 - val_loss: 0.5853
Epoch 4/600
14/14 [==============================] - 0s 2ms/step - loss: 0.5646 - val_loss: 0.5396
Epoch 5/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5157 - val_loss: 0.4893
Epoch 6/600
14/14 [==============================] - 0s 2ms/step - loss: 0.4686 - val_loss: 0.4433
Epoch 7/600
14/14 [==============================] - 0s 3ms/step - loss: 0.4230 - val_loss: 0.3995
Epoch 8/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3800 - val_loss: 0.3568
Epoch 9/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3414 - val_loss: 0.3198
Epoch 10/600
14/14 [==============================] - 0s 3ms/step - loss: 0.3085 - val_loss: 0.2856
```

```
model_loss = pd.DataFrame(model.history.history)
model_loss.plot()
```

```
<AxesSubplot:>
```



Example Three: Adding in DropOut Layers

```
from tensorflow.keras.layers import Dropout
```

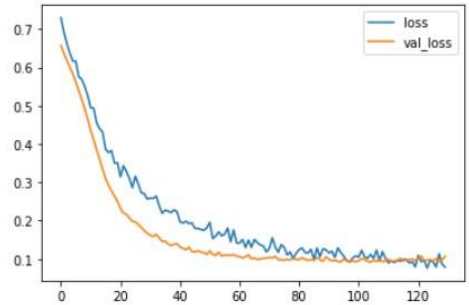
```
model = Sequential()
model.add(Dense(units=30,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=15,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
%%time
model.fit(x=X_train,
          y=y_train,
          epochs=600,
          validation_data=(X_test, y_test), verbose=1,
          callbacks=[early_stop]
          )
```

```
Epoch 1/600
14/14 [==============================] - 0s 6ms/step - loss: 0.7303 - val_loss: 0.6574
Epoch 2/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6922 - val_loss: 0.6371
Epoch 3/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6625 - val_loss: 0.6193
Epoch 4/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6373 - val_loss: 0.6010
Epoch 5/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6169 - val_loss: 0.5826
Epoch 6/600
14/14 [==============================] - 0s 3ms/step - loss: 0.6161 - val_loss: 0.5627
Epoch 7/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5756 - val_loss: 0.5394
Epoch 8/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5692 - val_loss: 0.5154
Epoch 9/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5501 - val_loss: 0.4925
Epoch 10/600
14/14 [==============================] - 0s 3ms/step - loss: 0.5267 - val_loss: 0.4651
```

```
model_loss = pd.DataFrame(model.history.history)
model_loss.plot()
```

```
<AxesSubplot:>
```



Model Evaluation

```
predictions = model.predict_classes(X_test)
```

```
WARNING:tensorflow:From <ipython-input-35-bc83193b8b59>:1: Sequential.predict_classes (from tensorflow.python.keras.engine.sequential) is deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
# https://en.wikipedia.org/wiki/Precision_and_recall
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        55
           1       0.99      0.98      0.98        88

   micro avg       0.98      0.98      0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143
```

```
print(confusion_matrix(y_test,predictions))
```

```
[[54  1]
 [ 2 86]]
```

EXPERIMENT NO. 7

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 17-2-2021
Faculty Signature:
Marks:

Objective(s): Implement K-fold, grid-search and checkpoint.
Outcome: Learning to implement k-fold, grid-search and checkpoint techniques.
Problem Statement: To build an advance ANN classification model for churn modelling data with: a. K-fold b. Grid Search c. Checkpoint
Background Study: **Kfold:** KFOLD is a model validation technique, where it's not using your pre-trained model. Rather it just use the hyper-parameter and trained a new model with k-1 data set and test the same model on the kth set.It will return the K different scores(accuracy percentage), which are based on kth test data set **Grid Search:** Grid search is a process that searches exhaustively through a manually specified subset of the hyperparameter space of the targeted algorithm. Random search, on the other hand, selects a value for each hyperparameter independently using a probability distribution. **Checkpoint:** When training deep learning models, the checkpoint is the weights of the model. These weights can be used to make predictions as is, or used as the basis for ongoing training. The API allows you to specify which metric to monitor, such as loss or accuracy on the training or validation dataset

Student Work Area

Code/Sample Outputs:

K-Fold:

```
# Artificial Neural Network

# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13]
y = dataset.iloc[:, -1]

X = X.values
y = y.values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

```

```
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("Country", OneHotEncoder(),[1])], remainder="passthrough") # The last arg ([0]) is the list of colu
X = ct.fit_transform(X)
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

```
# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import tensorflow as tf
print ("TensorFlow version: " + tf.__version__)

# Evaluating the ANN
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
```

```
TensorFlow version: 2.3.1
```

```
def build_classifier():
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
    # Probability that a node will turn off each epoc
    # Every epoc certain nodes are turned off
    classifier.add(Dropout(rate = 0.1))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dropout(rate = 0.1))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier

```

```
classifier = KerasClassifier(build_fn = build_classifier, batch_size = 10, epochs = 100)
# n_jobs = -1 can cause errors if using tensorflow gpu for multiprocessing
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_jobs = -1)
mean = accuracies.mean()
variance = accuracies.std()
```

Grid-Search:

```python
# Importing the libraries
import numpy as np
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13]
y = dataset.iloc[:, -1]

X = X.values
y = y.values
```

```python

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("Country", OneHotEncoder(),[1])], remainder="passthrough") # The last arg ([0]) is the list of colu
X = ct.fit_transform(X)
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import tensorflow as tf
print ("TensorFlow version: " + tf.__version__)

# Tuning the ANN
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from sklearn.model_selection import GridSearchCV

def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
    classifier.add(Dropout(rate = 0.1))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dropout(rate = 0.1))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier

classifier = KerasClassifier(build_fn = build_classifier)

parameters = {'batch_size': [25, 32],
              'epochs': [100, 500],
              'optimizer': ['adam', 'rmsprop']}

grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs=-1)

grid_search = grid_search.fit(X_train, y_train)

best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

```
TensorFlow version: 2.3.1
```

In [*]:

```

# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
y_pred = grid_search.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Predicting a single new observation
"""Predict if the customer with the following informations will leave the bank:
Geography: France
Credit Score: 600
Gender: Male
Age: 40
Tenure: 3
Balance: 60000
Number of Products: 2
Has Credit Card: Yes
Is Active Member: Yes
Estimated Salary: 50000"""
new_prediction = grid_search.predict(sc.transform(np.array([[0.0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])))
new_prediction = (new_prediction > 0.5)
```

Checkpoint:

```
# Artificial Neural Network

# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13]
y = dataset.iloc[:, -1]

X = X.values
y = y.values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("Country", OneHotEncoder(),[1])], remainder="passthrough") # The last arg ([0]) is the list of colu
X = ct.fit_transform(X)
X = X[:, 1:]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Importing the Keras libraries and packages
import tensorflow as tf
print ("TensorFlow version: " + tf.__version__)

# Tuning the ANN
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.model_selection import GridSearchCV

def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
    classifier.add(Dropout(rate = 0.1))
    classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dropout(rate = 0.1))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    # load weights
    # model.load_weights("weights.best.hdf5")
    classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier

classifier = KerasClassifier(build_fn = build_classifier)

parameters = {'batch_size': [25, 32],
              'epochs': [100, 500],
              'optimizer': ['adam', 'rmsprop']}

grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs=-1)

filepath="weights.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]

grid_search = grid_search.fit(X_train, y_train, callbacks=callbacks_list, verbose=0)

best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

```
TensorFlow version: 2.3.1
```

[*]:

```

# Predicting the Test set results
y_pred = grid_search.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Predicting a single new observation
"""Predict if the customer with the following informations will leave the bank:
Geography: France
Credit Score: 600
Gender: Male
Age: 40
Tenure: 3
Balance: 60000
Number of Products: 2
Has Credit Card: Yes
Is Active Member: Yes
Estimated Salary: 50000"""
new_prediction = grid_search.predict(sc.transform(np.array([[0.0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])))
new_prediction = (new_prediction > 0.5)
```

EXPERIMENT NO. 8

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 24-02-2021
Faculty Signature:
Marks:

Objective(s): Performing CNN for Image Classification on MNSIT Dataset
Outcome: Learning to implement Simple CNN model
Problem Statement: To build a CNN Model to Classify Images.
Background Study: In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs are regularized versions of multilayer perceptrons.

Student Work Area

Code/Sample Outputs:

Convolutional Neural Networks for Image Classification

```
import pandas as pd
import numpy as np
```

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Visualizing the Image Data

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
x_train.shape
```

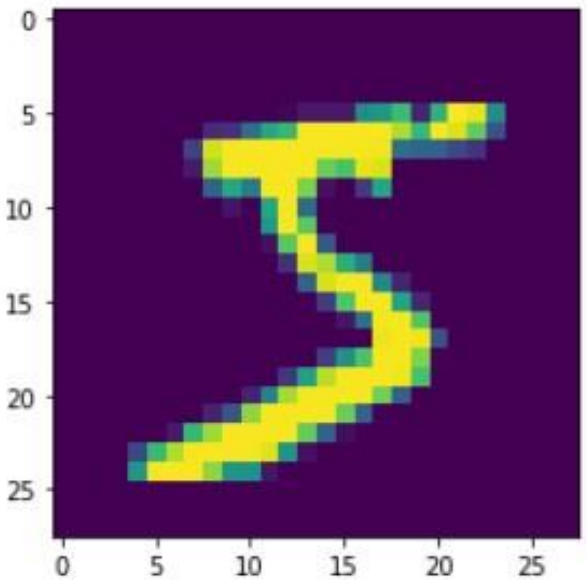
```
(60000, 28, 28)
```

```
single_image.shape
```

```
(28, 28)
```

```
plt.imshow(single_image)
```

```
<matplotlib.image.AxesImage at 0x7f8554686f50>
```



PreProcessing Data

We first need to make sure the labels will be understandable by our CNN.

Labels

```
y_train
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
y_test
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

Hmmm, looks like our labels are literally categories of numbers. We need to translate this to be "one hot encoded" so our CNN can understand, otherwise it will think this is some sort of regression problem on a continuous axis. Luckily , Keras has an easy to use function for this:

```
from tensorflow.keras.utils import to_categorical
```

```
y_train.shape
```

```
(60000,)
```

```
y_example = to_categorical(y_train)
```

```
y_example
```

```
array([[0., 0., 0., ..., 0., 0., 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., 1., 0.]], dtype=float32)
```

Processing X Data

We should normalize the X data

```
single_image.max()
```

```
255
```

```
single_image.min()
```

```
0
```

```
x_train = x_train/255
x_test = x_test/255
```

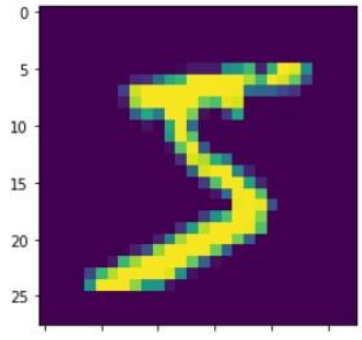
```
scaled_single = x_train[0]
```

```
scaled_single.max()
```

```
1.0
```

```
plt.imshow(scaled_single)
```

```
<matplotlib.image.AxesImage at 0x7f8554182d10>
```



Reshaping the Data

Right now our data is 60,000 images stored in 28 by 28 pixel array formation.

This is correct for a CNN, but we need to add one more dimension to show we're dealing with 1 RGB channel (since technically the images are in black and white, only showing values from 0-255 on a single channel), an color image would have 3 dimensions.

```
x_train.shape
```

```
(60000, 28, 28)
```

```
x_test.shape
```

```
(10000, 28, 28)
```

Reshape to include channel dimension (in this case, 1 channel)

```
x_train = x_train.reshape(60000, 28, 28, 1)
```

```
x_train.shape
```

```
(60000, 28, 28, 1)
```

```
x_test = x_test.reshape(10000,28,28,1)
```

```
x_test.shape
```

```
(10000, 28, 28, 1)
```

Training the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

```
model = Sequential()

# CONVOLUTIONAL LAYER, FILTER CHANGE
model.add(Conv2D(filters=16, kernel_size=(4,4),input_shape=(28, 28, 1), activation='relu',))
#ADDING ADDITIONAL LAYER AND CHANGING FILTERS
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(28, 28, 1), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 28 by 28 to 764 BEFORE FINAL LAYER
model.add(Flatten())

# 128 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF NEURONS)
model.add(Dense(128, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 25, 25, 16)        272
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 22, 22, 32)        8224
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 11, 11, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 3872)              0
_________________________________________________________________
dense (Dense)                (None, 128)               495744
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 505,530
Trainable params: 505,530
Non-trainable params: 0
_________________________________________________________________
```

```
1 from tensorflow.keras.callbacks import EarlyStopping
```

```
1 early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

Train the Model

```
1 model.fit(x_train,y_cat_train,epochs=10,validation_data=(x_test,y_cat_test),callbacks=[early_stop])
```

```
Epoch 1/10
1875/1875 [==============================] - 78s 41ms/step - loss: 0.2555 - accuracy: 0.9244 - val_loss: 0.0582 - val_accuracy:
0.9799
Epoch 2/10
1875/1875 [==============================] - 76s 41ms/step - loss: 0.0399 - accuracy: 0.9873 - val_loss: 0.0383 - val_accuracy:
0.9884
Epoch 3/10
1875/1875 [==============================] - 77s 41ms/step - loss: 0.0243 - accuracy: 0.9920 - val_loss: 0.0381 - val_accuracy:
0.9885
Epoch 4/10
1875/1875 [==============================] - 76s 41ms/step - loss: 0.0165 - accuracy: 0.9947 - val_loss: 0.0353 - val_accuracy:
0.9895
Epoch 5/10
1875/1875 [==============================] - 79s 42ms/step - loss: 0.0115 - accuracy: 0.9964 - val_loss: 0.0472 - val_accuracy:
0.9874
Epoch 6/10
1875/1875 [==============================] - 78s 42ms/step - loss: 0.0087 - accuracy: 0.9970 - val_loss: 0.0573 - val_accuracy:
0.9835

<tensorflow.python.keras.callbacks.History at 0x7f855539af90>
```

Evaluate the Model

```
1 model.metrics_names
```

```
['loss', 'accuracy']
```

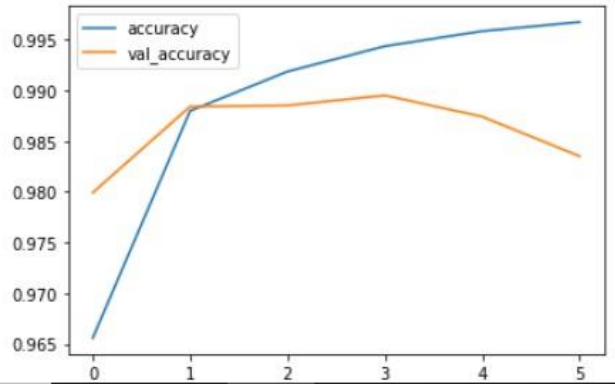
```
1 losses = pd.DataFrame(model.history.history)
```

```
1 losses.head()
```

	loss	accuracy	val_loss	val_accuracy
0	0.116103	0.965567	0.058227	0.9799
1	0.038583	0.987967	0.038329	0.9884
2	0.024922	0.991850	0.038132	0.9885
3	0.017259	0.994350	0.035318	0.9895
4	0.012699	0.995833	0.047218	0.9874

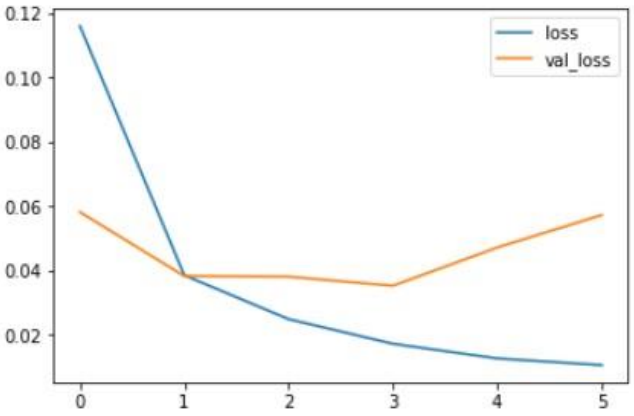
```
1 losses[['accuracy','val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f854bf95a90>
```




```
1 losses[['loss','val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f854cf0bc90>
```



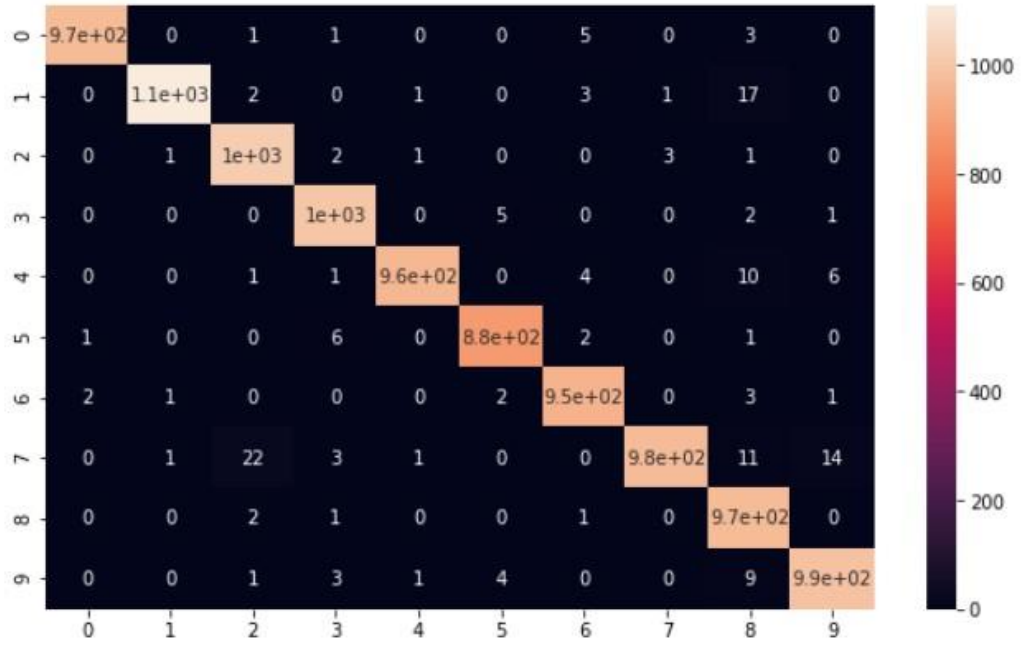

```
1 print(model.metrics_names)
2 print(model.evaluate(x_test,y_cat_test,verbose=0))
```

```
['loss', 'accuracy']
[0.0572994090616703, 0.9835000038146973]
```

```
import seaborn as sns
```

```
plt.figure(figsize=(10,6))
sns.heatmap(confusion_matrix(y_test,predictions),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8545342690>
```

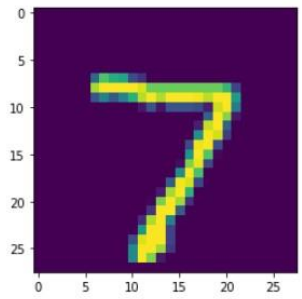


Predicting a given image

```
my_number = x_test[0]
```

```
plt.imshow(my_number.reshape(28,28))
```

```
<matplotlib.image.AxesImage at 0x7f853c9bd190>
```




```
# SHAPE --> (num_images,width,height,color_channels)
model.predict_classes(my_number.reshape(1,28,28,1))
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
array([7])
```

Effects after Hypertuning:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

```
model = Sequential()

model.add(Conv2D(filters=32,kernel_size(4,4), input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(filters=32, kernel_size(4,4), input_shape=(28, 28, 1), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizeer='adam', metrics=['accuracy'])
```

Train the Model

```
model.fit(x_train,y_cat_train,epochs=10,validation_data=(x_test,y_cat_test),callbacks=[early_stop])
```

```
Epoch 1/10
1875/1875 [==============================] - 104s 55ms/step - loss: 0.1119 - accuracy: 0.9658 - val_loss: 0.0588 - val_accurac
y: 0.9806
Epoch 2/10
1875/1875 [==============================] - 103s 55ms/step - loss: 0.0388 - accuracy: 0.9881 - val_loss: 0.0278 - val_accurac
y: 0.9920
Epoch 3/10
1875/1875 [==============================] - 102s 55ms/step - loss: 0.0243 - accuracy: 0.9923 - val_loss: 0.0407 - val_accurac
y: 0.9871
Epoch 4/10
1875/1875 [==============================] - 102s 54ms/step - loss: 0.0180 - accuracy: 0.9944 - val_loss: 0.0359 - val_accurac
y: 0.9896

<tensorflow.python.keras.callbacks.History at 0x185875e2cd0>
```

Training the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

```
model = Sequential()

# CONVOLUTIONAL LAYER, FILTER CHANGE
model.add(Conv2D(filters=16, kernel_size=(4,4),input_shape=(28, 28, 1), activation='relu',))
#ADDING ADDITIONAL LAYER AND CHANGING FILTERS
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(28, 28, 1), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 28 by 28 to 764 BEFORE FINAL LAYER
model.add(Flatten())

# 128 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF NEURONS)
model.add(Dense(128, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Train the Model

```
model.fit(x_train,y_cat_train,epochs=10,validation_data=(x_test,y_cat_test),callbacks=[early_stop])
```

```
Epoch 1/10
1875/1875 [==============================] - 67s 36ms/step - loss: 0.1155 - accuracy: 0.9649 - val_loss: 0.0419 - val_accuracy:
0.9870
Epoch 2/10
1875/1875 [==============================] - 67s 36ms/step - loss: 0.0378 - accuracy: 0.9880 - val_loss: 0.0364 - val_accuracy:
0.9879
Epoch 3/10
1875/1875 [==============================] - 67s 36ms/step - loss: 0.0247 - accuracy: 0.9920 - val_loss: 0.0331 - val_accuracy:
0.9891
Epoch 4/10
1875/1875 [==============================] - 67s 36ms/step - loss: 0.0178 - accuracy: 0.9942 - val_loss: 0.0306 - val_accuracy:
0.9905
Epoch 5/10
1875/1875 [==============================] - 69s 37ms/step - loss: 0.0139 - accuracy: 0.9957 - val_loss: 0.0382 - val_accuracy:
0.9894
Epoch 6/10
1875/1875 [==============================] - 66s 35ms/step - loss: 0.0109 - accuracy: 0.9964 - val_loss: 0.0362 - val_accuracy:
0.9909

<tensorflow.python.keras.callbacks.History at 0x18587b56d60>
```

EXPERIMENT NO. 9

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 03-03-2021
Faculty Signature:
Marks:

Objective(s): Performing CNN for Image Classification on Dogs and Cat images
Outcome: Learning to implement NN model for binary Classification.
Problem Statement: To build a CNN Model to Classify Images.
Background Study: Binary classification is the task of classifying the elements of a set into two groups on the basis of a classification rule. CNN models are used to classify the given data in either one of the category i.e. cat or dog for the given problem.

Student Work Area

Code/Sample Outputs:

```
#IMPORTING LIBRARIES
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

```
#READING DATASET
train = ImageDataGenerator(rescale=1/255)
test = ImageDataGenerator(rescale=1/255)

train_dataset = train.flow_from_directory(r"C:\Users\Piyush\Desktop\Bharat\cat-dog\dog vs cat\dataset\training_set",
                                          target_size=(150,150),
                                          batch_size = 32,
                                          class_mode = 'binary')

test_dataset = test.flow_from_directory(r"C:\Users\Piyush\Desktop\Bharat\cat-dog\dog vs cat\dataset\test_set",
                                          target_size=(150,150),
                                          batch_size =32,
                                          class_mode = 'binary')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

```
#MODEL TRAINING
model = keras.Sequential()

# Convolutional layer and maxpool layer 1
model.add(keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
model.add(keras.layers.MaxPool2D(2,2))

# Convolutional layer and maxpool layer 2
model.add(keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))

# Convolutional layer and maxpool layer 3
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))

# Convolutional layer and maxpool layer 4
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))

# This layer flattens the resulting image array to 1D array
model.add(keras.layers.Flatten())

# Hidden layer with 512 neurons and Rectified Linear Unit activation function
model.add(keras.layers.Dense(512,activation='relu'))

# Output layer with single neuron which gives 0 for Cat or 1 for Dog
#Here we use sigmoid activation function which makes our model output to lie between 0 and 1
model.add(keras.layers.Dense(1,activation='sigmoid'))
```

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
model.fit_generator(train_dataset,
        steps_per_epoch = 250,
        epochs = 10,
        validation_data = test_dataset

        )
```

```
WARNING:tensorflow:From <ipython-input-7-66f4e830450d>:1: Model.fit_generator (from tensorflow.python.keras.engine.training) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/10
250/250 [==============================] - 116s 464ms/step - loss: 0.6617 - accuracy: 0.5960 - val_loss: 0.6620 - val_accuracy:
0.6205
Epoch 2/10
250/250 [==============================] - 89s 354ms/step - loss: 0.5908 - accuracy: 0.6811 - val_loss: 0.6006 - val_accuracy:
0.6755
Epoch 3/10
250/250 [==============================] - 91s 365ms/step - loss: 0.5160 - accuracy: 0.7479 - val_loss: 0.5061 - val_accuracy:
0.7575
Epoch 4/10
250/250 [==============================] - 90s 360ms/step - loss: 0.4590 - accuracy: 0.7820 - val_loss: 0.4441 - val_accuracy:
0.8050
Epoch 5/10
250/250 [==============================] - 87s 348ms/step - loss: 0.4088 - accuracy: 0.8079 - val_loss: 0.4257 - val_accuracy:
0.8135
Epoch 6/10
250/250 [==============================] - 87s 347ms/step - loss: 0.3504 - accuracy: 0.8425 - val_loss: 0.4063 - val_accuracy:
0.8225
Epoch 7/10
250/250 [==============================] - 88s 350ms/step - loss: 0.3062 - accuracy: 0.8676 - val_loss: 0.4303 - val_accuracy:
0.8210
Epoch 8/10
```

EXPERIMENT NO. 10

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 10-03-2021
Faculty Signature:
Marks:

Objective(s): Performing LeNet Model on MNSIT Dataset
Outcome: Learning to implement a LeNet model
Problem Statement: To import LeNet Model and use it to classify images.
Background Study: **LeNet** is a convolutional neural network **structure** proposed by Yann LeCun et al. in 1989.Convolutional neural networks are a kind of feed-forward neural network whose artificial neurons can respond to a part of the surrounding cells in the coverage range and perform well in large-scale image processing.

Student Work Area

Code/Sample Outputs:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten

# https://www.tensorflow.org/api_docs/python/tf/keras/layers/ZeroPadding2D
# ZeroPadding2D This layer can add rows and columns of zeros at the top, bottom, left and right side of an image tensor

from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.regularizers import l2
from keras.datasets import mnist
from keras.utils import np_utils
import keras

# loads the MNIST dataset
(x_train, y_train), (x_test, y_test)  = mnist.load_data()

# Lets store the number of rows and columns
img_rows = x_train[0].shape[0]
img_cols = x_train[1].shape[0]

# Getting our date in the right 'shape' needed for Keras
# We need to add a 4th dimenion to our data thereby changing our
# Our original image shape of (60000,28,28) to (60000,28,28,1)
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)

# store the shape of a single image
input_shape = (img_rows, img_cols, 1)

# change our image type to float32 data type
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize our data by changing the range from (0 to 255) to (0 to 1)
x_train /= 255
x_test /= 255

# Now we one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

num_classes = y_test.shape[1]
num_pixels = x_train.shape[1] * x_train.shape[2]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 4s 0us/step
```

```
# create model
model = Sequential()

# 2 sets of CRP (Convolution, RELU, Pooling)
model.add(Conv2D(20, (5, 5),
                 padding = "same",
                 input_shape = input_shape))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

model.add(Conv2D(50, (5, 5),
                 padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

# Fully connected layers (w/ RELU)
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))

# Softmax (for classification)
model.add(Dense(num_classes))
model.add(Activation("softmax"))

model.compile(loss = 'categorical_crossentropy',
              optimizer = keras.optimizers.Adadelta(),
              metrics = ['accuracy'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 20)        520
_________________________________________________________________
activation (Activation)      (None, 28, 28, 20)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 20)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 50)        25050
_________________________________________________________________
activation_1 (Activation)    (None, 14, 14, 50)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 50)          0
_________________________________________________________________
flatten (Flatten)            (None, 2450)              0
_________________________________________________________________
dense (Dense)                (None, 500)               1225500
_________________________________________________________________
activation_2 (Activation)    (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 1,256,080
Trainable params: 1,256,080
Non-trainable params: 0
_________________________________________________________________
None
```

Now let us train LeNet on our MNIST Dataset

```
:
```

```python
# Training Parameters
batch_size = 128
epochs = 10

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test),
          shuffle=True)

#model.save("Trained Models/mnist_LeNet.h5")

# Evaluate the performance of our trained model
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
Epoch 1/10
469/469 [==============================] - 57s 121ms/step - loss: 0.8243 - accuracy: 0.8159 - val_loss: 0.7541 - val_accurac
y: 0.8395
Epoch 2/10
469/469 [==============================] - 58s 123ms/step - loss: 0.7302 - accuracy: 0.8295 - val_loss: 0.6723 - val_accurac
y: 0.8491
Epoch 3/10
469/469 [==============================] - 58s 124ms/step - loss: 0.6589 - accuracy: 0.8420 - val_loss: 0.6096 - val_accurac
y: 0.8623
Epoch 4/10
469/469 [==============================] - 58s 123ms/step - loss: 0.6037 - accuracy: 0.8520 - val_loss: 0.5611 - val_accurac
y: 0.8696
Epoch 5/10
469/469 [==============================] - 58s 123ms/step - loss: 0.5600 - accuracy: 0.8604 - val_loss: 0.5220 - val_accurac
y: 0.8779
Epoch 6/10
469/469 [==============================] - 58s 123ms/step - loss: 0.5246 - accuracy: 0.8674 - val_loss: 0.4901 - val_accurac
y: 0.8825
Epoch 7/10
469/469 [==============================] - 62s 132ms/step - loss: 0.4950 - accuracy: 0.8730 - val_loss: 0.4635 - val_accurac
y: 0.8883
```

EXPERIMENT NO. 11

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 17-03-2021
Faculty Signature:
Marks:

Objective(s): To Create Alex-Net Model on CIFAR-10 Dataset
Outcome: Learning to implement Alex-Net model
Problem Statement: To Use Alex-Net Model to classify Images.
Background Study: AlexNet allows for multi-GPU training by putting half of the model's neurons on one GPU and the other half on another GPU. Not only does this mean that a bigger model can be trained, but it also cuts down on the training time.

Student Work Area

Code/Sample Outputs:

First let's load and prep our CIFAR10 data

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.regularizers import l2

# Loads the CIFAR dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Display our data shape/dimensions
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Now we one hot encode outputs
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 180s 1us/step
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

```
l2_reg = 0

# Initialize model
model = Sequential()

# 1st Conv Layer
model.add(Conv2D(96, (11, 11), input_shape=x_train.shape[1:],
    padding='same', kernel_regularizer=l2(l2_reg)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 2nd Conv Layer
model.add(Conv2D(256, (5, 5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 3rd Conv Layer
model.add(ZeroPadding2D((1, 1)))
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 4th Conv Layer
model.add(ZeroPadding2D((1, 1)))
model.add(Conv2D(1024, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))

# 5th Conv Layer
model.add(ZeroPadding2D((1, 1)))
model.add(Conv2D(1024, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 1st FC Layer
```

```
# 1st FC Layer
model.add(Flatten())
model.add(Dense(3072))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

# 2nd FC Layer
model.add(Dense(4096))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

# 3rd FC Layer
model.add(Dense(num_classes))
model.add(BatchNormalization())S
model.add(Activation('softmax'))

print(model.summary())

model.compile(loss = 'categorical_crossentropy',
              optimizer = keras.optimizers.Adadelta(),
              metrics = ['accuracy'])

```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 96)        34944
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 96)        384
_________________________________________________________________
activation (Activation)      (None, 32, 32, 96)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 256)       614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
activation_1 (Activation)    (None, 16, 16, 256)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 10, 10, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 512)       1180160
_________________________________________________________________
batch_normalization_2 (Batch (None, 10, 10, 512)       2048
_________________________________________________________________
activation_2 (Activation)    (None, 10, 10, 512)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 512)         0
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 7, 7, 512)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 1024)        4719616
_________________________________________________________________
batch_normalization_3 (Batch (None, 7, 7, 1024)        4096
_________________________________________________________________
activation_3 (Activation)    (None, 7, 7, 1024)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 9, 9, 1024)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 1024)        9438208
```

```
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 9, 1024)        4096
_________________________________________________________________
activation_4 (Activation)    (None, 9, 9, 1024)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 1024)        0
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 3072)              50334720
_________________________________________________________________
batch_normalization_5 (Batch (None, 3072)              12288
_________________________________________________________________
activation_5 (Activation)    (None, 3072)              0
_________________________________________________________________
dropout (Dropout)            (None, 3072)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              12587008
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384
_________________________________________________________________
activation_6 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                40970
_________________________________________________________________
batch_normalization_7 (Batch (None, 10)                40
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 78,990,642
Trainable params: 78,970,462
Non-trainable params: 20,180
_________________________________________________________________
None
```

Now let us train AlexNet on our CIFAR10 Dataset

```python
# Training Parameters
batch_size = 32
epochs = 1

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test),
          shuffle=True)

model.save("Trained Models/CIFAR10_AlexNet_1_Epoch.h5")

# Evaluate the performance of our trained model
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
1563/1563 [==============================] - 3343s 2s/step - loss: 2.0685 - accuracy: 0.2613 - val_loss: 1.7043 - val_accuracy:
0.4194
```

EXPERIMENT NO. 12

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 17-03-2021
Faculty Signature:
Marks:

Objective(s): To Build an Image classifier using Keras and CNN
Outcome: Learning to build an image classifier
Problem Statement: To build an image classifier for fashion MNIST dataset.
Background Study: Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

Student Work Area

Code/Sample Outputs:

```
import keras
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets,models,layers
# Load the fashion-mnist pre-shuffled train data and test data
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
print("x_train shape:", x_train.shape ,"Y_train shape:", y_train.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 4us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 302s 11us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 32s 7us/step
x_train shape: (60000, 28, 28) Y_train shape: (60000,)
```

```
X_train=x_train.astype('float32') / 255
X_test=x_test.astype('float32') / 255
X_train=X_train.reshape(60000,28,28,1)
X_test=X_test.reshape(10000,28,28,1)
X_train[0].shape
```

```
(28, 28, 1)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten

model = Sequential()
model.add(Conv2D(64,(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 13, 13, 64)        0
_________________________________________________________________
dropout_21 (Dropout)         (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 13, 13, 32)        8224
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
dropout_22 (Dropout)         (None, 6, 6, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               295168
_________________________________________________________________
dropout_23 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                2570
=================================================================
Total params: 306,602
Trainable params: 306,602
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
model.fit(x_train,
          y_train,
          batch_size=64,
          epochs=10,
          validation_data=(x_test, y_test),
          )
```

```
Epoch 1/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.4383 - accuracy: 0.8417 - val_loss: 0.3559 - val_accuracy:
0.8731
Epoch 2/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.2908 - accuracy: 0.8939 - val_loss: 0.3014 - val_accuracy:
0.8873
Epoch 3/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.2458 - accuracy: 0.9092 - val_loss: 0.2569 - val_accuracy:
0.9055
Epoch 4/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.2156 - accuracy: 0.9207 - val_loss: 0.2608 - val_accuracy:
0.9035
Epoch 5/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.1875 - accuracy: 0.9294 - val_loss: 0.2542 - val_accuracy:
0.9087
Epoch 6/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.1657 - accuracy: 0.9377 - val_loss: 0.2769 - val_accuracy:
0.9052
Epoch 7/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.1434 - accuracy: 0.9456 - val_loss: 0.2639 - val_accuracy:
0.9115
Epoch 8/10
1875/1875 [==============================] - 24s 13ms/step - loss: 0.1258 - accuracy: 0.9515 - val_loss: 0.2684 - val_accuracy:
0.9120
Epoch 9/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.1097 - accuracy: 0.9592 - val_loss: 0.2968 - val_accuracy:
0.9125
Epoch 10/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.0965 - accuracy: 0.9640 - val_loss: 0.3100 - val_accuracy:
0.9114
```

EXPERIMENT NO. 13

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 31-03-2021
Faculty Signature:
Marks:

Objective(s): To train a CNN model for CIFAR-10 Dataset
Outcome: Learning to train CNN model
Problem Statement: To build a CNN Model to Classify Images.
Background Study: In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs are regularized versions of multilayer perceptrons.

Student Work Area

Code/Sample Outputs:

```
#IMPORTING LIBRARIES
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras import datasets,models,layers
```

```
#LOADING DATASET
data = tf.keras.datasets.cifar10
```

```
(train_images, train_labels), (test_images, test_labels) = data.load_data()
```

```
#NORMALIZING DATA
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
#MODEL
model = tf.keras.models.Sequential([
                        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
                        tf.keras.layers.MaxPooling2D(2, 2),
                        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
                        tf.keras.layers.MaxPooling2D(2, 2),
                        tf.keras.layers.Conv2D(512, (3, 3), activation='relu'),
                        tf.keras.layers.Flatten(),
                        tf.keras.layers.Dense(512, activation='relu'),
                        tf.keras.layers.Dense(10, activation='softmax')])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 512)         295424
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 4,514,762
Trainable params: 4,514,762
Non-trainable params: 0
_________________________________________________________________
```

```python
#TRAINING MODEL
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=20,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1563/1563 [==============================] - 60s 39ms/step - loss: 1.3950 - accuracy: 0.4951 - val_loss: 1.1095 - val_accurac
y: 0.6045
Epoch 2/20
1563/1563 [==============================] - 43s 28ms/step - loss: 0.9893 - accuracy: 0.6498 - val_loss: 0.9342 - val_accurac
y: 0.6707
Epoch 3/20
1563/1563 [==============================] - 43s 27ms/step - loss: 0.7922 - accuracy: 0.7225 - val_loss: 0.8590 - val_accurac
y: 0.7014
Epoch 4/20
1563/1563 [==============================] - 43s 28ms/step - loss: 0.6303 - accuracy: 0.7803 - val_loss: 0.9208 - val_accurac
y: 0.6957
Epoch 5/20
1563/1563 [==============================] - 44s 28ms/step - loss: 0.4768 - accuracy: 0.8328 - val_loss: 0.9345 - val_accurac
y: 0.7115
Epoch 6/20
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3294 - accuracy: 0.8830 - val_loss: 0.9962 - val_accurac
y: 0.7120
Epoch 7/20
1563/1563 [==============================] - 43s 27ms/step - loss: 0.2195 - accuracy: 0.9229 - val_loss: 1.2158 - val_accurac
```

EXPERIMENT NO. 14

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 28-04-2021
Faculty Signature:
Marks:

Objective(s): To implement autoencoders for dimensionality reduction.
Outcome: Learning to implement autoencoders.
Problem Statement: Use autoencoders for dimensionality reduction.
Background Study: Dimensionality reduction, or dimension reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the lowdimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

Student Work Area

Code/Sample Outputs:

AutoEncoders for Dimensionality Reduction

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

```
from sklearn.datasets import make_blobs
```

```
data = make_blobs(n_samples=300,
    n_features=2,
    centers=2,
    cluster_std=1.0,random_state=101)
```

```
X,y = data
```

```
np.random.seed(seed=101)
z_noise = np.random.normal(size=len(X))
z_noise = pd.Series(z_noise)
```

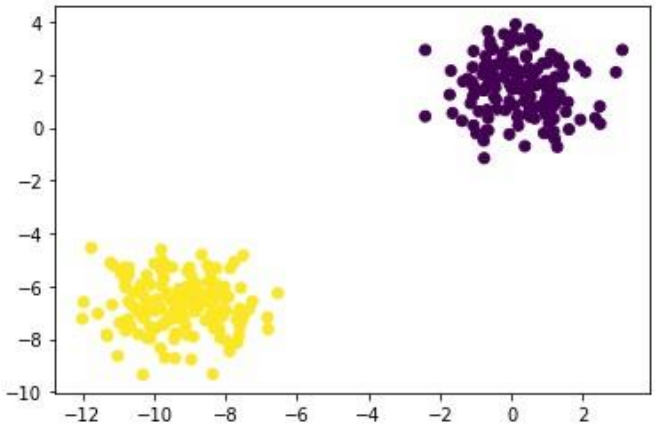
```
feat = pd.DataFrame(X)
feat = pd.concat([feat,z_noise],axis=1)
feat.columns = ['X1','X2','X3']
```

```
feat.head()
```

	X1	X2	X3
0	-8.877721	-6.760078	2.706850
1	0.630638	3.107075	0.628133
2	-0.240609	2.820690	0.907969
3	-7.839091	-8.207545	0.503826
4	-10.972908	-7.390676	0.651118

```
plt.scatter(feat['X1'],feat['X2'],c=y)
```

```
<matplotlib.collections.PathCollection at 0x7f8baebaf630>
```



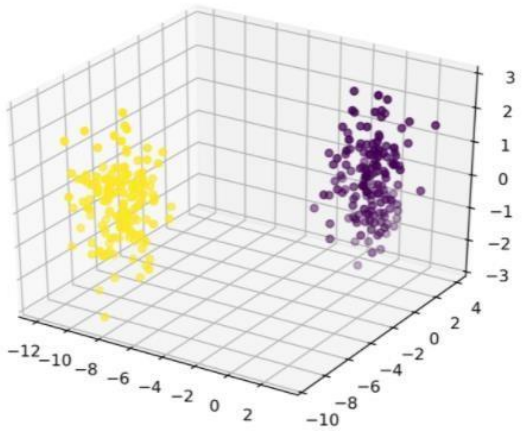
https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html#scatter-plots

```
from mpl_toolkits.mplot3d import Axes3D
```

```
%matplotlib notebook
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(feat['X1'],feat['X2'],feat['X3'],c=y)
```

```
<IPython.core.display.Javascript object>
```




```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f8baea40e48>
```

Encoder and Decoder

Stochastic Gradient Descent Optimizaer recommended for Auto Encoders because we can control the learning rate.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
```

```
# 3 --> 2
encoder = Sequential()
encoder.add(Dense(units=2,activation='relu',input_shape=[3]))
```

```
# 2 ---> 3
decoder = Sequential()
decoder.add(Dense(units=3,activation='relu',input_shape=[2]))
```

```
# ENCODER
# 3 ---> 2 ----> 3
autoencoder = Sequential([encoder,decoder])
autoencoder.compile(loss="mse" ,optimizer=SGD(lr=1.5))
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
# Note how all the data is used! There is no "right" answer here
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(feat)
```

```
1 scaled_data
```

```
array([[2.07475524e-01, 1.95571963e-01, 9.85845750e-01],
       [8.35929679e-01, 9.39652004e-01, 6.09403243e-01],
       [7.78344698e-01, 9.18055791e-01, 6.60079904e-01],
       [2.76123683e-01, 8.64187747e-02, 5.86892040e-01],
       [6.89943218e-02, 1.48018660e-01, 6.13565724e-01],
       [7.93328662e-02, 2.66181866e-01, 4.37825912e-01],
       [8.58345111e-01, 7.43675286e-01, 3.42071017e-01],
       [1.13649812e-01, 1.61015757e-01, 6.05388874e-01],
       [7.97071012e-01, 9.69575837e-01, 1.30174900e-01],
       [8.04420781e-01, 8.70758874e-01, 6.29683804e-01],
       [1.45851346e-01, 3.01543367e-01, 5.91417162e-01],
       [5.50858487e-02, 2.00066208e-01, 3.88988121e-01],
       [8.28535567e-01, 9.61146391e-01, 5.29823913e-01],
       [2.59466086e-01, 1.83035970e-01, 3.58225464e-01],
       [8.67698238e-01, 7.58466630e-01, 3.26649038e-01],
       [2.98886547e-01, 1.72814182e-01, 6.68607073e-01],
       [7.44511367e-01, 8.79978328e-01, 5.30204031e-01],
       [8.41866546e-01, 8.28631260e-01, 8.53992807e-01],
       [7.40798609e-01, 8.11956328e-01, 9.67576557e-01],
```

```
1 autoencoder.fit(scaled_data,scaled_data,epochs=5)
```

```
Train on 300 samples
Epoch 1/5
300/300 [==============================] - 0s 2ms/sample - loss: 0.2592
Epoch 2/5
300/300 [==============================] - 0s 381us/sample - loss: 0.2544
Epoch 3/5
300/300 [==============================] - 0s 294us/sample - loss: 0.2496
Epoch 4/5
300/300 [==============================] - 0s 198us/sample - loss: 0.1862
Epoch 5/5
300/300 [==============================] - 0s 195us/sample - loss: 0.1708

<tensorflow.python.keras.callbacks.History at 0x7f8b7c042710>
```

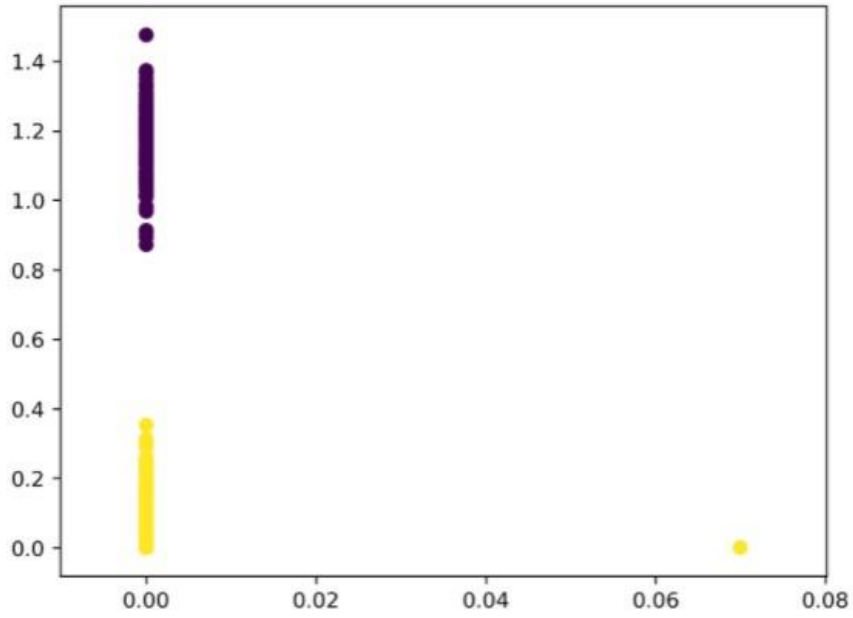
```
1 encoded_2dim = encoder.predict(scaled_data)
```

```
1 encoded_2dim
```

```
array([[0.        , 0.2374014 ],
       [0.        , 1.3352594 ],
       [0.        , 1.2914723 ],
       [0.        , 0.04181945],
       [0.        , 0.        ],
       [0.        , 0.0842607 ],
       [0.        , 1.0526376 ],
       [0.        , 0.03457493],
       [0.        , 1.2089851 ],
       [0.        , 1.2452296 ],
       [0.        , 0.2070699 ],
       [0.        , 0.        ],
       [0.        , 1.3322477 ],
       [0.        , 0.07492808],
       [0.        , 1.0703382 ],
       [0.        , 0.17540339],
       [0.        , 1.1917231 ],
       [0.        , 1.2842189 ],
       [0.        , 1.2381858 ],
```

```
1 plt.scatter(encoded_2dim[:,0],encoded_2dim[:,1],c=y)
```

```
<IPython.core.display.Javascript object>
```




```
<matplotlib.collections.PathCollection at 0x7f8b5c4d4438>
```

EXPERIMENT NO. 15

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 28-04-2021
Faculty Signature:
Marks:

Objective(s): Using MNIST dataset, improve autoencoder's performance using convolutional layers
Outcome: Learning to implement autoencoders with CNN layers
Problem Statement: To improve autoencoder's performance using cnn layers
Background Study: Autoencoder is a type of neural network that can be used to learn a compressed representation of raw data. An autoencoder is composed of an encoder and a decoder submodel. The encoder compresses the input, and the decoder attempts to recreate the input from the compressed version provided by the encoder.

Student Work Area

Code/Sample Outputs:

Convolutional Autoencoder

Sticking with the MNIST dataset, let's improve our autoencoder's performance using convolutional layers. Again, loading modules and the data.

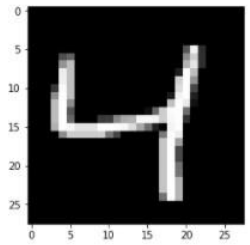
```
%matplotlib inline

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', validation_size=0)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

```
img = mnist.train.images[2]
plt.imshow(img.reshape((28, 28)), cmap='Greys_r')
```

```
<matplotlib.image.AxesImage at 0x7f4631f1a4e0>
```



Network Architecture

The encoder part of the network will be a typical convolutional pyramid. Each convolutional layer will be followed by a max-pooling layer to reduce the dimensions of the layers. The decoder though might be something new to you. The decoder needs to convert from a narrow representation to a wide reconstructed image. For example, the representation could be a 4x4x8 max-pool layer. This is the output of the encoder, but also the input to the decoder. We want to get a 28x28x1 image out from the decoder so we need to work our way back up from the narrow decoder input layer. A schematic of the network is shown below.

```
inputs_ = tf.placeholder(tf.float32, (None, 28, 28, 1), name='inputs')
targets_ = tf.placeholder(tf.float32, (None, 28, 28, 1), name='targets')

### Encoder
conv1 = tf.layers.conv2d(inputs_, 16, (3,3), padding='same', activation=tf.nn.relu)
# Now 28x28x16
maxpool1 = tf.layers.max_pooling2d(conv1, (2,2), (2,2), padding='same')
# Now 14x14x16
conv2 = tf.layers.conv2d(maxpool1, 8, (3,3), padding='same', activation=tf.nn.relu)
# Now 14x14x8
maxpool2 = tf.layers.max_pooling2d(conv2, (2,2), (2,2), padding='same')
# Now 7x7x8
conv3 = tf.layers.conv2d(maxpool2, 8, (3,3), padding='same', activation=tf.nn.relu)
# Now 7x7x8
encoded = tf.layers.max_pooling2d(conv3, (2,2), (2,2), padding='same')
# Now 4x4x8

### Decoder
upsample1 = tf.image.resize_nearest_neighbor(encoded, (7,7))
# Now 7x7x8
conv4 = tf.layers.conv2d(upsample1, 8, (3,3), padding='same', activation=tf.nn.relu)
# Now 7x7x8
upsample2 = tf.image.resize_nearest_neighbor(conv4, (14,14))
# Now 14x14x8
conv5 = tf.layers.conv2d(upsample2, 8, (3,3), padding='same', activation=tf.nn.relu)
# Now 14x14x8
upsample3 = tf.image.resize_nearest_neighbor(conv5, (28,28))
# Now 28x28x8
conv6 = tf.layers.conv2d(upsample3, 16, (3,3), padding='same', activation=tf.nn.relu)
# Now 28x28x16

logits = tf.layers.conv2d(conv6, 1, (3,3), padding='same', activation=None)
#Now 28x28x1

decoded = tf.nn.sigmoid(logits, name='decoded')

loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=targets_, logits=logits)
cost = tf.reduce_mean(loss)
opt = tf.train.AdamOptimizer(0.001).minimize(cost)
```

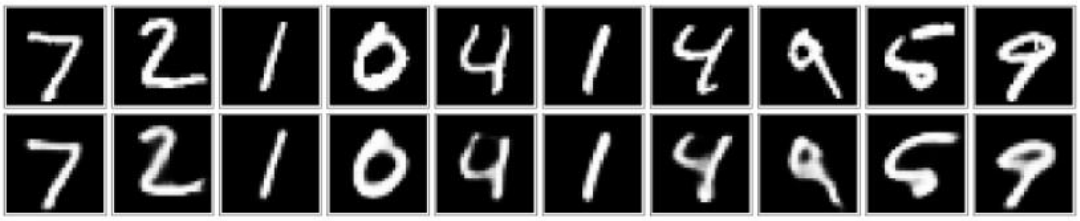
Training

As before, here wi'll train the network. Instead of flattening the images though, we can pass them in as 28x28x1 arrays.

```
sess = tf.Session()
```

```
epochs = 20
batch_size = 200
sess.run(tf.global_variables_initializer())
for e in range(epochs):
    for ii in range(mnist.train.num_examples//batch_size):
        batch = mnist.train.next_batch(batch_size)
        imgs = batch[0].reshape((-1, 28, 28, 1))
        batch_cost, _ = sess.run([cost, opt], feed_dict={inputs_: imgs,
                                                         targets_: imgs})

        print("Epoch: {}/{}...".format(e+1, epochs),
              "Training loss: {:.4f}".format(batch_cost))
```

```
fig, axes = plt.subplots(nrows=2, ncols=10, sharex=True, sharey=True, figsize=(20,4))
in_imgs = mnist.test.images[:10]
reconstructed = sess.run(decoded, feed_dict={inputs_: in_imgs.reshape((10, 28, 28, 1))})

for images, row in zip([in_imgs, reconstructed], axes):
    for img, ax in zip(images, row):
        ax.imshow(img.reshape((28, 28)), cmap='Greys_r')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)


fig.tight_layout(pad=0.1)
```



```
sess.close()
```

Denoising

As I've mentioned before, autoencoders like the ones you've built so far aren't too useful in practive. However, they can be used to denoise images quite successfully just by training the network on noisy images. We can create the noisy images ourselves by adding Gaussian noise to the training images, then clipping the values to be between 0 and 1. We'll use noisy images as input and the original, clean images as targets. Here's an example of the noisy images I generated and the denoised images.

```
inputs_ = tf.placeholder(tf.float32, (None, 28, 28, 1), name='inputs')
targets_ = tf.placeholder(tf.float32, (None, 28, 28, 1), name='targets')

### Encoder
conv1 = tf.layers.conv2d(inputs_, 32, (3,3), padding='same', activation=tf.nn.relu)
# Now 28x28x32
maxpool1 = tf.layers.max_pooling2d(conv1, (2,2), (2,2), padding='same')
# Now 14x14x32
conv2 = tf.layers.conv2d(maxpool1, 32, (3,3), padding='same', activation=tf.nn.relu)
# Now 14x14x32
maxpool2 = tf.layers.max_pooling2d(conv2, (2,2), (2,2), padding='same')
# Now 7x7x32
conv3 = tf.layers.conv2d(maxpool2, 16, (3,3), padding='same', activation=tf.nn.relu)
# Now 7x7x16
encoded = tf.layers.max_pooling2d(conv3, (2,2), (2,2), padding='same')
# Now 4x4x16

### Decoder
upsample1 = tf.image.resize_nearest_neighbor(encoded, (7,7))
# Now 7x7x16
conv4 = tf.layers.conv2d(upsample1, 16, (3,3), padding='same', activation=tf.nn.relu)
# Now 7x7x16
upsample2 = tf.image.resize_nearest_neighbor(conv4, (14,14))
# Now 14x14x16
conv5 = tf.layers.conv2d(upsample2, 32, (3,3), padding='same', activation=tf.nn.relu)
# Now 14x14x32
upsample3 = tf.image.resize_nearest_neighbor(conv5, (28,28))
# Now 28x28x32
conv6 = tf.layers.conv2d(upsample3, 32, (3,3), padding='same', activation=tf.nn.relu)
# Now 28x28x32

logits = tf.layers.conv2d(conv6, 1, (3,3), padding='same', activation=None)
#Now 28x28x1

decoded = tf.nn.sigmoid(logits, name='decoded')

loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=targets_, logits=logits)
cost = tf.reduce_mean(loss)
opt = tf.train.AdamOptimizer(0.001).minimize(cost)
```

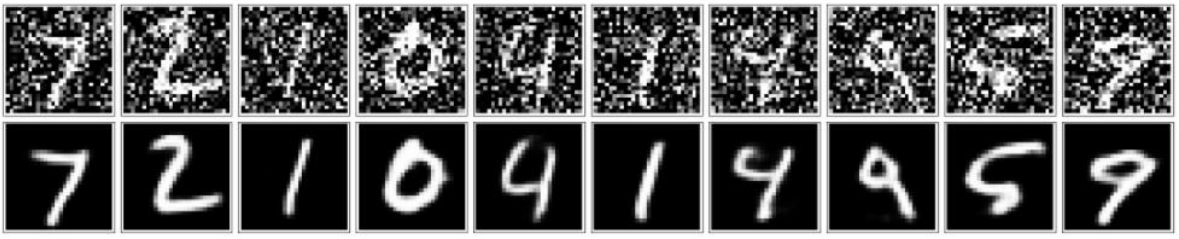
```
sess = tf.Session()
```

```
epochs = 100
batch_size = 200
# Set's how much noise we're adding to the MNIST images
noise_factor = 0.5
sess.run(tf.global_variables_initializer())
for e in range(epochs):
    for ii in range(mnist.train.num_examples//batch_size):
        batch = mnist.train.next_batch(batch_size)
        # Get images from the batch
        imgs = batch[0].reshape((-1, 28, 28, 1))

        # Add random noise to the input images
        noisy_imgs = imgs + noise_factor * np.random.randn(*imgs.shape)
        # Clip the images to be between 0 and 1
        noisy_imgs = np.clip(noisy_imgs, 0., 1.)

        # Noisy images as inputs, original images as targets
        batch_cost, _ = sess.run([cost, opt], feed_dict={inputs_: noisy_imgs,
                                                         targets_: imgs})

        print("Epoch: {}/{}...".format(e+1, epochs),
              "Training loss: {:.4f}".format(batch_cost))
```

Checking out the performance

Here I'm adding noise to the test images and passing them through the autoencoder. It does a suprising great job of removing the noise, even though it's sometimes difficult to tell what the original number is.

```
fig, axes = plt.subplots(nrows=2, ncols=10, sharex=True, sharey=True, figsize=(20,4))
in_imgs = mnist.test.images[:10]
noisy_imgs = in_imgs + noise_factor * np.random.randn(*in_imgs.shape)
noisy_imgs = np.clip(noisy_imgs, 0., 1.)

reconstructed = sess.run(decoded, feed_dict={inputs_: noisy_imgs.reshape((10, 28, 28, 1))})

for images, row in zip([noisy_imgs, reconstructed], axes):
    for img, ax in zip(images, row):
        ax.imshow(img.reshape((28, 28)), cmap='Greys_r')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

fig.tight_layout(pad=0.1)
```



EXPERIMENT NO. 16

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 05-05-2021
Faculty Signature:
Marks:

Objective(s): To train a RNN model Alcohol sales Dataset
Outcome: Learning to train RNN model
Problem Statement: To build a RNN Model on Alcohol sales dataset.
Background Study: Recurrent Neural Network(RNN) are a type of Neural Network where the output from previous step are fed as input to the current step.It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output.

Student Work Area

Code/Sample Outputs:

```
1 import pandas as pd
2 import numpy as np
3 %matplotlib inline
4 import matplotlib.pyplot as plt
```

Data

https://fred.stlouisfed.org/series/S4248SM144NCEN

```
1 df = pd.read_csv('Alcohol_Sales.csv',index_col='DATE',parse_dates=True)
2 df.index.freq = 'MS'
```

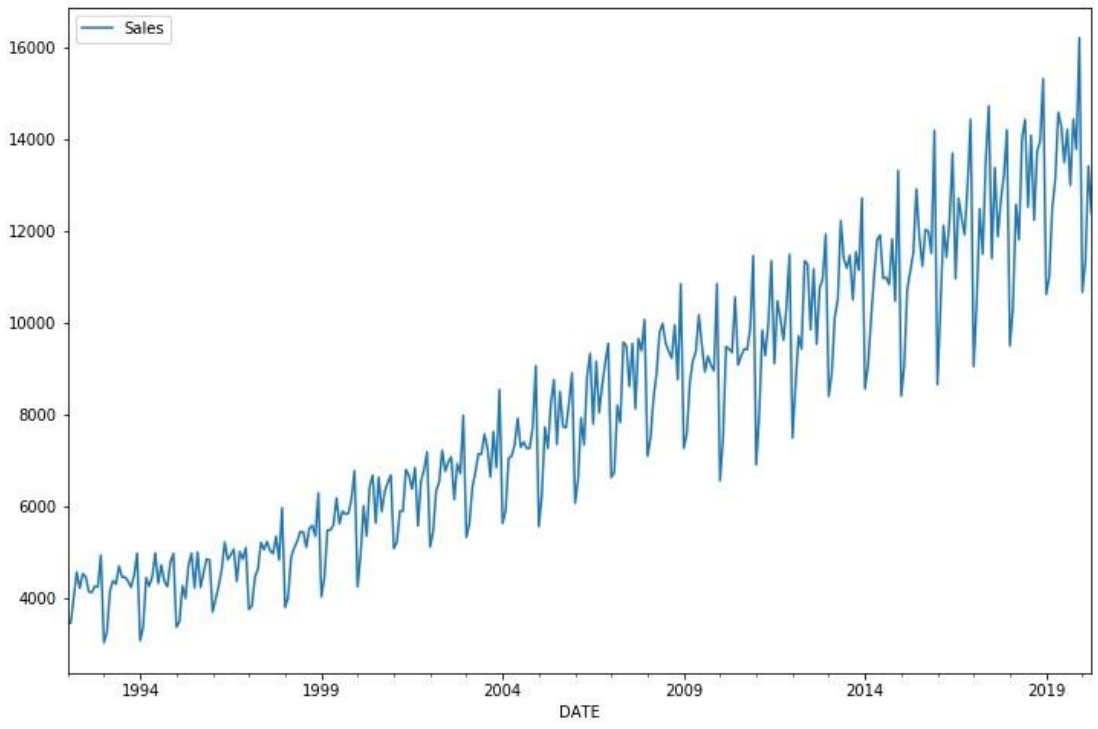
```
1 df.head()
```

DATE	S4248SM144NCEN
1992-01-01	3459
1992-02-01	3458
1992-03-01	4002
1992-04-01	4564
1992-05-01	4221

```
1 df.columns = ['Sales']
```

```
1 df.plot(figsize=(12,8))
```

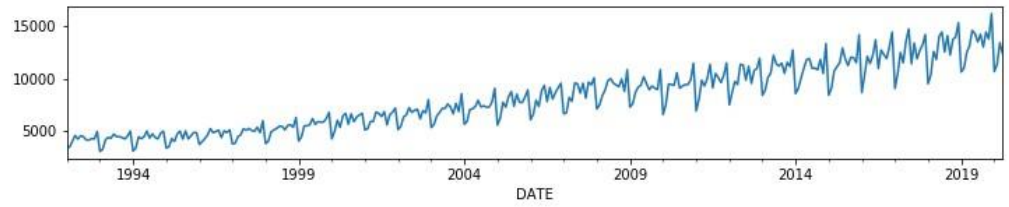
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0ec40c780>
```



```
from statsmodels.tsa.seasonal import seasonal_decompose
```

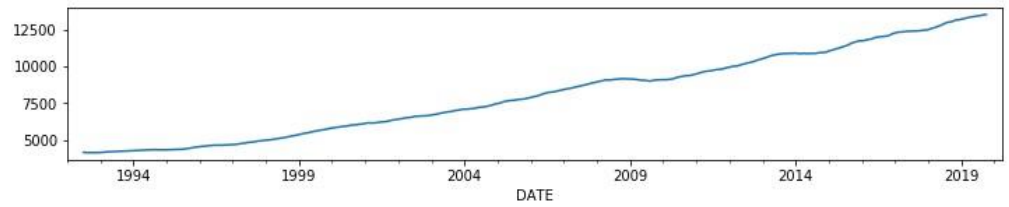
```
results = seasonal_decompose(df['Sales'])
results.observed.plot(figsize=(12,2))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0e81e85c0>
```



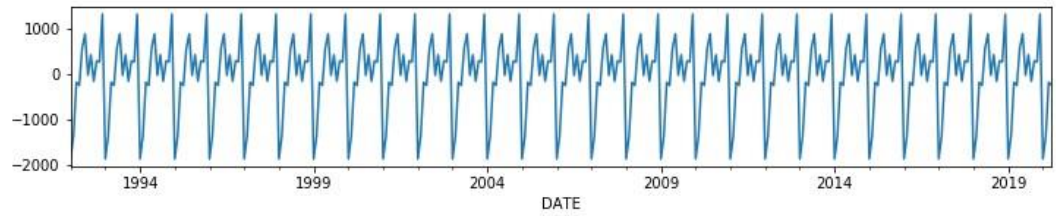

```
results.trend.plot(figsize=(12,2))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0e80cca90>
```



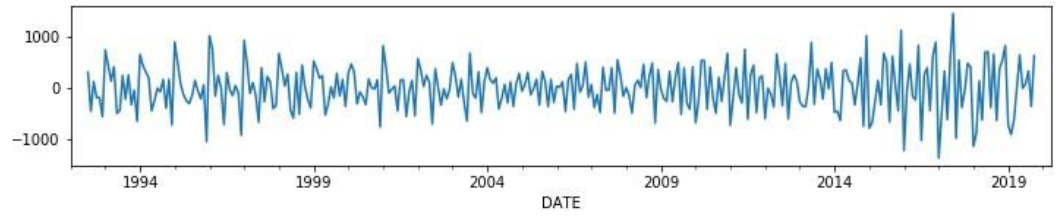

```
results.seasonal.plot(figsize=(12,2))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0e804bc88>
```




```
results.resid.plot(figsize=(12,2))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0e3fa2438>
```



Train Test Split

```
len(df)
```

```
340
```

```
325-12
```

```
313
```

```
train = df.iloc[:313]
test = df.iloc[313:]
```

```
len(test)
```

```
27
```

Scale Data

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
# IGNORE WARNING ITS JUST CONVERTING TO FLOATS
# WE ONLY FIT TO TRAININ DATA, OTHERWISE WE ARE CHEATING ASSUMING INFO ABOUT TEST SET
scaler.fit(train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

Time Series Generator

This class takes in a sequence of data-points gathered at equal intervals, along with time series parameters such as stride, length of history, etc., to produce batches for training/validation.

Arguments

```
    data: Indexable generator (such as list or Numpy array)
        containing consecutive data points (timesteps).
        The data should be at 2D, and axis 0 is expected
        to be the time dimension.
    targets: Targets corresponding to timesteps in `data`.
        It should have same length as `data`.
    length: Length of the output sequences (in number of timesteps).
    sampling_rate: Period between successive individual timesteps
        within sequences. For rate `r`, timesteps
        `data[i]`, `data[i-r]`, ... `data[i - length]`
        are used for create a sample sequence.
    stride: Period between successive output sequences.
        For stride `s`, consecutive output samples would
        be centered around `data[i]`, `data[i+s]`, `data[i+2*s]`, etc.
    start_index: Data points earlier than `start_index` will not be used
        in the output sequences. This is useful to reserve part of the
        data for test or validation.
    end_index: Data points later than `end_index` will not be used
        in the output sequences. This is useful to reserve part of the
        data for test or validation.
    shuffle: Whether to shuffle output samples,
        or instead draw them in chronological order.
    reverse: Boolean: if `true`, timesteps in each output sample will be
        in reverse chronological order.
    batch_size: Number of timeseries samples in each batch
        (except maybe the last one).
```

```
from keras.preprocessing.sequence import TimeseriesGenerator
```

```
Using TensorFlow backend.
```

```
scaled_train
```

```
array([[0.03662502],
       [0.03653945],
       [0.08309088],
       [0.13118261],
       [0.10183125],
       [0.12818757],
       [0.12279651],
       [0.09464316],
       [0.09370187],
       [0.10508301],
       [0.10345713],
       [0.16301557],
       [0.        ],
       [0.01968167],
       [0.09661133],
       [0.11518056],
       [0.10919048],
       [0.14247818],
       [0.12211193],
```

```
# define generator
n_input = 2
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

```
len(scaled_train)
```

```
313
```

```
len(generator) # n_input = 2
```

```
311
```

```
scaled_train
```

```
array([[0.03662502],
       [0.03653945],
       [0.08309088],
       [0.13118261],
       [0.10183125],
       [0.12818757],
       [0.12279651],
       [0.09464316],
       [0.09370187],
       [0.10508301],
       [0.10345713],
       [0.16301557],
       [0.        ],
       [0.01968167],
       [0.09661133],
       [0.11518056],
       [0.10919048],
       [0.14247818],
       [0.12211193],
```

```
# What does the first batch look like?
X,y = generator[0]
```

```
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.03662502 0.03653945]
Predict this y:
 [[0.08309088]]
```

```
# Let's redefine to get 12 months back and then predict the next month out
n_input = 12
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

```
# What does the first batch look like?
X,y = generator[0]
```

```
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.03662502 0.03653945 0.08309088 0.13118261 0.10183125 0.12818757
 0.12279651 0.09464316 0.09370187 0.10508301 0.10345713 0.16301557]
Predict this y:
 [[0.]]
```

Create the Model

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

```
%%time
# fit model
model.fit_generator(generator,epochs=50)
```

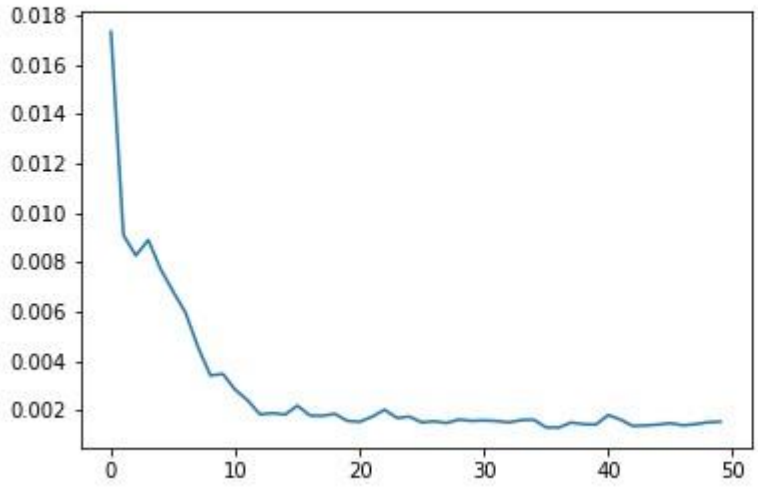
```
Epoch 1/50
301/301 [==============================] - 9s 32ms/step - loss: 0.0173
Epoch 2/50
301/301 [==============================] - 10s 33ms/step - loss: 0.0091
Epoch 3/50
301/301 [==============================] - 9s 29ms/step - loss: 0.0083
Epoch 4/50
301/301 [==============================] - 8s 27ms/step - loss: 0.0089
Epoch 5/50
301/301 [==============================] - 6s 21ms/step - loss: 0.0077
Epoch 6/50
301/301 [==============================] - 9s 32ms/step - loss: 0.0068
Epoch 7/50
301/301 [==============================] - 8s 28ms/step - loss: 0.0059
Epoch 8/50
301/301 [==============================] - 9s 31ms/step - loss: 0.0046
Epoch 9/50
301/301 [==============================] - 10s 32ms/step - loss: 0.0034
Epoch 10/50
```

```
model.history.history.keys()
```

```
dict_keys(['loss'])
```

```
1 loss_per_epoch = model.history.history['loss']
2 plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

```
[<matplotlib.lines.Line2D at 0x7fd0b01efda0>]
```



Evaluate on Test Data

```
1 first_eval_batch = scaled_train[-12:]
```

```
1 first_eval_batch
```

```
array([[0.63511895],
       [0.80865993],
       [0.72377203],
       [0.89902447],
       [1.        ],
       [0.71572822],
       [0.88499059],
       [0.75646072],
       [0.82423413],
       [0.87035769],
       [0.95498888],
       [0.55296937]])
```

```
1 first_eval_batch = first_eval_batch.reshape((1, n_input, n_features))
```

```
1 model.predict(first_eval_batch)
```

```
array([[0.6896863]], dtype=float32)
```

```
1 scaled_test[0]
```

```
array([0.62450796])
```

Now let's put this logic in a for loop to predict into the future for the entire test range.

```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))
```

```
current_batch.shape
```

```
(1, 12, 1)
```

```
current_batch
```

```
array([[[0.63511895],
        [0.80865993],
        [0.72377203],
        [0.89902447],
        [1.        ],
        [0.71572822],
        [0.88499059],
        [0.75646072],
        [0.82423413],
        [0.87035769],
        [0.95498888],
        [0.55296937]]])
```

```
np.append(current_batch[:,1:,:],[[[99]]],axis=1)
```

```
array([[[ 0.80865993],
        [ 0.72377203],
        [ 0.89902447],
        [ 1.        ],
        [ 0.71572822],
        [ 0.88499059],
        [ 0.75646072],
        [ 0.82423413],
        [ 0.87035769],
        [ 0.95498888],
        [ 0.55296937],
        [99.        ]]])
```

NOTE: PAY CLOSE ATTENTION HERE TO WHAT IS BEING OUTPUTED AND IN WHAT DIMENSIONS. ADD YOUR OWN PRINT() STATEMENTS TO SEE WHAT IS TRULY GOING ON!!

```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))

for i in range(len(test)):

    # get prediction 1 time stamp ahead ([0] is for grabbing just the number instead of [array])
    current_pred = model.predict(current_batch)[0]

    # store prediction
    test_predictions.append(current_pred)

    # update batch to now include prediction and drop first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

```
test_predictions
```

```
array([0.9014108], dtype=float32),
array([0.7660519], dtype=float32),
array([0.85677093], dtype=float32),
array([0.8933983], dtype=float32),
array([0.96434706], dtype=float32),
array([0.5751114], dtype=float32),
array([0.73173726], dtype=float32),
array([0.78904366], dtype=float32),
array([0.81055355], dtype=float32),
array([0.96692854], dtype=float32),
array([1.0079353], dtype=float32),
array([0.7764275], dtype=float32),
array([0.9184063], dtype=float32),
array([0.781284], dtype=float32),
array([0.89102334], dtype=float32),
array([0.91273236], dtype=float32),
array([0.97565484], dtype=float32),
array([0.60151213], dtype=float32),
array([0.7652707], dtype=float32),
array([0.79105306], dtype=float32),
array([0.8600352], dtype=float32)]
```

```
scaled_test
```

```
array([[0.62450796],
       [0.81619031],
       [0.75081294],
       [0.9396714 ],
       [0.97458497],
       [0.81182612],
       [0.94489132],
       [0.78726681],
       [0.91528324],
       [0.93428034],
       [1.05057334],
       [0.64906726],
       [0.67987335],
       [0.80788978],
       [0.86145816],
       [0.98767756],
       [0.96234811],
       [0.89448913],
       [0.95618689],
       [0.85238747],
       [0.97518398]
```

Inverse Transformations and Compare

```
true_predictions = scaler.inverse_transform(test_predictions)
```

```
1 true_predictions
```

```
[11958.95283413],
[13998.14089382],
[14746.24353027],
[11760.36095619],
[13564.88682032],
[11983.08236861],
[13043.22511923],
[13471.25235748],
[14300.35979569],
[ 9751.75169373],
[11582.0815742 ],
[12251.7642684 ],
[12503.12879372],
[14330.52693784],
[14809.73174715],
[12104.33185148],
[13763.49611175],
[12161.08452773],
[13443.49872601],
[13697.19039106],
[14432.50246572],
```

```
1 test
```

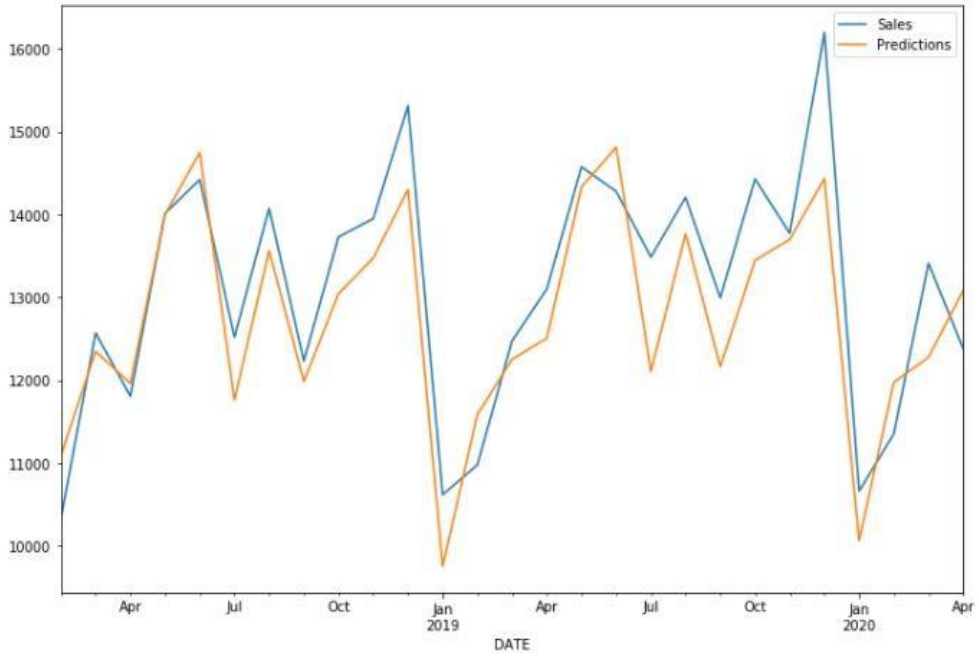
DATE	Sales
2018-02-01	10329
2018-03-01	12569
2018-04-01	11805
2018-05-01	14012
2018-06-01	14420
2018-07-01	12518
2018-08-01	14073
2018-09-01	12231
2018-10-01	13727
2018-11-01	13949
2018-12-01	15308
2019-01-01	10616

```
1 test
```

DATE	Sales	Predictions
2018-02-01	10329	11090.674083
2018-03-01	12569	12346.415679
2018-04-01	11805	11958.952834
2018-05-01	14012	13998.140894
2018-06-01	14420	14746.243530
2018-07-01	12518	11760.360956
2018-08-01	14073	13564.886820
2018-09-01	12231	11983.082369
2018-10-01	13727	13043.225119
2018-11-01	13949	13471.252357
2018-12-01	15308	14300.359796
2019-01-01	10616	9751.751694

```
1 test.plot(figsize=(12,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd0b01624a8>
```



Saving and Loading Models

```
1 model.save('my_rnn_model.h5')
```

load a model

```
1 from keras.models import load_model
2 new_model = load_model('my_rnn_model.h5')
```

```
1 new_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

EXPERIMENT NO. 17

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 05-05-2021
Faculty Signature:
Marks:

Objective(s): To train a RNN model for predicting miles travelled by vehicles
Outcome: Learning to train RNN model
Problem Statement: To build a RNN Model for predicting miles travelled by vehicles.
Background Study: Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras. Unlike regression predictive modeling, time series also adds the complexity of a sequence dependence among the input variables. A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks

Student Work Area

Code/Sample Outputs:

TASK: IMPORT THE BASIC LIBRARIES YOU THINK YOU WILL USE

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

Data

Info about this data set: https://fred.stlouisfed.org/series/TRFVOLUSM227NFWA

Read in the data set "Miles_Traveled.csv" from the Data folder. Figure out how to set the date to a datetime index columns

```
data=pd.read_csv("Miles_Traveled.csv ", index_col='DATE', parse_dates=True)
data.index.freq = 'MS'
```

```
data.head()
```

DATE	TRFVOLUSM227NFWA
1970-01-01	80173.0
1970-02-01	77442.0
1970-03-01	90223.0
1970-04-01	89956.0
1970-05-01	97972.0

Task: Change the column names to Value

```
data.columns = ['value']
```

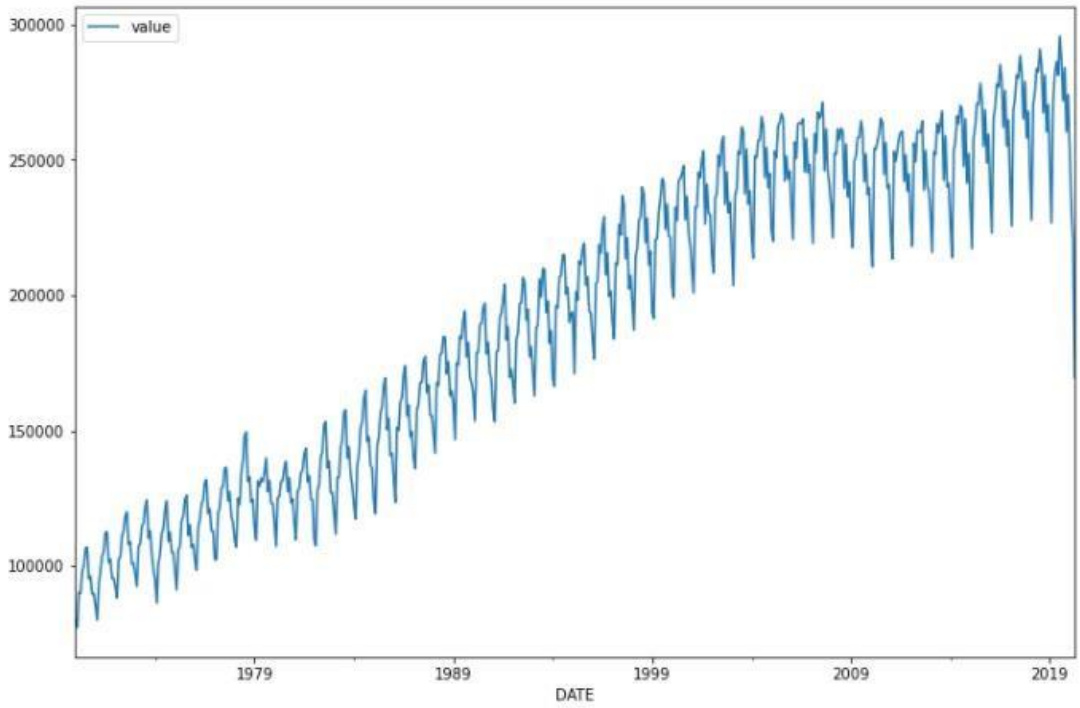
```
data.head()
```

DATE	value
1970-01-01	80173.0
1970-02-01	77442.0
1970-03-01	90223.0
1970-04-01	89956.0
1970-05-01	97972.0

TASK: Plot out the time series

```
1 data.plot(figsize=(12,8))
```

```
<AxesSubplot:xlabel='DATE'>
```



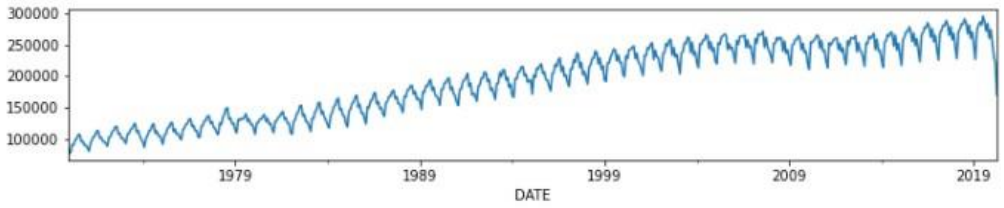
TASK: Perform a Seasonal Decomposition on the model and plot out the ETS components

```
1 from statsmodels.tsa.seasonal import seasonal_decompose
```

```
1 results = seasonal_decompose(data['value'])
2 results.observed.plot(figsize=(12,2))
```

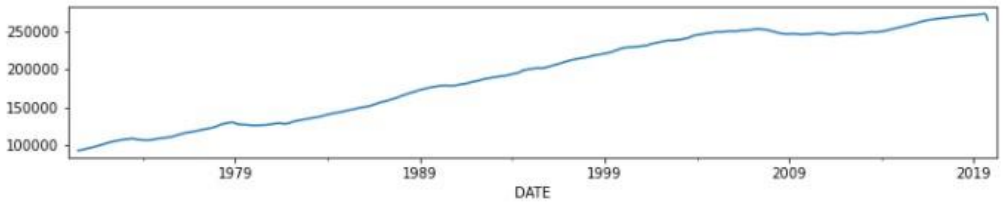
```
C:\Users\HP\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:49: FutureWarning: The Panel class is remov
Accessing it from the top-level namespace will also be removed in the next version
  data_klasses = (pandas.Series, pandas.DataFrame, pandas.Panel)
```

```
<AxesSubplot:xlabel='DATE'>
```



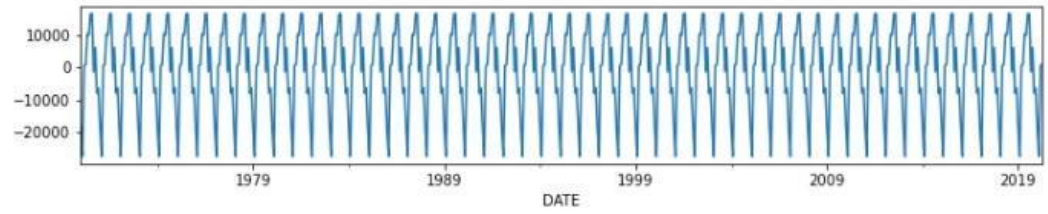

```
1 results.trend.plot(figsize=(12,2))
```

```
<AxesSubplot:xlabel='DATE'>
```



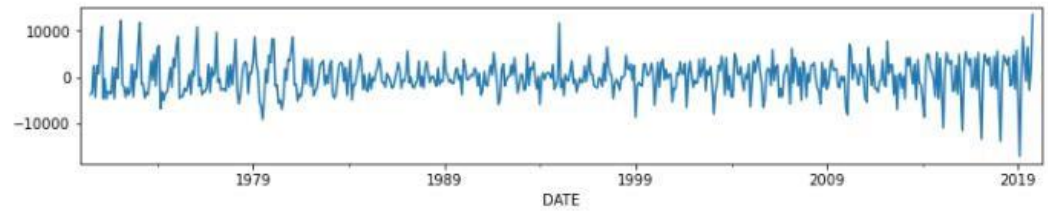
```
1 results.seasonal.plot(figsize=(12,2))
```

```
<AxesSubplot:xlabel='DATE'>
```




```
1 results.resid.plot(figsize=(12,2))
```

```
<AxesSubplot:xlabel='DATE'>
```



Train Test Split

TASK: Figure out the length of the data set

```
1 len(data)
```

```
604
```

```
1 604-16
```

```
588
```

TASK: Split the data into a train/test split where the test set is the last 12 months of data.

```
1 train = data.iloc[:588]
2 test = data.iloc[588:]
```

```
1 len(test)
```

```
16
```

Scale Data

TASK: Use a MinMaxScaler to scale the train and test sets into scaled versions.

```
1 from sklearn.preprocessing import MinMaxScaler
```

```
1 scaler = MinMaxScaler()
```

```
1 scaler.fit(train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
1 scaled_train = scaler.transform(train)
2 scaled_test = scaler.transform(test)
```

Time Series Generator

TASK: Create a TimeSeriesGenerator object based off the scaled_train data. The n_input is up to you, but at a minimum it should be at least 12.

```
from keras.preprocessing.sequence import TimeseriesGenerator
```

```
scaled_train
```

```
array([[0.01278875],
       [0.        ],
       [0.05985099],
       [0.05860068],
       [0.09613809],
       [0.10579872],
       [0.13556735],
       [0.13826933],
       [0.08341021],
       [0.08838804],
       [0.05732696],
       [0.05838996],
       [0.0369661 ],
       [0.0125312 ],
       [0.0727334 ],
       [0.09676558],
       [0.1227505 ],
       [0.13107653],
       [0.16401073],
       [0.16483491],
       [0.11167565]
```

```
# define generator
n_input = 2
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

```
len(scaled_train)
```

```
588
```

```
len(generator)
```

```
586
```

```
scaled_train
```

```
array([[0.01278875],
       [0.        ],
       [0.05985099],
       [0.05860068],
       [0.09613809],
       [0.10579872],
       [0.13556735],
       [0.13826933],
       [0.08341021],
       [0.08838804],
       [0.05732696],
       [0.05838996],
       [0.0369661 ],
       [0.0125312 ],
       [0.0727334 ],
       [0.09676558],
       [0.1227505 ],
       [0.13107653],
       [0.16401073],
       [0.16483491],
       [0.11167565]
```

```
X,y = generator[0]
```

```
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')

```

```
Given the Array:
[0.01278875 0.        ]
Predict this y:
 [[0.05985099]]
```

```
n_input = 12
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

```
X,y = generator[0]
```

```
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.01278875 0.         0.05985099 0.05860068 0.09613809 0.10579872
 0.13556735 0.13826933 0.08341021 0.08838804 0.05732696 0.05838996]
Predict this y:
 [[0.0369661]]
```

Create the Model

TASK: Create a Keras Sequential Model with as many LSTAM units you want and a final Dense Layer.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               40800
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

TASK: Fit the model to the generator (it should be a lot of epochs, but do as many as you have the patience for! :)

```
%%time
# fit model
model.fit_generator(generator,epochs=50)

```

```
WARNING:tensorflow:From <timed eval>:2: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and
will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/50
576/576 [==============================] - 3s 4ms/step - loss: 0.0151
Epoch 2/50
576/576 [==============================] - 2s 4ms/step - loss: 0.0053
Epoch 3/50
576/576 [==============================] - 2s 4ms/step - loss: 0.0040
Epoch 4/50
576/576 [==============================] - 2s 4ms/step - loss: 0.0030
Epoch 5/50
576/576 [==============================] - 3s 5ms/step - loss: 0.0027
Epoch 6/50
576/576 [==============================] - 2s 4ms/step - loss: 0.0024
Epoch 7/50
576/576 [==============================] - 2s 4ms/step - loss: 0.0019
Epoch 8/50
576/576 [==============================] - 2s 4ms/step - loss: 0.0015
Epoch 9/50
```

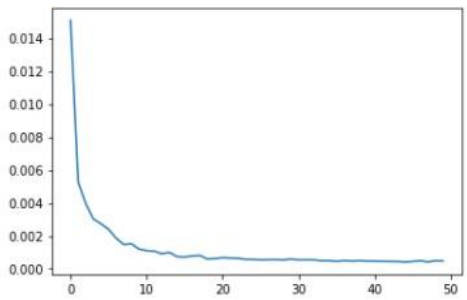
TASK: Plot the history of the loss that occured during training.

```
model.history.history.keys()
```

```
dict_keys(['loss'])
```

```
loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

```
[<matplotlib.lines.Line2D at 0x2c8f7581e80>]
```



Evaluate on Test Data

TASK: Based on your test data and input size, create an appropriate;y sized "first evaluation batch" like we did in the lecture.

```
first_eval_batch = scaled_train[-12:]
```

```
first_eval_batch
```

```
array([[0.78340599],
       [0.70390593],
       [0.90501388],
       [0.92572127],
       [0.96592788],
       [0.96094068],
       [1.        ],
       [0.97190314],
       [0.88969641],
       [0.95501225],
       [0.85709937],
       [0.90344514]])
```

```
first_eval_batch = first_eval_batch.reshape((1, n_input, n_features))
```

TASK: Generate predictions into the same time stamps as the test set

```
model.predict(first_eval_batch)
```

```
array([[0.8145201]], dtype=float32)
```

```
scaled_test[0]
```

```
array([0.79993163])
```

```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))
```

```
current_batch.shape
```

```
(1, 12, 1)
```

```
current_batch
```

```
array([[[0.78340599],
        [0.70390593],
        [0.90501388],
        [0.92572127],
        [0.96592788],
        [0.96094068],
        [1.        ],
        [0.97190314],
        [0.88969641],
        [0.95501225],
        [0.85709937],
        [0.90344514]]])
```

```
np.append(current_batch[:,1:,:],[[[99]]],axis=1)
```

```
array([[[ 0.70390593],
        [ 0.90501388],
        [ 0.92572127],
        [ 0.96592788],
        [ 0.96094068],
        [ 1.        ],
        [ 0.97190314],
        [ 0.88969641],
        [ 0.95501225],
        [ 0.85709937],
        [ 0.90344514],
        [99.        ]]])
```

```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))

for i in range(len(test)):
    # get prediction 1 time stamp ahead ([0] is for grabbing just the number instead of [array])
    current_pred = model.predict(current_batch)[0]
    # store prediction
    test_predictions.append(current_pred)
    # update batch to now include prediction and drop first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

```
test_predictions
```

```
[array([0.8145201], dtype=float32),
 array([0.7465621], dtype=float32),
 array([0.92984426], dtype=float32),
 array([0.9550873], dtype=float32),
 array([1.0013753], dtype=float32),
 array([0.9949056], dtype=float32),
 array([1.0326661], dtype=float32),
 array([1.0017182], dtype=float32),
 array([0.928159], dtype=float32),
 array([0.9822377], dtype=float32),
 array([0.89078593], dtype=float32),
 array([0.93344533], dtype=float32),
 array([0.8453373], dtype=float32),
 array([0.7876327], dtype=float32),
 array([0.95467985], dtype=float32),
 array([0.98394156], dtype=float32)]
```

```
scaled_test
```

```
array([[0.79993163],
       [0.69911542],
       [0.90905515],
       [0.95587388],
       [0.97756934],
       [0.95497947],
       [1.02189214],
       [0.98043522],
       [0.9103757 ],
       [0.9675294 ],
       [0.85725859],
       [0.92098227],
       [0.82460536],
       [0.72391558],
       [0.67099983],
       [0.43167078]])
```

Inverse Transformations and Compare

TASK: Inverse Transform your new forecasted predictions.

```
true_predictions = scaler.inverse_transform(test_predictions)
```

```
true_predictions
```

```
array([[251380.32819819],
       [236868.10174406],
       [276007.45223629],
       [281398.02853191],
       [291282.69494188],
       [289901.10424364],
       [297964.74491274],
       [291355.90865886],
       [275647.56965041],
       [287195.91340601],
       [267666.66353631],
       [276776.45083249],
       [257961.23835766],
       [245638.60099435],
       [281311.01723683],
       [287559.76725054]])
```

```
test
```

DATE	value
2019-01-01	248265.0
2019-02-01	226736.0
2019-03-01	271568.0
2019-04-01	281566.0
2019-05-01	286199.0
2019-06-01	281375.0
2019-07-01	295664.0
2019-08-01	286811.0
2019-09-01	271850.0
2019-10-01	284055.0
2019-11-01	260507.0
2019-12-01	274415.0

TASK: Create a new dataframe that has both the original test values and your predictions for them.

```
1 test['Predictions'] = true_predictions
```

```
C:\Users\HP\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
rsus-a-copy
  """Entry point for launching an IPython kernel.
```

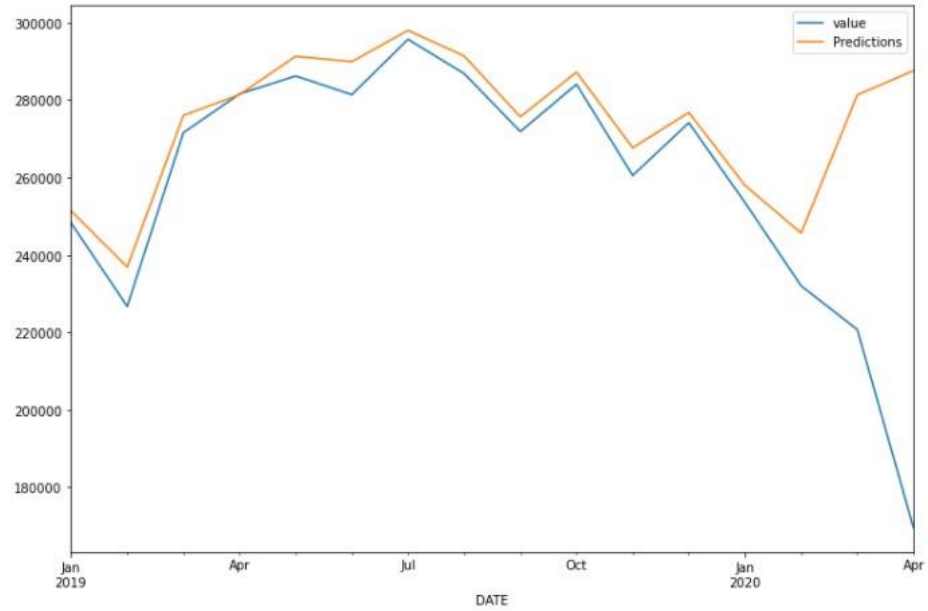
```
1 test
```

DATE	value	Predictions
2019-01-01	248265.0	251380.328198
2019-02-01	226736.0	236868.101744
2019-03-01	271568.0	276007.452236
2019-04-01	281566.0	281398.028532
2019-05-01	286199.0	291282.694942
2019-06-01	281375.0	289901.104244
2019-07-01	295664.0	297964.744913
2019-08-01	286811.0	291355.908659
2019-09-01	271850.0	275647.569650
2019-10-01	284055.0	287195.913406
2019-11-01	260507.0	267666.663536
2019-12-01	274115.0	276776.450832

TASK: Plot out the test set against your own predicted values.

```
1 test.plot(figsize=(12,8))
```

```
<AxesSubplot:xlabel='DATE'>
```



Saving Models

TASK: Optional, Save your model!

```
1 model.save('my_rnn_model.h5')
```

EXPERIMENT NO. 18

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 7-04-2021
Faculty Signature:
Marks:

Objective(s): To perform tokenization and stemming on text data using NLTK.
Outcome: Learning to implement tokenization and stemming on text data.
Problem Statement: To implement tokenization and stemming on text data.
Background Study: **Tokenization**: In Python tokenization basically refers to splitting up a larger body of text into smaller lines, words or even creating words for a non-English language. The various tokenization functions in-built into the nltk module itself and can be used in programs as shown below. **Stemming**: Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.

Student Work Area

Code/Sample Outputs:

Tokenization

```
import numpy
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
```

```
True
```

```
sent = sent_tokenize(" Before the extensive use of deep learning, many problems are solved using machine learning but did no
print(sent)
```

```
[' Before the extensive use of deep learning, many problems are solved using machine learning but did not get the results as ex
pected.', 'With the boom in technology specially in Graphical Processing Units(GPUs), deep learning comes into picture and solv
ed many complicated problems with the help of artificial neural networks.', 'One of the most important aspect of deep learning
is that no feature extraction is required.', 'The model itself manages the weight and extract the best features and then traini
ng is done whereas in machine learning, feature extraction is always done prior model training.']
```

```
len(sent)
```

```
4
```

```
word = word_tokenize('Many problems are solved using machine learning')

print(word)
```

```
['Many', 'problems', 'are', 'solved', 'using', 'machine', 'learning']
```

```
len(word)
```

```
7
```

```
from nltk.tokenize.punkt import PunktSentenceTokenizer
```

```
pst = PunktSentenceTokenizer()
```

```
punkt_sentence = pst.tokenize("Before the extensive use of deep learning, many problems are solved using machine learning bu
print(punkt_sentence)
```

```
['Before the extensive use of deep learning, many problems are solved using machine learning but did not get the results as exp
ected.', 'With the boom in technology specially in Graphical Processing Units(GPUs), deep learning comes into picture and solve
d many complicated problems with the help of artificial neural networks.', 'One of the most important aspect of deep learning i
s that no feature extraction is required.', 'The model itself manages the weight and extract the best features and then trainin
g is done whereas in machine learning, feature extraction is always done prior model training.']
```

```
len(punkt_sentence)
```

```
4
```

```
span_sentence = pst.span_tokenize("Before the extensive use of deep learning, many problems are solved using machine learnin
print(list(span_sentence))
```

```
[(0, 131), (132, 321), (322, 414), (415, 591)]
```

```
sentences = pst.sentences_from_tokens(word)
list(sentences)
```

```
[['Many', 'problems', 'are', 'solved', 'using', 'machine', 'learning']]
```

```
from nltk.probability import FreqDist
```

```
freq = FreqDist(word)
freq.most_common()
```

```
[('Many', 1),
 ('problems', 1),
 ('are', 1),
 ('solved', 1),
 ('using', 1),
 ('machine', 1),
 ('learning', 1)]
```

Stemming

Often when searching text for a certain keyword, it helps if the search returns variations of the word. For instance, searching for "boat" might also return "boats" and "boating". Here, "boat" would be the stem for [boat, boater, boating, boats].

Stemming is a somewhat crude method for cataloging related words; it essentially chops off letters from the end until the stem is reached. This works fairly well in most cases, but unfortunately English has many exceptions where a more sophisticated process is required. In fact, spaCy doesn't include a stemmer, opting instead to rely entirely on lemmatization. For those interested, there's some background on this decision here. We discuss the virtues of lemmatization in the next section.

Porter Stemmer

One of the most common - and effective - stemming tools is Porter's Algorithm developed by Martin Porter in 1980. The algorithm employs five phases of word reduction, each with its own set of mapping rules.

```
import nltk
from nltk.stem.porter import *
```

```
p_stemmer = PorterStemmer()
```

```
words = ['run', 'runner', 'running', 'ran', 'runs', 'easily', 'fairly', 'accordingly', 'consolingly']
```

```
for word in words:
  print(word+' -->  '+p_stemmer.stem(word))
```

```
run -->  run
runner -->  runner
running -->  run
ran -->  ran
runs -->  run
easily -->  easili
fairly -->  fairli
accordingly -->  accordingli
consolingly -->  consolingli
```

Snowball Stemmer

This is somewhat of a misnomer, as Snowball is the name of a stemming language developed by Martin Porter. The algorithm used here is more acurately called the "English Stemmer" or "Porter2 Stemmer". It offers a slight improvement over the original Porter stemmer, both in logic and speed. Since nltk uses the name SnowballStemmer, we'll use it here.

```
from nltk.stem.snowball import SnowballStemmer
s_stemmer = SnowballStemmer(language='english')
```

```
words = ['run', 'runner', 'running', 'ran', 'runs', 'easily', 'fairly', 'accordingly', 'consolingly']
```

```
for word in words:
  print(word+' -->  '+s_stemmer.stem(word))
```

```
run -->  run
runner -->  runner
running -->  run
ran -->  ran
runs -->  run
easily -->  easili
fairly -->  fair
accordingly -->  accord
consolingly -->  consol
```

EXPERIMENT NO. 19

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 7-04-2021
Faculty Signature:
Marks:

Objective(s): To perform lemmatization and remove stopwords on text data using NLTK.
Outcome: Learning to implement lemmatization and stopwords on text data.
Problem Statement: To apply lemmatization and remove stopwords.
Background Study: **Lemmatization**: *Lemmatization* is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. *Lemmatization* is similar to stemming but it brings context to the words. So it links words with similar meaning to one word. **Stopwords**: In computing, stop words are words that are filtered out before or after the natural language data (text) are processed. While "stop words" typically refers to the most common words in a language, all-natural language processing tools don't use a single universal list of stop words.

Student Work Area

Code/Sample Outputs:

Lemmatizing Words Using Wordnet

```
import nltk
from nltk.stem import *
import pandas as pd
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\wordnet.zip.
```

```
True
```

Lemmatizing Words

```
from nltk.stem import WordNetLemmatizer

wnl = WordNetLemmatizer()
print(wnl.lemmatize('definitions'))
```

```
definition
```

Lemmatizing words by specifying parts-of-speech

```
print('Adjective: ', wnl.lemmatize('running', pos='a'))
print('Adverb: ', wnl.lemmatize('running', pos='r'))
print('Noun: ', wnl.lemmatize('running', pos='n'))
print('Verb: ', wnl.lemmatize('running', pos='v'))
```

```
Adjective:  running
Adverb:  running
Noun:  running
Verb:  run
```

```
input_tokens = ['dictionaries', 'dictionary',
                'hushed', 'hush', 'hushing',
                'functional', 'functionally',
                'lying', 'lied', 'lies',
                'flawed', 'flaws', 'flawless',
                'friendship', 'friendships', 'friendly', 'friendless',
                'definitions', 'definition', 'definitely',
                'the', 'these', 'those',
                'motivational', 'motivate', 'motivating']
```

```
ss =  SnowballStemmer('english')

ss_stemmed_tokens = []
for token in input_tokens:
    ss_stemmed_tokens.append(ss.stem(token))
```

```
wnl_lemmatized_tokens = []
for token in input_tokens:
    wnl_lemmatized_tokens.append(wnl.lemmatize(token, pos='v'))
```

```
stems_lemmas_df = pd.DataFrame({
    'words': input_tokens,
    'Snowball Stemmer': ss_stemmed_tokens,
    'WordNet Lemmatizer': wnl_lemmatized_tokens
})
stems_lemmas_df
```

	words	Snowball Stemmer	WordNet Lemmatizer
0	dictionaries	dictionari	dictionaries
1	dictionary	dictionari	dictionary
2	hushed	hush	hush
3	hush	hush	hush
4	hushing	hush	hush
5	functional	function	functional
6	functionally	function	functionally
7	lying	lie	lie
8	lied	lie	lie
9	lies	lie	lie
10	flawed	flaw	flaw
11	flaws	flaw	flaw
12	flawless	flawless	flawless

```
from nltk.tokenize import word_tokenize

with open('DLdata.txt', 'r') as f:
    file_contents = f.read()
print(file_contents)
```

```
The advent of computer graphic processing units, improvement in mathematical models and availability of big data has allowed ar
tificial intelligence (AI) using machine learning (ML) and deep learning (DL) techniques to achieve robust performance for broa
d applications in social-media, the internet of things, the automotive industry and healthcare. DL systems in particular provid
e improved capability in image, speech and motion recognition as well as in natural language processing. In medicine, significa
nt progress of AI and DL systems has been demonstrated in image-centric specialties such as radiology, dermatology, pathology a
nd ophthalmology. New studies, including pre-registered prospective clinical trials, have shown DL systems are accurate and eff
ective in detecting diabetic retinopathy (DR), glaucoma, age-related macular degeneration (AMD), retinopathy of prematurity, re
fractive error and in identifying cardiovascular risk factors and diseases, from digital fundus photographs. There is also incr
easing attention on the use of AI and DL systems in identifying disease features, progression and treatment response for retina
l diseases such as neovascular AMD and diabetic macular edema using optical coherence tomography (OCT). Additionally, the appli
cation of ML to visual fields may be useful in detecting glaucoma progression. There are limited studies that incorporate clini
cal data including electronic health records, in AL and DL algorithms, and no prospective studies to demonstrate that AI and DL
algorithms can predict the development of clinical eye disease. This article describes global eye disease burden, unmet needs a
nd common conditions of public health importance for which AI and DL systems may be applicable. Technical and clinical aspects
to build a DL system to address those needs, and the potential challenges for clinical adoption are discussed. AI, ML and DL wi
ll likely play a crucial role in clinical ophthalmology practice, with implications for screening, diagnosis and follow up of t
he major causes of vision impairment in the setting of ageing populations globally.
```

```
word_tokens = word_tokenize(file_contents)
```

```
wnl = WordNetLemmatizer()
lemmatized_words = []

for word in word_tokens:
    lemmatized_words.append(wnl.lemmatize(word, pos="v"))
```

```
" ".join(lemmatized_words)
```

```
'The advent of computer graphic process units , improvement in mathematical model and availability of big data have allow artif
icial intelligence ( AI ) use machine learn ( ML ) and deep learn ( DL ) techniques to achieve robust performance for broad app
lications in social-media , the internet of things , the automotive industry and healthcare . DL systems in particular provide
improve capability in image , speech and motion recognition as well as in natural language process . In medicine , significant
progress of AI and DL systems have be demonstrate in image-centric specialties such as radiology , dermatology , pathology and
ophthalmology . New study , include pre-registered prospective clinical trials , have show DL systems be accurate and effective
in detect diabetic retinopathy ( DR ) , glaucoma , age-related macular degeneration ( AMD ) , retinopathy of prematurity , refr
active error and in identify cardiovascular risk factor and diseases , from digital fundus photograph . There be also increase
attention on the use of AI and DL systems in identify disease feature , progression and treatment response for retinal diseases
such as neovascular AMD and diabetic macular edema use optical coherence tomography ( OCT ) . Additionally , the application of
ML to visual field may be useful in detect glaucoma progression . There be limit study that incorporate clinical data include e
lectronic health record , in AL and DL algorithms , and no prospective study to demonstrate that AI and DL algorithms can predi
ct the development of clinical eye disease . This article describe global eye disease burden , unmet need and common condition
of public health importance for which AI and DL systems may be applicable . Technical and clinical aspects to build a DL system
to address those need , and the potential challenge for clinical adoption be discuss . AI , ML and DL will likely play a crucia
l role in clinical ophthalmology practice , with implications for screen , diagnosis and follow up of the major cause of vision
impairment in the set of age populations globally .'
```

Stopwords

```
from nltk import word_tokenize
from nltk.corpus import stopwords

import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

```
True
```

```
print(stopwords.fileids())
```

```
['arabic', 'azerbaijani', 'danish', 'dutch', 'english', 'finnish', 'french', 'german', 'greek', 'hungarian', 'indonesian', 'ita
lian', 'kazakh', 'nepali', 'norwegian', 'portuguese', 'romanian', 'russian', 'slovene', 'spanish', 'swedish', 'tajik', 'turkis
h']
```

```
print("English: ",stopwords.words('english'))
print("Arabic: ",stopwords.words('arabic'))
```

```
English:  ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doi
ng', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'ou
t', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
ny', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've',
'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shou
ldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
Arabic:  ['إذ', 'إذا', 'إذما', 'إذن', 'أف', 'أقل', 'أكثر', 'ألا', 'إلا', 'التي', 'الذي', 'الذين', 'اللاتي', 'اللائي', 'اللتان', 'اللتيا', 'اللتين', 'اللذان', 'اللذين',
ن', 'اللواتي', 'إلى', 'إليك', 'إليكم', 'إليكما', 'إليكن', 'أم', 'أما', 'أما', 'إما', 'أن', 'إن', 'إنا', 'أنا', 'أنت', 'أنتم', 'أنتما', 'أنتن', 'إنما', 'إنه', 'أنى',
ى', 'آه', 'آها', 'أو', 'أولاء', 'أولئك', 'أوه', 'آي', 'أي', 'أيها', 'إي', 'أين', 'أينما', 'إيه', 'بخ', 'بس', 'بعد', 'بعض', 'بك', 'بكم',
'بكم', 'بكما', 'بكن', 'بل', 'بلى', 'بما', 'بماذا', 'بمن', 'بنا', 'به', 'بها', 'بهما', 'بهم', 'بهن', 'بي', 'بين', 'بيد', 'تلك', 'تلكم', 'تلكما', 'ته', 'تي',
'تين', 'تينك', 'ثم', 'ثمة', 'حاشا', 'حبذا', 'حتى', 'حيث', 'حيثما', 'حين', 'خلا', 'دون', 'ذا', 'ذات', 'ذاك', 'ذان', 'ذانك', 'ذلك', 'ذلكم', 'ذلكما', 'ذلك
ن', 'ذه', 'ذو', 'ذوا', 'ذواتا', 'ذواتي', 'ذي', 'ذين', 'ذينك', 'ريث', 'سوف', 'سوى', 'شتان', 'عدا', 'عسى', 'عل', 'على', 'عليك', 'عليه', 'عما', 'ع
ن', 'عند', 'غير', 'فإذا', 'فإن', 'فلا', 'فمن', 'في', 'فيم', 'فيما', 'فيه', 'فيها', 'قد', 'كأن', 'كأنما', 'كأي', 'كأين', 'كذا', 'كذلك', 'كل', 'كلا', 'كلاهم
ا', 'كلتا', 'كلما', 'كليكما', 'كليهما', 'كم', 'كم', 'كما', 'كي', 'كيت', 'كيف', 'كيفما', 'لا', 'لاسيما', 'لدى', 'لست', 'لستم', 'لستما', 'لستن', 'لسن', 'لسن
ا', 'لعل', 'لك', 'لكم', 'لكما', 'لكن', 'لكنما', 'لكي', 'لكيلا', 'لم', 'لما', 'لن', 'لنا', 'له', 'لها', 'لهم', 'لهما', 'لهن', 'لو', 'لولا', 'لوما', 'لي', 'لا
ن', 'ليت', 'ليس', 'ليسا', 'ليست', 'ليستا', 'ليسوا', 'ما', 'ماذا', 'متى', 'مذ', 'مع', 'مما', 'ممن', 'من', 'منه', 'منها', 'منذ', 'مه', 'مهما', 'نحن', 'ه
حو', 'نعم', 'ها', 'هاتان', 'هاته', 'هاتي', 'هاتين', 'هاك', 'هاهنا', 'هذا', 'هذان', 'هذه', 'هذي', 'هذين', 'هكذا', 'هل', 'هلا', 'هم', 'هما', 'هن', 'هذ
ا', 'هناك', 'هنالك', 'هو', 'هؤلاء', 'هي', 'هيا', 'هيت', 'هيهات', 'والذي', 'والذين', 'وإذ', 'وإذا', 'وإن', 'ولا', 'ولكن', 'ولو', 'وما', 'ومن', 'وهو', 'ي
ا']
```

```
text_array = ["A bird in hand is worth two in the bush.",
              "Good things come to those who wait.",
              "These watches cost $1500! ",
              "There are other fish in the sea.",
              "The ball is in your court.",
              "Mr. Smith Goes to Washington ",
              "Doogie Howser M.D."]

text = " ".join(text_array)
text
```

```
'A bird in hand is worth two in the bush. Good things come to those who wait. These watches cost $1500!  There are other fish i
n the sea. The ball is in your court. Mr. Smith Goes to Washington  Doogie Howser M.D.'
```

```
word_tokens = word_tokenize(text)

word_tokens
```

```
['A',
 'bird',
 'in',
 'hand',
 'is',
 'worth',
 'two',
 'in',
 'the',
 'bush',
 '.',
 'Good',
 'things',
 'come',
 'to',
 'those',
 'who',
 'wait',
 '.',
 'These',
 'watches'
```

```
stop_words = set(stopwords.words('english'))

filtered_words = []

for word in word_tokens:
    if word not in stop_words:
        filtered_words.append(word)

print(filtered_words)
```

```
['A', 'bird', 'hand', 'worth', 'two', 'bush', '.', 'Good', 'things', 'come', 'wait', '.', 'These', 'watches', 'cost', '$', '150
0', '!', 'There', 'fish', 'sea', '.', 'The', 'ball', 'court', '.', 'Mr.', 'Smith', 'Goes', 'Washington', 'Doogie', 'Howser',
'M.D', '.']
```

```
with open("DLdata.txt", "w") as f:
    for word in filtered_words:
        f.write(word)
        f.write(' ')
```

```
with open("DLdata.txt", "r") as f:
    file_contents = f.read()

print(file_contents)
```

```
A bird hand worth two bush . Good things come wait . These watches cost $ 1500 ! There fish sea . The ball court . Mr. Smith Go
es Washington Doogie Howser M.D .
```

```
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer()
count_vectorizer.fit([file_contents])
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

```
transformed_vector = count_vectorizer.transform(text_array)

transformed_vector.shape
```

```
(7, 25)
```

```
feature_names_nltk = count_vectorizer.get_feature_names()
```

```
count_vectorizer.vocabulary_
```

```
{'bird': 2,
 'hand': 11,
 'worth': 24,
 'two': 20,
 'bush': 3,
 'good': 10,
 'things': 19,
 'come': 4,
 'wait': 21,
 'these': 18,
 'watches': 23,
 'cost': 5,
 '1500': 0,
 'there': 17,
 'fish': 8,
 'sea': 14,
 'the': 16,
 'ball': 1,
 'court': 6,
 'mr': 13,
 'smith': 15
```

```
transformed_vector.toarray()
```

```
array([[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
        0, 0, 1],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
        0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
        0, 0, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
        1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0]], dtype=int64)
```

```
count_vectorizer.inverse_transform(transformed_vector)
```

```
[array(['bird', 'bush', 'hand', 'the', 'two', 'worth'], dtype='<U10'),
 array(['come', 'good', 'things', 'wait'], dtype='<U10'),
 array(['1500', 'cost', 'these', 'watches'], dtype='<U10'),
 array(['fish', 'sea', 'the', 'there'], dtype='<U10'),
 array(['ball', 'court', 'the'], dtype='<U10'),
 array(['goes', 'mr', 'smith', 'washington'], dtype='<U10'),
 array(['doogie', 'howser'], dtype='<U10')]
```

Removing Stpwords Using sklearn

```
count_vectorizer = CountVectorizer(stop_words='english')

transformed_vector = count_vectorizer.fit_transform(text_array)

transformed_vector.shape
```

```
(7, 21)
```

```
feature_names_sklearn = count_vectorizer.get_feature_names()
```

```
transformed_vector.toarray()
```

```
array([[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 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, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]],
      dtype=int64)
```

```
count_vectorizer.inverse_transform(transformed_vector)
```

```
[array(['bird', 'hand', 'worth', 'bush'], dtype='<U10'),
 array(['good', 'things', 'come', 'wait'], dtype='<U10'),
 array(['watches', 'cost', '1500'], dtype='<U10'),
 array(['fish', 'sea'], dtype='<U10'),
 array(['ball', 'court'], dtype='<U10'),
 array(['mr', 'smith', 'goes', 'washington'], dtype='<U10'),
 array(['doogie', 'howser'], dtype='<U10')]
```

Set Difference of Both

```
def set_diff(first, second):
        second = set(second)
        return [item for item in first if item not in second]
```

```
set_diff(feature_names_sklearn, feature_names_nltk)
```

```
[]
```

```
set_diff(feature_names_nltk, feature_names_sklearn)
```

```
['the', 'there', 'these', 'two']
```

Filtering words based on frequency

```
from sklearn.datasets import fetch_20newsgroups

newsgroups = fetch_20newsgroups(subset='train')
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
```

```
1 newsgroups.keys()
```

```
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])
```

```
1 print(newsgroups.data[0])
```

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

```
1 newsgroups.target_names
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

```
1 count_vectorizer = CountVectorizer()
2
3 transformed_vector = count_vectorizer.fit_transform(newsgroups.data)
4
5 transformed_vector.shape
```

```
(11314, 130107)
```

EXPERIMENT NO. 20

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 7-04-2021
Faculty Signature:
Marks:

Objective(s): To explore the part-of-speech(POS) tagging and Named Entity Recognition on text data using NLTK a) To extract all nouns from the given text b) To extract all verbs from the given text
Outcome: Learning to implement POS on text data.
Problem Statement: To build POS using NLTK on text data.
Background Study: **POS**: It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

Student Work Area

Code/Sample Outputs:

Named Entity Recognition (NER)

spaCy has an **'ner'** pipeline component that identifies token spans fitting a predetermined set of named entities. These are available as the `ents` property of a `Doc` object.

```
# Perform standard imports
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
# Write a function to display basic entity info:
def show_ents(doc):
    if doc.ents:
        for ent in doc.ents:
            print(ent.text+' - '+ent.label_+' - '+str(spacy.explain(ent.label_)))
    else:
        print('No named entities found.')
```

```
doc = nlp(u'May I go to Washington, DC next May to see the Washington Monument?')

show_ents(doc)
```

```
Washington - GPE - Countries, cities, states
next May - DATE - Absolute or relative dates or periods
the Washington Monument - ORG - Companies, agencies, institutions, etc.
```

Here we see tokens combine to form the entities `Washington, DC`, `next May` and `the Washington Monument`

```
doc = nlp(u'Can I please borrow 500 dollars from you to buy some Microsoft stock?')

for ent in doc.ents:
    print(ent.text, ent.start, ent.end, ent.start_char, ent.end_char, ent.label_)
```

```
500 dollars 4 6 20 31 MONEY
Microsoft 11 12 53 62 ORG
```

Adding a Named Entity to a Span

Normally we would have spaCy build a library of named entities by training it on several samples of text.
In this case, we only want to add one value:

```
doc = nlp(u'TATA to build a U.K. factory for $6 million')

show_ents(doc)
```

```
U.K. - GPE - Countries, cities, states
$6 million - MONEY - Monetary values, including unit
```

Right now, spaCy does not recognize "TATA" as a company.

```
from spacy.tokens import Span

# Get the hash value of the ORG entity label
ORG = doc.vocab.strings[u'ORG']

# Create a Span for the new entity
new_ent = Span(doc, 0, 1, label=ORG)

# Add the entity to the existing Doc object
doc.ents = list(doc.ents) + [new_ent]
```

In the code above, the arguments passed to `Span()` are:

- `doc` - the name of the Doc object
- `0` - the *start* index position of the span
- `1` - the *stop* index position (exclusive)
- `label=ORG` - the label assigned to our entity

```
show_ents(doc)
```

```
TATA - ORG - Companies, agencies, institutions, etc.
U.K. - GPE - Countries, cities, states
$6 million - MONEY - Monetary values, including unit
```

Adding Named Entities to All Matching Spans

What if we want to tag *all* occurrences of "TATA"? In this section we show how to use the PhraseMatcher to identify a series of spans in the Doc:

```
doc = nlp(u'Our company plans to introduce a new vacuum cleaner. '
          u'If successful, the vacuum cleaner will be our first product.')

show_ents(doc)
```

```
No named entities found.
```

```
# Import PhraseMatcher and create a matcher object:
from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
```

```
# Create the desired phrase patterns:
phrase_list = ['vacuum cleaner', 'vacuum-cleaner']
phrase_patterns = [nlp(text) for text in phrase_list]
```

```
# Apply the patterns to our matcher object:
matcher.add('newproduct', None, *phrase_patterns)

# Apply the matcher to our Doc object:
matches = matcher(doc)

# See what matches occur:
matches
```

```
[(2689272359382549672, 7, 9), (2689272359382549672, 14, 16)]
```

```
# Here we create Spans from each match, and create named entities from them:
from spacy.tokens import Span

PROD = doc.vocab.strings[u'PRODUCT']

new_ents = [Span(doc, match[1],match[2],label=PROD) for match in matches]

doc.ents = list(doc.ents) + new_ents
```

```
show_ents(doc)
```

```
vacuum cleaner - PRODUCT - Objects, vehicles, foods, etc. (not services)
vacuum cleaner - PRODUCT - Objects, vehicles, foods, etc. (not services)
```

Counting Entities

While spaCy may not have a built-in tool for counting entities, we can pass a conditional statement into a list comprehension:

```
doc = nlp(u'Originally priced at $29.50, the sweater was marked down to five dollars.')

show_ents(doc)
```

```
29.50 - MONEY - Monetary values, including unit
five dollars - MONEY - Monetary values, including unit
```

```
len([ent for ent in doc.ents if ent.label_=='MONEY'])
```

```
2
```

For more on **Named Entity Recognition** visit https://spacy.io/usage/linguistic-features#101

Noun Chunks

`Doc.noun_chunks` are *base noun phrases*: token spans that include the noun and words describing the noun. Noun chunks cannot be nested, cannot overlap, and do not involve prepositional phrases or relative clauses.
Where `Doc.ents` rely on the **ner** pipeline component, `Doc.noun_chunks` are provided by the **parser**.

`noun_chunks` components:

`.text`	The original noun chunk text.
`.root.text`	The original text of the word connecting the noun chunk to the rest of the parse.
`.root.dep_`	Dependency relation connecting the root to its head.
`.root.head.text`	The text of the root token's head.

```
doc = nlp(u"Autonomous cars shift insurance liability toward manufacturers.")

for chunk in doc.noun_chunks:
    print(chunk.text+' - '+chunk.root.text+' - '+chunk.root.dep_+' - '+chunk.root.head.text)
```

```
Autonomous cars - cars - nsubj - shift
insurance liability - liability - dobj - shift
manufacturers - manufacturers - pobj - toward
```

Part-of-Speech Tagging

```
import nltk
from nltk import word_tokenize
```

```
nltk.download('tagsets')
```

```
[nltk_data] Downloading package tagsets to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping help\tagsets.zip.
```

```
True
```

```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

```
True
```

```
nltk.help.upenn_tagset()
```

```
$: dollar
    $ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
'': closing quotation mark
    ' ''
(: opening parenthesis
    ( [ {
): closing parenthesis
    ) ] }
,: comma
    ,
--: dash
    --
.: sentence terminator
    . ! ?
:: colon or ellipsis
    : ; ...
CC: conjunction, coordinating
    & 'n and both but either et for less minus neither nor or plus so
    therefore times v. versus vs. whether yet
CD: numeral, cardinal
    mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-
```

```
text = "I refuse to let this refuse get me down"

tokenized_words = word_tokenize(text)
tagged_words = nltk.pos_tag(tokenized_words)
tagged_words
```

```
[('I', 'PRP'),
 ('refuse', 'VBP'),
 ('to', 'TO'),
 ('let', 'VB'),
 ('this', 'DT'),
 ('refuse', 'NN'),
 ('get', 'VB'),
 ('me', 'PRP'),
 ('down', 'RP')]
```

```
text = """Bear with me, this effort with soon bear fruit,
          otherwise we'll have to run from the bear"""

tokenized_words = word_tokenize(text)
tagged_words = nltk.pos_tag(tokenized_words)
tagged_words
```

```
[('Bear', 'NNP'),
 ('with', 'IN'),
 ('me', 'PRP'),
 (',', ','),
 ('this', 'DT'),
 ('effort', 'NN'),
 ('with', 'IN'),
 ('soon', 'RB'),
 ('bear', 'JJ'),
 ('fruit', 'NN'),
 (',', ','),
 ('otherwise', 'RB'),
 ('we', 'PRP'),
 ("'ll", 'MD'),
 ('have', 'VB'),
 ('to', 'TO'),
 ('run', 'VB'),
 ('from', 'IN'),
 ('the', 'DT'),
 ('bear', 'NN')]
```

```
text = "A bird in hand is worth two in the bush. " +\
       "Good things come to those who wait. " +\
       "There are other fish in the sea. " +\
       "The ball is in your court."

tokenized_words = word_tokenize(text)
tagged_words = nltk.pos_tag(tokenized_words)
tagged_words
```

```
[('A', 'DT'),
 ('bird', 'NN'),
 ('in', 'IN'),
 ('hand', 'NN'),
 ('is', 'VBZ'),
 ('worth', 'JJ'),
 ('two', 'CD'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('bush', 'NN'),
 ('.', '.'),
 ('Good', 'JJ'),
 ('things', 'NNS'),
 ('come', 'VBP'),
 ('to', 'TO'),
 ('those', 'DT'),
 ('who', 'WP'),
 ('wait', 'VBP'),
 ('.', '.'),
 ('There', 'EX'),
 ('are', 'VBP')
```

```
from nltk.probability import FreqDist

fd = FreqDist(tagged_words)
fd_tagged = FreqDist(tag for (word, tag) in tagged_words)
fd_tagged.most_common(10)
```

```
[('NN', 7),
 ('DT', 5),
 ('IN', 4),
 ('.', 4),
 ('JJ', 3),
 ('VBP', 3),
 ('VBZ', 2),
 ('CD', 1),
 ('NNS', 1),
 ('TO', 1)]
```

The Brown Corpus was the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre, such as news, editorial, and so on. 1.1 gives an example of each genre (for a complete list, see http://icame.uib.no/brown/bcm-los.html).

```
nltk.download('brown')
```

```
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

```
True
```

```
nltk.corpus.brown.words()[:10]
```

```
['The',
 'Fulton',
 'County',
 'Grand',
 'Jury',
 'said',
 'Friday',
 'an',
 'investigation',
 'of']
```

```
text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())
```

Lexical categories like "noun" and part-of-speech tags like NN seem to have their uses, but the details will be obscure to many readers. You might wonder what justification there is for introducing this extra level of information. Many of these categories arise from superficial analysis the distribution of words in text. Consider the following analysis involving woman (a noun), bought (a verb), over (a preposition), and the (a determiner). The text.similar() method takes a word w, finds all contexts w1w w2, then finds all words w' that appear in the same context, i.e. w1w'w2.

Similar words here belong to the same part of speech

```
text.similar('boy')
```

```
man time day way girl year house people world city family state room
country car woman program church government job
```

```
text.similar('run')
```

```
get be do in see work go have take make put and find time look day say
use come show
```

```
text.similar('over')
```

```
in on to of and for with from at by that into as up out down through
is all about
```

Extracting Nouns From the data

```
data = "The advent of computer graphic processing units, improvement in mathematical models and availability of big data has
data
```

```
'The advent of computer graphic processing units, improvement in mathematical models and availability of big data has allowed a
rtificial intelligence (AI) using machine learning (ML) and deep learning (DL) techniques to achieve robust performance for bro
ad applications in social-media, the internet of things, the automotive industry and healthcare. DL systems in particular provi
de improved capability in image, speech and motion recognition as well as in natural language processing. In medicine, signific
ant progress of AI and DL systems has been demonstrated in image-centric specialties such as radiology, dermatology, pathology
and ophthalmology. New studies, including pre-registered prospective clinical trials, have shown DL systems are accurate and ef
fective in detecting diabetic retinopathy (DR), glaucoma, age-related macular degeneration (AMD), retinopathy of prematurity, r
efractive error and in identifying cardiovascular risk factors and diseases, from digital fundus photographs. There is also inc
reasing attention on the use of AI and DL systems in identifying disease features, progression and treatment response for retin
al diseases such as neovascular AMD and diabetic macular edema using optical coherence tomography (OCT). Additionally, the appl
ication of ML to visual fields may be useful in detecting glaucoma progression. There are limited studies that incorporate clin
ical data including electronic health records, in AL and DL algorithms, and no prospective studies to demonstrate that AI and D
L algorithms can predict the development of clinical eye disease. This article describes global eye disease burden, unmet needs
and common conditions of public health importance for which AI and DL systems may be applicable. Technical and clinical aspects
to build a DL system to address those needs, and the potential challenges for clinical adoption are discussed. AI, ML and DL wi
ll likely play a crucial role in clinical ophthalmology practice, with implications for screening, diagnosis and follow up of t
he major causes of vision impairment in the setting of ageing populations globally. '
```

```
tokenized_words = word_tokenize(data)
extracting_nouns = [word for (word, pos) in nltk.pos_tag(tokenized_words) if(pos[:2] == 'NN')]
print(extracting_nouns)
```

```
['advent', 'computer', 'units', 'improvement', 'models', 'availability', 'data', 'intelligence', 'AI', 'machine', 'learning',
'ML', 'learning', 'DL', 'techniques', 'performance', 'applications', 'social-media', 'internet', 'things', 'industry', 'healthc
are', 'DL', 'systems', 'provide', 'capability', 'image', 'speech', 'motion', 'recognition', 'language', 'processing', 'medicin
e', 'progress', 'AI', 'DL', 'systems', 'specialties', 'radiology', 'dermatology', 'pathology', 'ophthalmology', 'New', 'studie
s', 'trials', 'DL', 'systems', 'retinopathy', 'DR', 'glaucoma', 'degeneration', 'AMD', 'retinopathy', 'prematurity', 'error',
'risk', 'factors', 'diseases', 'fundus', 'photographs', 'attention', 'use', 'AI', 'DL', 'systems', 'disease', 'features', 'prog
ression', 'treatment', 'response', 'diseases', 'AMD', 'macular', 'edema', 'coherence', 'tomography', 'OCT', 'application', 'M
L', 'fields', 'progression', 'studies', 'data', 'health', 'records', 'AL', 'DL', 'algorithms', 'studies', 'AI', 'DL', 'developm
ent', 'eye', 'disease', 'article', 'eye', 'disease', 'burden', 'needs', 'conditions', 'health', 'importance', 'AI', 'DL', 'syst
ems', 'aspects', 'DL', 'system', 'needs', 'challenges', 'adoption', 'AI', 'ML', 'DL', 'role', 'ophthalmology', 'practice', 'imp
lications', 'screening', 'diagnosis', 'causes', 'vision', 'impairment', 'setting', 'populations']
```

Extracting Verbs From the data

```
tokenized_words = word_tokenize(data)
extracting_verbs = [word for (word, pos) in nltk.pos_tag(tokenized_words) if(pos[:2] == 'VB')]
print(extracting_verbs)
```

```
['processing', 'has', 'allowed', 'using', 'achieve', 'improved', 'has', 'been', 'demonstrated', 'including', 'have', 'shown',
'are', 'detecting', 'identifying', 'is', 'increasing', 'identifying', 'using', 'be', 'detecting', 'are', 'incorporate', 'includ
ing', 'demonstrate', 'algorithms', 'predict', 'describes', 'be', 'build', 'address', 'are', 'discussed', 'play', 'follow', 'age
ing']
```

EXPERIMENT NO. 21

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 14-04-2021
Faculty Signature:
Marks:

Objective(s):

To create Bag-of-Words(BOW) and Bag-og-n-grams using the following:

(a) Bag-of-words Using the Count Vectorizer

(b) Bag-of-n-grams Using the Count Vectorizer

(c) Bag-of-words Using the Tf-Idf Vectorizer

Outcome: Learning to create bag of words.

Problem Statement:
To build bag of words using different vectorizers.

Background Study:
The bag-of-words model is a simplifying representation used in natural language processing and information retrieval. In this model, a text is represented as the bag of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model has also been used for computer vision.

Student Work Area

Code/Sample Outputs:

Vectorize text as a bag-of-words

```
import sklearn

print(sklearn.__version__)
```

```
0.20.3
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
train_text = ["A bird in hand is worth two in the bush.",
              "Good things come to those who wait.",
              "These watches cost $1500! ",
              "There are other fish in the sea.",
              "The ball is in your court.",
              "Mr. Smith Goes to Washington ",
              "Doogie Howser M.D."]

count_vectorizer = CountVectorizer()
```

```
count_vectorizer.fit(train_text)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

```
count_vectorizer.get_feature_names()
```

```
['1500',
 'are',
 'ball',
 'bird',
 'bush',
 'come',
 'cost',
 'court',
 'doogie',
 'fish',
 'goes',
 'good',
 'hand',
 'howser',
 'in',
 'is',
 'mr',
 'other',
 'sea',
 'smith',
 'the'
```

```
count_vectorizer.get_stop_words()
```

```
count_vectorizer.vocabulary_
```

```
{'bird': 3,
 'in': 14,
 'hand': 12,
 'is': 15,
 'worth': 31,
 'two': 26,
 'the': 20,
 'bush': 4,
 'good': 11,
 'things': 23,
 'come': 5,
 'to': 25,
 'those': 24,
 'who': 30,
 'wait': 27,
 'these': 22,
 'watches': 29,
 'cost': 6,
 '1500': 0,
 'there': 21,
 'are': 1
```

```
count_vectorizer.vocabulary_.get('things')
```

```
23
```

```
train_text
```

```
['A bird in hand is worth two in the bush.',
 'Good things come to those who wait.',
 'These watches cost $1500! ',
 'There are other fish in the sea.',
 'The ball is in your court.',
 'Mr. Smith Goes to Washington ',
 'Doogie Howser M.D.']
```

```
transformed_vector = count_vectorizer.transform(train_text)
```

```
print(transformed_vector.shape)
```

```
(7, 33)
```

```
print(transformed_vector.toarray())
```

```
[[0 0 0 1 1 0 0 0 0 0 0 0 1 0 2 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0]
 [0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0]
 [1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

```
test_text = ["Every cloud has a silver lining."]

count_vectorizer.transform(test_text).toarray()
```

```
array([[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]], dtype=int64)
```

```
count_vectorizer.fit(train_text + test_text)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

```
print(count_vectorizer.vocabulary_)
```

```
{'bird': 3, 'in': 17, 'hand': 14, 'is': 18, 'worth': 36, 'two': 31, 'the': 25, 'bush': 4, 'good': 13, 'things': 28, 'come': 6,
'to': 30, 'those': 29, 'who': 35, 'wait': 32, 'these': 27, 'watches': 34, 'cost': 7, '1500': 0, 'there': 26, 'are': 1, 'other':
21, 'fish': 11, 'sea': 22, 'ball': 2, 'your': 37, 'court': 8, 'mr': 20, 'smith': 24, 'goes': 12, 'washington': 33, 'doogie': 9,
'howser': 16, 'every': 10, 'cloud': 5, 'has': 15, 'silver': 23, 'lining': 19}
```

```
count_vectorizer.transform(test_text).toarray()
```

```
array([[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
```

```
text = ["That bird is sitting in the bush and this bird is in hand.",
        "Wait and then walk",
        "Watches are cool "]

transformed_vector = count_vectorizer.transform(text)

transformed_vector
```

```
<3x38 sparse matrix of type '<class 'numpy.int64'>'
        with 9 stored elements in Compressed Sparse Row format>
```

```
print(transformed_vector)
```

```
  (0, 3)        2
  (0, 4)        1
  (0, 14)       1
  (0, 17)       2
  (0, 18)       2
  (0, 25)       1
  (1, 32)       1
  (2, 1)        1
  (2, 34)       1
```

```
transformed_vector.shape
```

```
(3, 38)
```

```
print(transformed_vector.toarray())
```

```
[[0 0 0 2 1 0 0 0 0 0 0 0 0 1 0 0 2 2 0 0 0 0 0 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 1 0 0
  0 0]
 [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 1 0
  0 0]]
```

```
print(count_vectorizer.vocabulary_)
```

```
{'marie': 91, 'curie': 34, 'was': 153, 'polish': 111, 'born': 21, 'physicist': 108, 'and': 11, 'chemist': 27, 'one': 103, 'of':
101, 'the': 144, 'most': 96, 'famous': 56, 'scientists': 128, 'her': 72, 'time': 146, 'together': 148, 'with': 161, 'husband':
76, 'pierre': 110, 'she': 131, 'awarded': 15, 'nobel': 99, 'prize': 114, 'in': 77, '1903': 4, 'went': 154, 'on': 102, 'to': 14
7, 'win': 159, 'another': 13, '1911': 6, 'sklodowska': 134, 'warsaw': 152, 'november': 100, '1867': 0, 'daughter': 37, 'teache
r': 140, '1891': 1, 'paris': 107, 'study': 136, 'physics': 109, 'mathematics': 93, 'at': 14, 'sorbonne': 135, 'where': 157, 'me
t': 95, 'professor': 115, 'school': 126, 'they': 145, 'were': 155, 'married': 92, '1895': 2, 'curies': 35, 'worked': 164, 'inve
stigating': 79, 'radioactivity': 117, 'building': 22, 'work': 163, 'german': 64, 'roentgen': 125, 'french': 61, 'becquerel': 1
7, 'july': 82, '1898': 3, 'announced': 12, 'discovery': 43, 'new': 98, 'chemical': 26, 'element': 49, 'polonium': 112, 'end': 5
0, 'year': 166, 'radium': 119, 'along': 9, 'for': 59, 'life': 87, 'cut': 36, 'short': 132, '1906': 5, 'when': 156, 'he': 67, 'k
nocked': 84, 'down': 45, 'killed': 83, 'by': 23, 'carriage': 24, 'took': 149, 'over': 106, 'his': 75, 'teaching': 141, 'post':
113, 'becoming': 16, 'first': 58, 'woman': 162, 'teach': 139, 'devoted': 41, 'herself': 73, 'continuing': 30, 'that': 143, 'ha
d': 66, 'begun': 19, 'received': 122, 'second': 129, 'chemistry': 28, 'research': 124, 'crucial': 33, 'development': 40, 'ray
s': 121, 'surgery': 138, 'during': 47, 'world': 165, 'war': 151, 'helped': 71, 'equip': 52, 'ambulances': 10, 'ray': 120, 'equi
pment': 53, 'which': 158, 'drove': 46, 'front': 63, 'lines': 88, 'international': 78, 'red': 123, 'cross': 32, 'made': 89, 'hea
d': 68, 'its': 81, 'radiological': 118, 'service': 130, 'held': 70, 'training': 150, 'courses': 31, 'medical': 94, 'orderlies':
105, 'doctors': 44, 'techniques': 142, 'despite': 38, 'success': 137, 'continued': 29, 'face': 55, 'great': 65, 'opposition': 1
04, 'from': 62, 'male': 90, 'france': 60, 'never': 97, 'significant': 133, 'financial': 57, 'benefits': 20, 'late': 85, '1920
s': 7, 'health': 69, 'beginning': 18, 'deteriorate': 39, 'died': 42, '1934': 8, 'leukaemia': 86, 'caused': 25, 'exposure': 54,
'high': 74, 'energy': 51, 'radiation': 116, 'eldest': 48, 'irene': 80, 'scientist': 127, 'winner': 160}
```

We lost:

- The meaning of text corpus
- The ordering of the words

```
count_vectorizer.inverse_transform(transformed_vector)
```

```
[array(['marie', 'curie', 'was', 'polish', 'born', 'physicist', 'and',
        'chemist', 'one', 'of', 'the', 'most', 'famous', 'scientists',
        'her', 'time'], dtype='<U13'),
 array(['was', 'and', 'the', 'her', 'together', 'with', 'husband',
        'pierre', 'she', 'awarded', 'nobel', 'prize', 'in', '1903', 'went',
        'on', 'to', 'win', 'another', '1911'], dtype='<U13'),
 array(['marie', 'was', 'born', 'of', 'the', 'in', 'on', 'sklodowska',
        'warsaw', 'november', '1867', 'daughter', 'teacher'], dtype='<U13'),
 array(['curie', 'and', 'of', 'the', 'pierre', 'she', 'in', 'went', 'to',
        '1891', 'paris', 'study', 'physics', 'mathematics', 'at',
        'sorbonne', 'where', 'met', 'professor', 'school'], dtype='<U13'),
 array(['in', 'they', 'were', 'married', '1895'], dtype='<U13'),
 array(['physicist', 'and', 'of', 'the', 'together', 'on', 'curies',
        'worked', 'investigating', 'radioactivity', 'building', 'work',
        'german', 'roentgen', 'french', 'becquerel'], dtype='<U13'),
 array(['of', 'the', 'in', 'curies', 'july', '1898', 'announced',
        'discovery', 'new', 'chemical', 'element', 'polonium'],
       dtype='<U13'),
 array(['of', 'the', 'another', 'at', 'they', 'announced', 'discovery',
        'end', 'year', 'radium'], dtype='<U13'),
 array(['the', 'with', 'awarded', 'nobel', 'prize', 'in', '1903',
```

Vectorize text as a bag-of-n-grams

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
train_text = ["A bird in hand is worth two in the bush.",
              "Good things come to those who wait.",
              "These watches cost $1500! ",
              "There are other fish in the sea.",
              "The ball is in your court.",
              "Mr. Smith Goes to Washington ",
              "Doogie Howser M.D."]
```

```
# An ngram_range of (1, 1) means only unigrams, (1, 2) means unigrams and bigrams, and (2, 2) means only bigrams.

n_gram_vectorizer = CountVectorizer(ngram_range=(2, 2))
```

```
transformed_vector = n_gram_vectorizer.fit_transform(train_text)
```

```
n_gram_vectorizer.vocabulary_
# Second numbers are Ids
```

```
{'bird in': 2,
 'in hand': 10,
 'hand is': 9,
 'is worth': 14,
 'worth two': 30,
 'two in': 27,
 'in the': 11,
 'the bush': 19,
 'good things': 8,
 'things come': 23,
 'come to': 3,
 'to those': 25,
 'those who': 24,
 'who wait': 29,
 'these watches': 22,
 'watches cost': 28,
 'cost 1500': 4,
 'there are': 21,
 'are other': 0,
 'other fish': 16,
 'fish in': 6
```

```
transformed_vector.toarray()
```

```
array([[0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 1, 1, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 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]])
```

```
n_gram_vectorizer = CountVectorizer(ngram_range=(1, 2))

transformed_vector = n_gram_vectorizer.fit_transform(train_text)

n_gram_vectorizer.vocabulary_
```

```
{'bird': 5,
 'in': 24,
 'hand': 21,
 'is': 28,
 'worth': 61,
 'two': 53,
 'the': 38,
 'bush': 7,
 'bird in': 6,
 'in hand': 25,
 'hand is': 22,
 'is worth': 30,
 'worth two': 62,
 'two in': 54,
 'in the': 26,
 'the bush': 40,
 'good': 19,
 'things': 46,
 'come': 8,
 'to': 50,
 'those': 48
```

```
transformed_vector.toarray()
```

```
array([[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        1, 0, 2, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
        1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 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, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        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]])
```

```
with open('biography.txt', 'r') as f:
    file_contents = f.read()

print(file_contents)
```

```
Marie Curie was a Polish-born physicist and chemist and one of the most famous scientists of her time.
Together with her husband Pierre, she was awarded the Nobel Prize in 1903, and she went on to win another in 1911.
Marie Sklodowska was born in Warsaw on 7 November 1867, the daughter of a teacher.
In 1891, she went to Paris to study physics and mathematics at the Sorbonne where she met Pierre Curie, professor of the Sch
ool of Physics.
They were married in 1895.
The Curies worked together investigating radioactivity, building on the work of the German physicist Roentgen and the French
physicist Becquerel.
In July 1898, the Curies announced the discovery of a new chemical element, polonium.
At the end of the year, they announced the discovery of another, radium.
The Curies, along with Becquerel, were awarded the Nobel Prize for Physics in 1903.
Pierre's life was cut short in 1906 when he was knocked down and killed by a carriage.
Marie took over his teaching post, becoming the first woman to teach at the Sorbonne, and devoted herself to continuing the
work that they had begun together.
She received a second Nobel Prize, for Chemistry, in 1911.
The Curie's research was crucial in the development of x-rays in surgery.
During World War One Curie helped to equip ambulances with x-ray equipment, which she herself drove to the front lines.
The International Red Cross made her head of its radiological service and she held training courses for medical orderlies an
d doctors in the new techniques.
Despite her success, Marie continued to face great opposition from male scientists in France, and she never received signifi
cant financial benefits from her work.
```

```
sentences = file_contents.split('\n')

print(sentences)
```

```
['Marie Curie was a Polish-born physicist and chemist and one of the most famous scientists of her time.', 'Together with her h
usband Pierre, she was awarded the Nobel Prize in 1903, and she went on to win another in 1911.', 'Marie Sklodowska was born in
Warsaw on 7 November 1867, the daughter of a teacher.', 'In 1891, she went to Paris to study physics and mathematics at the Sor
bonne where she met Pierre Curie, professor of the School of Physics.', 'They were married in 1895.', 'The Curies worked togeth
er investigating radioactivity, building on the work of the German physicist Roentgen and the French physicist Becquerel.', 'In
July 1898, the Curies announced the discovery of a new chemical element, polonium.', 'At the end of the year, they announced th
e discovery of another, radium.', 'The Curies, along with Becquerel, were awarded the Nobel Prize for Physics in 1903.', "Pierr
e's life was cut short in 1906 when he was knocked down and killed by a carriage.", 'Marie took over his teaching post, becomin
g the first woman to teach at the Sorbonne, and devoted herself to continuing the work that they had begun together.', 'She rec
eived a second Nobel Prize, for Chemistry, in 1911.', "The Curie's research was crucial in the development of x-rays in surger
y.", 'During World War One Curie helped to equip ambulances with x-ray equipment, which she herself drove to the front lines.',
'The International Red Cross made her head of its radiological service and she held training courses for medical orderlies and
doctors in the new techniques.', 'Despite her success, Marie continued to face great opposition from male scientists in France,
and she never received significant financial benefits from her work.', 'By the late 1920s her health was beginning to deteriora
te.', 'She died on 4 July 1934 from leukaemia, caused by exposure to high-energy radiation from her research.', "The Curies' el
dest daughter Irene was herself a scientist and winner of the Nobel Prize for Chemistry.", '']
```

```
n_gram_vectorizer = CountVectorizer(ngram_range=(2, 3))
```

```
transformed_vector = n_gram_vectorizer.fit_transform(sentences)
```

```
vocabulary = n_gram_vectorizer.vocabulary_

vocabulary
```

```
{'marie curie': 251,
 'curie was': 92,
 'was polish': 504,
 'polish born': 330,
 'born physicist': 59,
 'physicist and': 315,
 'and chemist': 18,
 'chemist and': 72,
 'and one': 28,
 'one of': 305,
 'of the': 289,
 'the most': 437,
 'most famous': 265,
 'famous scientists': 142,
 'scientists of': 367,
 'of her': 279,
 'her time': 191,
 'marie curie was': 252,
 'curie was polish': 93,
 'was polish born': 505,
 'polish born physicist': 331
```

```
# The converse mapping from feature name to column index is stored in the vocabulary_ attribute of the vectorizer:
n_gram_vectorizer.vocabulary_.get('marie curie')
```

```
251
```

```
n_gram_vectorizer.vocabulary_.get('of her')
```

```
279
```

```
vocabulary.items()
```

```
dict_items([('marie curie', 251), ('curie was', 92), ('was polish', 504), ('polish born', 330), ('born physicist', 59), ('ph
ysicist and', 315), ('and chemist', 18), ('chemist and', 72), ('and one', 28), ('one of', 305), ('of the', 289), ('the mos
t', 437), ('most famous', 265), ('famous scientists', 142), ('scientists of', 367), ('of her', 279), ('her time', 191), ('ma
rie curie was', 252), ('curie was polish', 93), ('was polish born', 505), ('polish born physicist', 331), ('born physicist a
nd', 60), ('physicist and chemist', 316), ('and chemist and', 19), ('chemist and one', 73), ('and one of', 29), ('one of th
e', 306), ('of the most', 291), ('the most famous', 438), ('most famous scientists', 266), ('famous scientists of', 143),
('scientists of her', 368), ('of her time', 280), ('together with', 480), ('with her', 526), ('her husband', 186), ('husband
pierre', 203), ('pierre she', 328), ('she was', 385), ('was awarded', 490), ('awarded the', 46), ('the nobel', 441), ('nobel
prize', 272), ('prize in', 337), ('in 1903', 208), ('1903 and', 6), ('and she', 30), ('she went', 387), ('went on', 506),
('on to', 301), ('to win', 476), ('win another', 520), ('another in', 40), ('in 1911', 212), ('together with her', 481), ('w
ith her husband', 527), ('her husband pierre', 187), ('husband pierre she', 204), ('pierre she was', 329), ('she was awarde
d', 386), ('was awarded the', 491), ('awarded the nobel', 47), ('the nobel prize', 442), ('nobel prize in', 274), ('prize in
1903', 338), ('in 1903 and', 209), ('1903 and she', 7), ('and she went', 33), ('she went on', 388), ('went on to', 507), ('o
n to win', 302), ('to win another', 477), ('win another in', 521), ('another in 1911', 41), ('marie sklodowska', 253), ('skl
odowska was', 394), ('was born', 494), ('born in', 57), ('in warsaw', 221), ('warsaw on', 488), ('on november', 297), ('nove
mber 1867', 275), ('1867 the', 0), ('the daughter', 417), ('daughter of', 106), ('of teacher', 288), ('marie sklodowska wa
s', 254), ('sklodowska was born', 395), ('was born in', 495), ('born in warsaw', 58), ('in warsaw on', 222), ('warsaw on nov
ember', 489), ('on november 1867', 298), ('november 1867 the', 276), ('1867 the daughter', 1), ('the daughter of', 418), ('d
aughter of teacher', 107), ('in 1891', 205), ('1891 she', 2), ('went to', 508), ('to paris', 468), ('paris to', 313), ('to s
tudy', 470), ('study physics', 400), ('physics and', 320), ('and mathematics', 26), ('mathematics at', 259), ('at the', 43),
```

```
sorted_word_list = sorted([(word_count[i], n_gram) for n_gram, i in vocabulary.items()], reverse=True)

sorted_word_list
```

```
[(5, 'of the'),
 (4, 'the curies'),
 (4, 'nobel prize'),
 (3, 'the nobel prize'),
 (3, 'the nobel'),
 (3, 'prize for'),
 (3, 'nobel prize for'),
 (3, 'at the'),
 (3, 'and she'),
 (2, 'the work'),
 (2, 'the sorbonne'),
 (2, 'the discovery of'),
 (2, 'the discovery'),
 (2, 'she went'),
 (2, 'prize for chemistry'),
 (2, 'in the'),
 (2, 'in 1911'),
 (2, 'in 1903'),
 (2, 'from her'),
 (2, 'for chemistry'),
```

Using nltk to find ngrams

```
train_text
```

```
['A bird in hand is worth two in the bush.',
 'Good things come to those who wait.',
 'These watches cost $1500! ',
 'There are other fish in the sea.',
 'The ball is in your court.',
 'Mr. Smith Goes to Washington ',
 'Doogie Howser M.D.']
```

```
from nltk import bigrams
from nltk import trigrams

from nltk.tokenize import word_tokenize
```

```
word_tokens = word_tokenize(" ".join(train_text))

word_tokens
```

```
['A',
 'bird',
 'in',
 'hand',
 'is',
 'worth',
 'two',
 'in',
 'the',
 'bush',
 '.',
 'Good',
 'things',
 'come',
 'to',
 'those',
 'who',
 'wait',
 '.',
 'These',
 'watches'
```

```
nltk_bigrams = bigrams(word_tokens)

list(nltk_bigrams)
```

```
[('A', 'bird'),
 ('bird', 'in'),
 ('in', 'hand'),
 ('hand', 'is'),
 ('is', 'worth'),
 ('worth', 'two'),
 ('two', 'in'),
 ('in', 'the'),
 ('the', 'bush'),
 ('bush', '.'),
 ('.', 'Good'),
 ('Good', 'things'),
 ('things', 'come'),
 ('come', 'to'),
 ('to', 'those'),
 ('those', 'who'),
 ('who', 'wait'),
 ('wait', '.'),
 ('.', 'These'),
 ('These', 'watches'),
 ('watches', 'cost'),
```

```
nltk_trigrams = trigrams(word_tokens)

list(nltk_trigrams)
```

```
[('A', 'bird', 'in'),
 ('bird', 'in', 'hand'),
 ('in', 'hand', 'is'),
 ('hand', 'is', 'worth'),
 ('is', 'worth', 'two'),
 ('worth', 'two', 'in'),
 ('two', 'in', 'the'),
 ('in', 'the', 'bush'),
 ('the', 'bush', '.'),
 ('bush', '.', 'Good'),
 ('.', 'Good', 'things'),
 ('Good', 'things', 'come'),
 ('things', 'come', 'to'),
 ('come', 'to', 'those'),
 ('to', 'those', 'who'),
 ('those', 'who', 'wait'),
 ('who', 'wait', '.'),
 ('wait', '.', 'These'),
 ('.', 'These', 'watches'),
 ('These', 'watches', 'cost'),
 ('watches', 'cost', '$'),
```

```
from nltk import ngrams

fivegrams = ngrams(word_tokens, 5)

for grams in fivegrams:
    print(grams)
```

```
('A', 'bird', 'in', 'hand', 'is')
('bird', 'in', 'hand', 'is', 'worth')
('in', 'hand', 'is', 'worth', 'two')
('hand', 'is', 'worth', 'two', 'in')
('is', 'worth', 'two', 'in', 'the')
('worth', 'two', 'in', 'the', 'bush')
('two', 'in', 'the', 'bush', '.')
('in', 'the', 'bush', '.', 'Good')
('the', 'bush', '.', 'Good', 'things')
('bush', '.', 'Good', 'things', 'come')
('.', 'Good', 'things', 'come', 'to')
('Good', 'things', 'come', 'to', 'those')
('things', 'come', 'to', 'those', 'who')
('come', 'to', 'those', 'who', 'wait')
('to', 'those', 'who', 'wait', '.')
('those', 'who', 'wait', '.', 'These')
('who', 'wait', '.', 'These', 'watches')
('wait', '.', 'These', 'watches', 'cost')
('.', 'These', 'watches', 'cost', '$')
('These', 'watches', 'cost', '$', '1500')
('watches', 'cost', '$', '1500', ')')
```

- https://tedboy.github.io/nlps/generated/generated/nltk.BigramAssocMeasures
- http://www.nltk.org/_modules/nltk/collocations.html#BigramCollocationFinder

```
from nltk.collocations import BigramAssocMeasures
from nltk.collocations import BigramCollocationFinder
```

```
word_tokens = word_tokenize(file_contents)
```

```
bigram_measures = BigramAssocMeasures()

finder = BigramCollocationFinder.from_words(word_tokens)

finder.apply_freq_filter(3)

matches = finder.nbest(bigram_measures.raw_freq, 15)

matches
```

```
[('.', 'The'),
 ('of', 'the'),
 ('Nobel', 'Prize'),
 (',', 'and'),
 ('The', 'Curies'),
 ('and', 'she'),
 ('the', 'Nobel')]
```

```
bigram_measures = BigramAssocMeasures()

finder = BigramCollocationFinder.from_words(word_tokens)

finder.apply_freq_filter(2)

matches = finder.nbest(bigram_measures.raw_freq, 15)

matches
```

```
[('.', 'The'),
 ('of', 'the'),
 ('Nobel', 'Prize'),
 (',', 'and'),
 ('The', 'Curies'),
 ('and', 'she'),
 ('the', 'Nobel'),
 (',', 'she'),
 (',', 'the'),
 ('.', 'In'),
 ('.', 'Marie'),
 ('.', 'She'),
 ('1911', '.'),
 ('Prize', 'for'),
 ('announced', 'the')]
```

Creating a count vector

```
from sklearn.feature_extraction.text import CountVectorizer

train_text = ["A bird in hand is worth two in the bush.",
              "Good things come to those who wait.",
              "These watches cost $1500! ",
              "There are other fish in the sea.",
              "The ball is in your court.",
              "Mr. Smith Goes to Washington ",
              "Doogie Howser M.D."]
```

```
count_vectorizer = CountVectorizer()

frequency_term_matrix = count_vectorizer.fit_transform(train_text)
```

```
len(count_vectorizer.vocabulary_)
```

```
33
```

```
frequency_term_matrix.shape
```

```
(7, 33)
```

```
frequency_term_matrix.toarray()
```

```
array([[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
        0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

Building the tf-idf matrix

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()
```

```
tfidf_vector1 = tfidf_transformer.fit_transform(frequency_term_matrix)

tfidf_vector1.shape
```

```
(7, 33)
```

```
tfidf_vector1.toarray()
```

```
       0.        , 0.        , 0.        , 0.        , 0.40801493,
       0.        , 0.        , 0.        , 0.        , 0.28949879,
       0.        , 0.        , 0.40801493, 0.40801493, 0.        ,
       0.28949879, 0.40801493, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ],
      [0.        , 0.        , 0.46146654, 0.        , 0.        ,
       0.        , 0.        , 0.46146654, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.3274243 ,
       0.38305686, 0.        , 0.        , 0.        , 0.        ,
       0.3274243 , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.46146654],
      [0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.46180424, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.46180424, 0.        , 0.        , 0.46180424,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.38333718, 0.        , 0.        , 0.46180424, 0.        ,
       0.        , 0.        , 0.        ],
```

```
print(tfidf_vector1.toarray())
```

```
[[0.         0.         0.         0.34908308 0.34908308 0.
  0.         0.         0.         0.         0.         0.
  0.34908308 0.         0.49536976 0.28976893 0.         0.
  0.         0.         0.24768488 0.         0.         0.
  0.         0.         0.34908308 0.         0.         0.
  0.         0.34908308 0.        ]
 [0.         0.         0.         0.         0.         0.38665001
  0.         0.         0.         0.         0.         0.38665001
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.38665001
  0.38665001 0.32095271 0.         0.38665001 0.         0.
  0.38665001 0.         0.         0.        ]
 [0.5        0.         0.         0.         0.         0.
  0.5        0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.5        0.
  0.         0.         0.         0.         0.         0.5
  0.         0.         0.         0.        ]
 [0.         0.40801493 0.         0.         0.         0.
```

TfidfVectorizer = CountVectorizer + TfidfTransformer

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()
```

```
tfidf_vector2 = tfidf_vectorizer.fit_transform(train_text)

tfidf_vectorizer.vocabulary_
```

```
{'bird': 3,
 'in': 14,
 'hand': 12,
 'is': 15,
 'worth': 31,
 'two': 26,
 'the': 20,
 'bush': 4,
 'good': 11,
 'things': 23,
 'come': 5,
 'to': 25,
 'those': 24,
 'who': 30,
 'wait': 27,
 'these': 22,
 'watches': 29,
 'cost': 6,
 '1500': 0,
 'there': 21
```

```
tfidf_vector2.shape
```

```
(7, 33)
```

```
tfidf_vectorizer.idf_
```

```
array([2.38629436, 2.38629436, 2.38629436, 2.38629436, 2.38629436,
       2.38629436, 2.38629436, 2.38629436, 2.38629436, 2.38629436,
       2.38629436, 2.38629436, 2.38629436, 2.38629436, 1.69314718,
       1.98082925, 2.38629436, 2.38629436, 2.38629436, 2.38629436,
       1.69314718, 2.38629436, 2.38629436, 2.38629436, 2.38629436,
       1.98082925, 2.38629436, 2.38629436, 2.38629436, 2.38629436,
       2.38629436, 2.38629436, 2.38629436])
```

```
dict(zip(tfidf_vectorizer.get_feature_names(), tfidf_vectorizer.idf_))
```

```
{'1500': 2.386294361119891,
 'are': 2.386294361119891,
 'ball': 2.386294361119891,
 'bird': 2.386294361119891,
 'bush': 2.386294361119891,
 'come': 2.386294361119891,
 'cost': 2.386294361119891,
 'court': 2.386294361119891,
 'doogie': 2.386294361119891,
 'fish': 2.386294361119891,
 'goes': 2.386294361119891,
 'good': 2.386294361119891,
 'hand': 2.386294361119891,
 'howser': 2.386294361119891,
 'in': 1.6931471805599454,
 'is': 1.9808292530117262,
 'mr': 2.386294361119891,
 'other': 2.386294361119891,
 'sea': 2.386294361119891,
 'smith': 2.386294361119891
```

Final scorings of each word from the other words in the vocabulary.

- The scores are normalized to values between 0 and 1

```
1 tfidf_vector2.toarray()
```

```
array([[0.        , 0.        , 0.        , 0.34908308, 0.34908308,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.34908308, 0.        , 0.49536976,
        0.28976893, 0.        , 0.        , 0.        , 0.        ,
        0.24768488, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.34908308, 0.        , 0.        , 0.        ,
        0.        , 0.34908308, 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.38665001, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.38665001, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.38665001, 0.38665001,
        0.32095271, 0.        , 0.38665001, 0.        , 0.        ,
        0.38665001, 0.        , 0.        ],
       [0.5       , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.5       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.5       , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.5       ,
```

```
1 print(tfidf_vector2)
```

```
  (0, 4)        0.3490830767264469
  (0, 20)       0.24768487776302442
  (0, 26)       0.3490830767264469
  (0, 31)       0.3490830767264469
  (0, 15)       0.2897689326921819
  (0, 12)       0.3490830767264469
  (0, 14)       0.49536975552604884
  (0, 3)        0.3490830767264469
  (1, 27)       0.386650005027498
  (1, 30)       0.386650005027498
  (1, 24)       0.386650005027498
  (1, 25)       0.32095270940344806
  (1, 5)        0.386650005027498
  (1, 23)       0.386650005027498
  (1, 11)       0.386650005027498
  (2, 0)        0.5
  (2, 6)        0.5
  (2, 29)       0.5
  (2, 22)       0.5
  (3, 18)       0.40801492725049465
```

```
1 print(tfidf_vector1)
```

```
  (0, 31)       0.34908307672644684
  (0, 26)       0.34908307672644684
  (0, 20)       0.2476848777630244
  (0, 15)       0.2897689326921819
  (0, 14)       0.4953697555260488
  (0, 12)       0.34908307672644684
  (0, 4)        0.34908307672644684
  (0, 3)        0.34908307672644684
  (1, 30)       0.386650005027498
  (1, 27)       0.386650005027498
  (1, 25)       0.32095270940344806
  (1, 24)       0.386650005027498
  (1, 23)       0.386650005027498
  (1, 11)       0.386650005027498
  (1, 5)        0.386650005027498
  (2, 29)       0.5
  (2, 22)       0.5
  (2, 6)        0.5
  (2, 0)        0.5
  (3, 21)       0.40801492725049476
```

EXPERIMENT NO. 22

Student Name and Roll Number: Chayan Gulati 18csu054
Semester /Section: 6/DS-VI-A2
Link to Code: https://github.com/chayangulati321/Deep-Learning
Date: 14-04-2021
Faculty Signature:
Marks:

Objective(s): To build a NLP model for spam detection using TFIDF vectorizer
Outcome: Learning how to make spam detector.
Problem Statement: To build a Model to detect spam messages.
Background Study: Using natural language processing to build a spam filter for text messages. We will make use of some NLP concepts and combine them with machine learning to build a spam filter for SMS text messages.

Student Work Area

Code/Sample Outputs:

Building a Natural Language Processor From Scratch

In this section we'll use basic Python to build a rudimentary NLP system. We'll build a *corpus of documents* (two small text files), create a *vocabulary* from all the words in both documents, and then demonstrate a *Bag of Words* technique to extract features from each document.

> **This first section is for illustration only!**
> Don't bother memorizing the code - we'd never do this in real life.

Start with some documents:

For simplicity we won't use any punctuation.

```
%%writefile 1.txt
This is a story about cats
our feline pets
Cats are furry animals
```

```
Overwriting 1.txt
```

```
%%writefile 2.txt
This story is about surfing
Catching waves is fun
Surfing is a popular water sport
```

```
Overwriting 2.txt
```

Build a vocabulary

The goal here is to build a numerical array from all the words that appear in every document. Later we'll create instances (vectors) for each individual document.

```
vocab = {}
i = 1

with open('1.txt') as f:
    x = f.read().lower().split()

for word in x:
    if word in vocab:
        continue
    else:
        vocab[word]=i
        i+=1

print(vocab)
```

```
{'this': 1, 'is': 2, 'a': 3, 'story': 4, 'about': 5, 'cats': 6, 'our': 7, 'feline': 8, 'pets': 9, 'are': 10, 'furry': 11, 'anim
als': 12}
```

```
with open('2.txt') as f:
    x = f.read().lower().split()

for word in x:
    if word in vocab:
        continue
    else:
        vocab[word]=i
        i+=1

print(vocab)
```

```
{'this': 1, 'is': 2, 'a': 3, 'story': 4, 'about': 5, 'cats': 6, 'our': 7, 'feline': 8, 'pets': 9, 'are': 10, 'furry': 11, 'anim
als': 12, 'surfing': 13, 'catching': 14, 'waves': 15, 'fun': 16, 'popular': 17, 'water': 18, 'sport': 19}
```

Even though `2.txt` has 15 words, only 7 new words were added to the dictionary.

Feature Extraction

Now that we've encapsulated our "entire language" in a dictionary, let's perform *feature extraction* on each of our original documents:

```
# Create an empty vector with space for each word in the vocabulary:
one = ['1.txt']+[0]*len(vocab)
one
```

```
['1.txt', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
# map the frequencies of each word in 1.txt to our vector:
with open('1.txt') as f:
    x = f.read().lower().split()

for word in x:
    one[vocab[word]]+=1

one
```

```
['1.txt', 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

We can see that most of the words in 1.txt appear only once, although "cats" appears twice.

```
# Do the same for the second document:
two = ['2.txt']+[0]*len(vocab)

with open('2.txt') as f:
    x = f.read().lower().split()

for word in x:
    two[vocab[word]]+=1
```

```
# Compare the two vectors:
print(f'{one}\n{two}')
```

```
['1.txt', 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
['2.txt', 1, 3, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1]
```

By comparing the vectors we see that some words are common to both, some appear only in `1.txt`, others only in `2.txt`. Extending this logic to tens of thousands of documents, we would see the vocabulary dictionary grow to hundreds of thousands of words. Vectors would contain mostly zero values, making them *sparse matrices*.

Feature Extraction from Text

In the **Scikit-learn Primer** lecture we applied a simple SVC classification model to the SMSSpamCollection dataset. We tried to predict the ham/spam label based on message length and punctuation counts. In this section we'll actually look at the text of each message and try to perform a classification based on content. We'll take advantage of some of scikit-learn's feature extraction tools.

Load a dataset

```
# Perform imports and load the dataset:
import numpy as np
import pandas as pd

df = pd.read_csv('smsspamcollection.tsv', sep='\t')
df.head()
```

	label	message	length	punct
0	ham	Go until jurong point, crazy.. Available only ...	111	9
1	ham	Ok lar... Joking wif u oni...	29	6
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	6
3	ham	U dun say so early hor... U c already then say...	49	6
4	ham	Nah I don't think he goes to usf, he lives aro...	61	2

Check for missing values:

Always a good practice.

```
df.isnull().sum()
```

```
label      0
message    0
length     0
punct      0
dtype: int64
```

Take a quick look at the *ham* and *spam* `label` column:

```
df['label'].value_counts()
```

```
ham     4825
spam     747
Name: label, dtype: int64
```

4825 out of 5572 messages, or 86.6%, are ham. This means that any text classification model we create has to perform **better than 86.6%** to beat random chance.

Split the data into train & test sets:

```
from sklearn.model_selection import train_test_split

X = df['message']  # this time we want to look at the text
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Scikit-learn's CountVectorizer

Text preprocessing, tokenizing and the ability to filter out stopwords are all included in CountVectorizer, which builds a dictionary of features and transforms documents to feature vectors.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()

X_train_counts = count_vect.fit_transform(X_train)
X_train_counts.shape
```

```
(3733, 7082)
```

This shows that our training set is comprised of 3733 documents, and 7082 features.

After the counting, the term weighting and normalization can be done with TF-IDF, using scikit-learn's `TfidfTransformer`.

This downscaling is called **tf–idf** for "Term Frequency times Inverse Document Frequency".

Both tf and tf–idf can be computed as follows using TfidfTransformer:

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()

X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape
```

```
(3733, 7082)
```

Note: the `fit_transform()` method actually performs two operations: it fits an estimator to the data and then transforms our count-matrix to a tf-idf representation.

Combine Steps with TfidVectorizer

In the future, we can combine the CountVectorizer and TfidTransformer steps into one using TfidVectorizer:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()

X_train_tfidf = vectorizer.fit_transform(X_train) # remember to use the original X_train set
X_train_tfidf.shape
```

```
(3733, 7082)
```

Train a Classifier

Here we'll introduce an SVM classifier that's similar to SVC, called LinearSVC. LinearSVC handles sparse input better, and scales well to large numbers of samples.

```
from sklearn.svm import LinearSVC
clf = LinearSVC()
clf.fit(X_train_tfidf,y_train)
```

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Earlier we named our SVC classifier **svc_model**. Here we're using the more generic name **clf** (for classifier).

Build a Pipeline

Remember that only our training set has been vectorized into a full vocabulary. In order to perform an analysis on our test set we'll have to submit it to the same procedures. Fortunately scikit-learn offers a Pipeline class that behaves like a compound classifier.

```
from sklearn.pipeline import Pipeline
# from sklearn.feature_extraction.text import TfidfVectorizer
# from sklearn.svm import LinearSVC

text_clf = Pipeline([('tfidf', TfidfVectorizer()),
                     ('clf', LinearSVC()),
])

# Feed the training data through the pipeline
text_clf.fit(X_train, y_train)
```

```
Pipeline(memory=None,
         steps=[('tfidf',
                 TfidfVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.float64'>,
                                 encoding='utf-8', input='content',
                                 lowercase=True, max_df=1.0, max_features=None,
                                 min_df=1, ngram_range=(1, 1), norm='l2',
                                 preprocessor=None, smooth_idf=True,
                                 stop_words=None, strip_accents=None,
                                 sublinear_tf=False,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, use_idf=True,
                                 vocabulary=None)),
                ('clf',
                 LinearSVC(C=1.0, class_weight=None, dual=True,
                           fit_intercept=True, intercept_scaling=1,
                           loss='squared_hinge', max_iter=1000,
                           multi_class='ovr', penalty='l2', random_state=None,
                           tol=0.0001, verbose=0))],
         verbose=False)
```

Test the classifier and display results

```
# Form a prediction set
predictions = text_clf.predict(X_test)
```

```
# Report the confusion matrix
from sklearn import metrics
print(metrics.confusion_matrix(y_test,predictions))
```

```
[[1586    7]
 [  12  234]]
```

```
# Print a classification report
print(metrics.classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

         ham       0.99      1.00      0.99      1593
        spam       0.97      0.95      0.96       246

    accuracy                           0.99      1839
   macro avg       0.98      0.97      0.98      1839
weighted avg       0.99      0.99      0.99      1839
```

```
# Print the overall accuracy
print(metrics.accuracy_score(y_test,predictions))
```

```
0.989668297988037
```

Using the text of the messages, our model performed exceedingly well; it correctly predicted spam **98.97%** of the time!

```
text_clf.predict(["Hi! How are you doing today?"])
```

```
array(['ham'], dtype=object)
```

```
text_clf.predict(["Congratulations! You have won $10 Millions! Dial 1224 to claim."])
```

```
array(['spam'], dtype=object)
```