**ANNAMALAI UNIVERSITY**

## FACULTY OF ENGINEERING AND TECHNOLOGY

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**B.E. COMPUTER SCIENCE & ENGINEERING**

**(Artificial Intelligence and Machine Learning)**

**SEMESTER – VI**

# AICP607 - DEEP LEARNING TOOLS LAB

## LABORATORY RECORD

## (FEBRUARY 2022 – JUNE 2022)

**Name :……………………………………………….**

**Reg. No :…………………………………………….**

# ANNAMALAI UNIVERSITY

## FACULTY OF ENGINEERING AND TECHNOLOGY

### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**B.E. COMPUTER SCIENCE AND ENGINEERING**
**(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)**

**VI SEMESTER**

**AICP607 - DEEP LEARNING TOOLS LAB**

## Bonafide Certificate

*Certified that this is the Bonafide Record of work done by*

*Mr./Ms.* ____________________________________________

*Reg. No.* ____________________ *of VI semester B.E. Computer Science and*

*Engineering (Artificial Intelligence and Machine Learning) in the*

***AICP607 – Deep Learning Tools Lab*** *during the even semester*

*(February 2022 - June 2022 ).*

Staff In-Charge

Internal Examiner External Examiner

Place : Annamalai Nagar
Date :

## Vision and Mission of the Department

### VISION

To provide a congenial ambience for individuals to develop and blossom as academically superior, socially conscious and nationally responsible citizens.

### MISION

**M1:** Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.

**M2:** Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment

**M3:** Build student community with high ethical standards to undertake innovative research and development in thrust areas of national and international needs

**M4:** Expose the students to the emerging technological advancements for meeting the demands of the industry.

### Program Educational Objectives (PEOs)

| PEOs | PEO Statements |
|---|---|
| **PEO1** | To prepare graduates with potential to get employed in the right role and/or become entrepreneurs to contribute to the society. |
| **PEO2** | To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science. |
| **PEO3** | To equip the graduates with the skills required to stay motivated and adapt to the dynamically changing world so as to remain successful in their career. |
| **PEO4** | To train the graduates with effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility. |

**COURSE OUTCOMES:**

At the end of this course, the students will be able to

1. Create and manipulate tensors using Tensorflow tool and to understand tensorflow concepts.

2. Know supervised learning and working with features and labels.

3. Acquire knowledge on CNN, RNN.

| Mapping of Course Outcomes with Programme Outcomes | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** |
| **CO1** | - | - | 2 | - | 2 | - | - | - | - | - | - | - |
| **CO2** | - | 3 | 3 | 1 | 3 | 1 | - | - | - | - | - | 2 |
| **CO3** | 2 | 2 | - | - | - | - | - | - | - | 2 | - | 2 |

## Rubric for CO3

| Rubric for CO3 in Laboratory Courses | | | | |
|---|---|---|---|---|
| **Rubric** | **Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks** | | | |
| | **Up To 2.5 Marks** | **Up To 5 Marks** | **Up To 7.5 Marks** | **Up To 10 marks** |
| **Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.** | Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem. | Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors. | Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors. | Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description. |

INDEX

Ex No : 1
Date :

# Data Augumentation Pipeline Using Tensorflow

## Aim

To implement Data Augumentation Pipeline Using Tensorflow.

## Source Code

### Importing Modules

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

### Loading CIFAR-10 dataset using tensorflow.keras.datasets

```
from tensorflow.keras.datasets import cifar10
```

```
train_ds, test_ds = cifar10.load_data() # Load the datset
train_ds = tf.data.Dataset.from_tensor_slices(train_ds) # Creating a pipeline
with training set
```

### Utility function for displaying sample images

```
def show_images(images):
    n = len(images)
    for i,image in enumerate(images):
        plt.subplot(1,n,i+1)
        plt.imshow(image)
        plt.axis("off")
    plt.show()
```

## Method 1 - Data Augumentation using Keras preprocessing Layers (`tf.keras.layers`)

### Preprocessing with Rezising and Rescaling Layers

```
IMG_SIZE =32
resize_and_rescale = tf.keras.Sequential([
    tf.keras.layers.Resizing(IMG_SIZE,IMG_SIZE),
```

```
    tf.keras.layers.Rescaling(1/256) # 0-255 to 0-1
])
```

```
resized_images = train_ds.take(4).map(lambda x,y: resize_and_rescale(x))
show_images(resized_images)
```

**Other Augumentation Layers**

```
data_augumentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(),
    tf.keras.layers.RandomRotation(factor=.2),
    tf.keras.layers.RandomContrast(factor=(.2,.9),),
])
```

```
aug_images = resized_images.map(lambda x: data_augumentation(x))
show_images(aug_images)
```

**Applying keras preprocessing layer to the dataset**

```
AUTOTUNE = tf.data.AUTOTUNE
def prepare(ds):
    ds = ds.map(lambda x,y:
(resize_and_rescale(x),y),num_parallel_calls=AUTOTUNE)
    ds = ds.map(lambda x,y:
(data_augumentation(x),y),num_parallel_calls=AUTOTUNE)
    return ds.prefetch(buffer_size=AUTOTUNE)
```

```
prepared_ds = prepare(train_ds.take(4)) # take 4 elements from dataset and
prepare
show_images(prepared_ds.map(lambda x,y:x)) # removing y from dataset to
display the images
```

**Adding Augumenting Keras Layers to the model**

```
model = tf.keras.Sequential([
    resize_and_rescale,
    data_augumentation,
    tf.keras.layers.Conv2D(16,3),
    tf.keras.layers.MaxPooling2D(),
    # The rest of the model goes here
])
model.compile() ## This is not the full model . this shows how this can be
used
```

## Method 2 - Data Augumentation using `tf.image`

**Applying transformations in a single image**

```
image , label = next(iter(train_ds)) # take a single image from the dataset
```

```
show_images([
    tf.image.flip_left_right(image),
    tf.image.flip_up_down(image),
    tf.image.adjust_brightness(image,-.2),
    tf.image.adjust_contrast(image,2),
    tf.image.adjust_saturation(image,2),
]) # manual augumentation

show_images([
    tf.image.stateless_random_contrast(image, lower=.1,upper=5,seed = (i,0))
    for i in range(1,6)
]) # random contrast change with different seeds
show_images([
    tf.image.stateless_random_brightness(image, max_delta=.8,seed = (i,0))
    for i in range(1,6)
]) # random brightness change with different seeds

# Note: seed is a Tensor of shape (2,) whose values are any integers
```

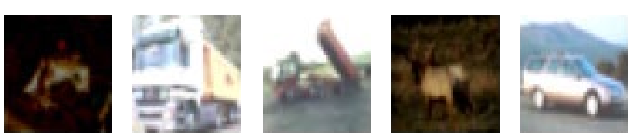
**Applying augumentations to the dataset**

```
def resize_and_rescale(img):
    img = tf.cast(img,tf.float32) # convert to float
    img = tf.image.resize(img,(32,32))
    img = img/255.0 # rescaling 0-255 to 0-1
    return img

def augument(img_label,seed):
    image,label = img_label
    image = resize_and_rescale(image)

    # Random crop back to the original size.
    image = tf.image.stateless_random_crop(image, size=[IMG_SIZE, IMG_SIZE,
3], seed=seed)
    # Random brightness.
    image = tf.image.stateless_random_brightness( image, max_delta=0.5,
seed=seed)
    image = tf.clip_by_value(image, 0, 1)
    return image,label
```

```
# Create a random number generator
rng = tf.random.Generator.from_seed(1,alg="philox")
```

```
# create a function to create seeds for the augument function
def augument_wrapper(image,label):
    seed = rng.make_seeds(2)[0] # stateless random ops require two seeds
    image,label = augument((image,label),seed)
    return image,label
```

```
augumented_ds = train_ds.take(5).map(augument_wrapper)
show_images(augumented_ds.map(lambda x,y : x)) # seperate images from the
dataset and display
```



## Result

Thus Data Augumentation Pipeline Using Tensorflow is implemented.

**Ex No : 2**

**Date :**

# Data Pipelining

## Aim

To implement Data Pipelining using Tensorflow.

## Source Code

```
import tensorflow as tf
import pathlib
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

### Dataset from tensor in memory

```
t1 = tf.constant([
    [1,2,3],
    [4,5,6],
    [7,8,9],
],dtype=tf.float32)
ds1 = tf.data.Dataset.from_tensors(t1) # use the tensor as the element
ds2 = tf.data.Dataset.from_tensor_slices(t1) # uses elements of a tensor as
elements
```

```
for element in ds1:
    print(element)
```

```
tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]], shape=(3, 3), dtype=float32)
```

```
for element in ds2:
    print(element)
```

```
tf.Tensor([1. 2. 3.], shape=(3,), dtype=float32)
tf.Tensor([4. 5. 6.], shape=(3,), dtype=float32)
tf.Tensor([7. 8. 9.], shape=(3,), dtype=float32)
```

### Dataset from numpy arrays

**Downloading dataset using `tf.keras.datasets` API**

```
train, test = tf.keras.datasets.fashion_mnist.load_data() # load the fashion
mnist data
```

```
images, labels = train
images = images/255
type(images),type(labels)
```

```
(numpy.ndarray, numpy.ndarray)
```

```
dataset = tf.data.Dataset.from_tensor_slices((images, labels)) # images as X
and labels as y
dataset
```

```
<TensorSliceDataset element_spec=(TensorSpec(shape=(28, 28),
dtype=tf.float64, name=None), TensorSpec(shape=(), dtype=tf.uint8,
name=None))>
```

## TextLine Dataset

### Downloading the dataset

```
directory_url =
'https://storage.googleapis.com/download.tensorflow.org/data/illiad/'
file_names = ['cowper.txt', 'derby.txt', 'butler.txt']

file_paths = [
    tf.keras.utils.get_file(file_name, directory_url + file_name) # download
each file using tf.keras.get_file
    for file_name in file_names
]
```

```
Downloading data from
https://storage.googleapis.com/download.tensorflow.org/data/illiad/cowper.txt
819200/815980 [==============================] - 0s 0us/step
827392/815980 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/download.tensorflow.org/data/illiad/derby.txt
811008/809730 [==============================] - 0s 0us/step
819200/809730 [==============================] - 0s 0us/step
Downloading data from
https://storage.googleapis.com/download.tensorflow.org/data/illiad/butler.txt
811008/807992 [==============================] - 0s 0us/step
819200/807992 [==============================] - 0s 0us/step
```

### Creating and viewing the dataset

```
text_line_dataset = tf.data.TextLineDataset(file_paths)
```

```
for line in text_line_dataset.take(5):
    print(line.numpy()) # elements are of type tensor so converted to numpy
type
```

```
b"\xef\xbb\xbfAchilles sing, O Goddess! Peleus' son;"
b'His wrath pernicious, who ten thousand woes'
b"Caused to Achaia's host, sent many a soul"
b'Illustrious into Ades premature,'
b'And Heroes gave (so stood the will of Jove)'
```

## Text dataset from folder

### Downloading the text directory dataset

```
data_url =
'https://storage.googleapis.com/download.tensorflow.org/data/stack_overflow_1
6k.tar.gz'

dataset_dir = tf.keras.utils.get_file(
    origin=data_url,
    untar=True, # data set is a tar.gz file so untar is used to uncompress
the dataset
    cache_dir= "stack_overflow",
    cache_subdir = ""
)
print(dataset_dir)
dataset_dir = pathlib.Path(dataset_dir).parent
train_dir = dataset_dir/'train'
train_dir
```

```
/tmp/.keras/stack_overflow_16k

PosixPath('/tmp/.keras/train')
```

```
batch_size = 32
seed = 42

raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    train_dir,
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed
)
```

```
Found 8000 files belonging to 4 classes.
Using 6400 files for training.
```

## CSV Datasets

### Downloading titanic dataset

```
titanic_file = tf.keras.utils.get_file("train.csv",
"https://storage.googleapis.com/tf-datasets/titanic/train.csv")
```

### Loading CSV using *pandas*

```
df = pd.read_csv(titanic_file)
titanic_dataset = tf.data.Dataset.from_tensor_slices(dict(df))
```

```
for feature_batch in titanic_dataset.take(1):
    for key,value in feature_batch.items():
        print("  {!r:20s}: {}".format(key,value))
```

```
'survived'          : 0
'sex'               : b'male'
'age'               : 22.0
'n_siblings_spouses': 1
'parch'             : 0
'fare'              : 7.25
'class'             : b'Third'
'deck'              : b'unknown'
'embark_town'       : b'Southampton'
'alone'             : b'n'
```

**Loading CSV using `tf.data.experimental.make_csv_dataset`**

```
titanic_batches = tf.data.experimental.make_csv_dataset(
    titanic_file,
    batch_size=4,# Setting the batch size
    label_name="survived", # selecting the label column
    select_columns=['class', 'fare', 'survived']
)
```

```
for feature_batch, label_batch in titanic_batches.take(1):
    print(f"Survived: {label_batch}")
    for key,value in feature_batch.items():
        print(f"{key:20s}: {value}")
```

```
Survived: [1 0 1 0]
fare                : [120.      27.9     91.0792  27.9    ]
class               : [b'First' b'Third' b'First' b'Third']
```

**Loading CSV using `tf.data.experimetal.CsvDataset`**

```
titanic_types = [tf.int32, tf.string, tf.float32, tf.int32, tf.int32,
tf.float32, tf.string, tf.string, tf.string, tf.string]
dataset = tf.data.experimental.CsvDataset(titanic_file, titanic_types ,
header=True)

for line in dataset.take(10):
  print([item.numpy() for item in line])
```

```
[0, b'male', 22.0, 1, 0, 7.25, b'Third', b'unknown', b'Southampton', b'n']
[1, b'female', 38.0, 1, 0, 71.2833, b'First', b'C', b'Cherbourg', b'n']
[1, b'female', 26.0, 0, 0, 7.925, b'Third', b'unknown', b'Southampton', b'y']
[1, b'female', 35.0, 1, 0, 53.1, b'First', b'C', b'Southampton', b'n']
[0, b'male', 28.0, 0, 0, 8.4583, b'Third', b'unknown', b'Queenstown', b'y']
[0, b'male', 2.0, 3, 1, 21.075, b'Third', b'unknown', b'Southampton', b'n']
[1, b'female', 27.0, 0, 2, 11.1333, b'Third', b'unknown', b'Southampton',
b'n']
[1, b'female', 14.0, 1, 0, 30.0708, b'Second', b'unknown', b'Cherbourg',
b'n']
[1, b'female', 4.0, 1, 1, 16.7, b'Third', b'G', b'Southampton', b'n']
[0, b'male', 20.0, 0, 0, 8.05, b'Third', b'unknown', b'Southampton', b'y']
```

## Dataset from sets of files

### Downloading flower dataset

```
flowers_root = tf.keras.utils.get_file(
    'flower_photos',

'https://storage.googleapis.com/download.tensorflow.org/example_images/flower
_photos.tgz',
    untar=True
)
flowers_root = pathlib.Path(flowers_root)
```

### Creating the dataset from flies

```
for item in flowers_root.glob("*"):
    print(item)
```

```
/root/.keras/datasets/flower_photos/tulips
/root/.keras/datasets/flower_photos/sunflowers
/root/.keras/datasets/flower_photos/LICENSE.txt
/root/.keras/datasets/flower_photos/roses
/root/.keras/datasets/flower_photos/daisy
/root/.keras/datasets/flower_photos/dandelion
```

```
file_path_ds = tf.data.Dataset.list_files(str(flowers_root/'*/*'))
```

```
def process_path(file_path):
    # folder name or label will be the second last value
    # when seperated using tf.strings.split with os.sep as seperator
    label = tf.strings.split(file_path,os.sep)[-2]
    return tf.io.read_file(file_path),label
```

```
labeled_ds = file_path_ds.map(process_path)
```

```
for image_raw, label in labeled_ds.take(1):
    print(image_raw,label,sep="\n") # image is loaded in binary and not as
numpy array
```

```
tf.Tensor(b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\
x00\xff\xe2\x0cXICC_PROFILE\x00\x01\x01\x00\x00\x0cHLino\x02\x10\x00\x00mntrR
GB XYZ \x07\xce\x00\x02\x00\t\x00\x06\x001\x00\x00acspMSFT\x00\x00\x00\x00IEC
sRGB\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xf6\xd6\x00\x01\
...

...
\x02T\x00\x00\\<\x00\x00\x08\x0cbTRC\x00\x00\x04<\x00\x00\x08\x0ctext\x00\x00
\x00\x00Copyright (c) 1998 Hewlett-Packard Company\x00\x00desc\x00\x00\x00\,
shape=(), dtype=string)
tf.Tensor(b'dandelion', shape=(), dtype=string)
```

## Batching datasets

```
# sample datasets
inc_dataset = tf.data.Dataset.range(100)
```

```
dec_dataset = tf.data.Dataset.range(0, -100, -1)
dataset = tf.data.Dataset.zip((inc_dataset, dec_dataset)) # joining both inc
and dec

# batching the dataset
batched_dataset = dataset.batch(4) # try with a different batch size

for batch in batched_dataset.take(4):
  print([arr.numpy() for arr in batch])
```

```
[array([0, 1, 2, 3]), array([ 0, -1, -2, -3])]
[array([4, 5, 6, 7]), array([-4, -5, -6, -7])]
[array([ 8,  9, 10, 11]), array([ -8,  -9, -10, -11])]
[array([12, 13, 14, 15]), array([-12, -13, -14, -15])]
```

```
batched_dataset
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None,), dtype=tf.int64,
name=None), TensorSpec(shape=(None,), dtype=tf.int64, name=None))>
```

```
dataset.batch(4,drop_remainder = True)
```

```
<BatchDataset element_spec=(TensorSpec(shape=(4,), dtype=tf.int64,
name=None), TensorSpec(shape=(4,), dtype=tf.int64, name=None))>
```

**Padded Batching**

```
dataset = tf.data.Dataset.range(100)
dataset = dataset.map(lambda x: tf.fill([tf.cast(x, tf.int32)], x))
padded_batch_dataset = dataset.padded_batch(4, padded_shapes=(None,)) #
(None,) uses the largest size as padding

for batch in padded_batch_dataset.take(2):
  print(batch.numpy())
  print()
```

```
[[0 0 0]
 [1 0 0]
 [2 2 0]
 [3 3 3]]

[[4 4 4 4 0 0 0]
 [5 5 5 5 5 0 0]
 [6 6 6 6 6 6 0]
 [7 7 7 7 7 7 7]]
```

## Shuffling Dataset

```
dataset = tf.data.TextLineDataset(titanic_file)

dataset.shuffle(buffer_size=10)
```

```
<ShuffleDataset element_spec=TensorSpec(shape=(), dtype=tf.string,
name=None)>
```

## Preprocessing Data

### Using `tf.map` to apply preporcessing

```
file_path_ds = tf.data.Dataset.list_files(str(flowers_root/'*/*'))
```

```
def parse_image(filename):
    label = tf.strings.split(filename, os.sep)[-2]

    image = tf.io.read_file(filename)
    image = tf.io.decode_jpeg(image)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, [128, 128])
    return image, label
```

```
image_ds = file_path_ds.map(parse_image)
```

```
def show(image,label):
    plt.imshow(image)
    plt.title(label.numpy().decode("utf-8"))
    plt.axis("off")
    plt.show()
```

```
for image,label in image_ds.take(3):
    show(image,label)
```




## Using tf.data with tf.keras

```
train, test = tf.keras.datasets.fashion_mnist.load_data()

images, labels = train
images = images/255.0
labels = labels.astype(np.int32)
```

### Creating the dataset and the model

```
fmnist_train_ds = tf.data.Dataset.from_tensor_slices((images, labels))
fmnist_train_ds = fmnist_train_ds.shuffle(5000).batch(32)

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(),
```

```
  tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

**Fiting the model**

```
model.fit(fmnist_train_ds, epochs=2)
```

```
Epoch 1/2
1875/1875 [==============================] - 11s 5ms/step - loss: 0.5968 - accuracy: 0.7998
Epoch 2/2
1875/1875 [==============================] - 9s 5ms/step - loss: 0.4621 - accuracy: 0.8416
```

**Evaluating the model**

```
loss, accuracy = model.evaluate(fmnist_train_ds)
print("Loss :", loss)
print("Accuracy :", accuracy)
```

```
1875/1875 [==============================] - 7s 3ms/step - loss: 0.4403 - accuracy: 0.8502
Loss : 0.44031789898872375
Accuracy : 0.8502166867256165
```

**Using the model to predict**

```
predict_ds = tf.data.Dataset.from_tensor_slices(images).batch(32) # creating a dataset with only images
result = model.predict(predict_ds, steps = 10)
print(result.shape)
```

```
(320, 10)
```

## Result

Thus Data Pipelining is implemented using Tensorflow.

**Ex No : 3**
**Date :**

# Deep Neural Networks

## Aim

To Implement Deep Neural Networks using Tensorflow.

## Source Code

### Classification using Deep Neural Networks

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

### Loading Dataset

```
titanic_file_path = tf.keras.utils.get_file("train.csv",
"https://storage.googleapis.com/tf-datasets/titanic/train.csv")
```

```
df = pd.read_csv(titanic_file_path)
df.head()
```

| | survived | sex | age | n_siblings_spouses | parch | fare | class | deck | embark_town | alone |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | male | 22.0 | 1 | 0 | 7.2500 | Third | unknown | Southampton | n |
| **1** | 1 | female | 38.0 | 1 | 0 | 71.2833 | First | C | Cherbourg | n |
| **2** | 1 | female | 26.0 | 0 | 0 | 7.9250 | Third | unknown | Southampton | y |
| **3** | 1 | female | 35.0 | 1 | 0 | 53.1000 | First | C | Southampton | n |
| **4** | 0 | male | 28.0 | 0 | 0 | 8.4583 | Third | unknown | Queenstown | y |

```
df.rename(columns = {"survived":"target"},inplace=True)
np.random.seed(5)

train, val, test = np.split(df.sample(frac=1), [int(0.8*len(df)),
int(0.9*len(df))])

train
```

| | target | sex | age | n_siblings_spouses | parch | fare | class | deck | embark_town | alone |
|---|---|---|---|---|---|---|---|---|---|---|
| **445** | 0 | male | 57.0 | 0 | 0 | 12.3500 | Second | unknown | Queenstown | y |
| **230** | 1 | female | 31.0 | 0 | 2 | 164.8667 | First | C | Southampton | n |
| **289** | 1 | male | 39.0 | 0 | 0 | 7.9250 | Third | unknown | Southampton | y |
| **622** | 0 | male | 28.0 | 0 | 0 | 10.5000 | Second | unknown | Southampton | y |
| **361** | 0 | female | 37.0 | 0 | 0 | 9.5875 | Third | unknown | Southampton | y |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **572** | 0 | male | 35.0 | 0 | 0 | 10.5000 | Second | unknown | Southampton | y |
| **54** | 0 | male | 26.0 | 2 | 0 | 8.6625 | Third | unknown | Southampton | n |
| **19** | 0 | male | 28.0 | 0 | 0 | 7.2250 | Third | unknown | Cherbourg | y |
| **609** | 1 | female | 27.0 | 1 | 0 | 13.8583 | Second | unknown | Cherbourg | n |
| **178** | 0 | male | 22.0 | 0 | 0 | 7.1250 | Third | unknown | Southampton | y |

501 rows × 10 columns

```
def df_to_dataset(dataframe, shuffle=True, batch_size=32):
  df = dataframe.copy()
  labels = df.pop('target')
  df = {key: value.values[:,tf.newaxis] for key, value in dataframe.items()}
  ds = tf.data.Dataset.from_tensor_slices((dict(df), labels))
  if shuffle:
    ds = ds.shuffle(buffer_size=len(dataframe))
  ds = ds.batch(batch_size)
  ds = ds.prefetch(batch_size)
  return ds
```

```
batch_size = 10
train_ds = df_to_dataset(train, batch_size=batch_size)
val_ds = df_to_dataset(val, batch_size=batch_size)
test_ds = df_to_dataset(test, batch_size=batch_size)
```

**Preprocessing the dataset**

```
def get_normalization_layer(name, dataset):
  # Create a Normalization layer for the feature.
  normalizer = tf.keras.layers.Normalization(axis=None)

  # Prepare a Dataset that only yields the feature.
  feature_ds = dataset.map(lambda x, y: x[name])

  # Learn the statistics of the data.
  normalizer.adapt(feature_ds)
  return normalizer
```

```
def get_category_encoding_layer(name, dataset, dtype, max_tokens=None):
  # Create a layer that turns strings into integer indices.
  if dtype == 'string':
    index = tf.keras.layers.StringLookup(max_tokens=max_tokens)
  # Otherwise, create a layer that turns integer values into integer indices.
  else:
    index = tf.keras.layers.IntegerLookup(max_tokens=max_tokens)

  # Prepare a `tf.data.Dataset` that only yields the feature.
  feature_ds = dataset.map(lambda x, y: x[name])

  # Learn the set of possible values and assign them a fixed integer index.
  index.adapt(feature_ds)

  # Encode the integer indices.
  encoder =
tf.keras.layers.CategoryEncoding(num_tokens=index.vocabulary_size())
  # Apply multi-hot encoding to the indices. The lambda function captures the
  # layer, so you can use them, or include them in the Keras Functional model
later.
  return lambda feature: encoder(index(feature))
```

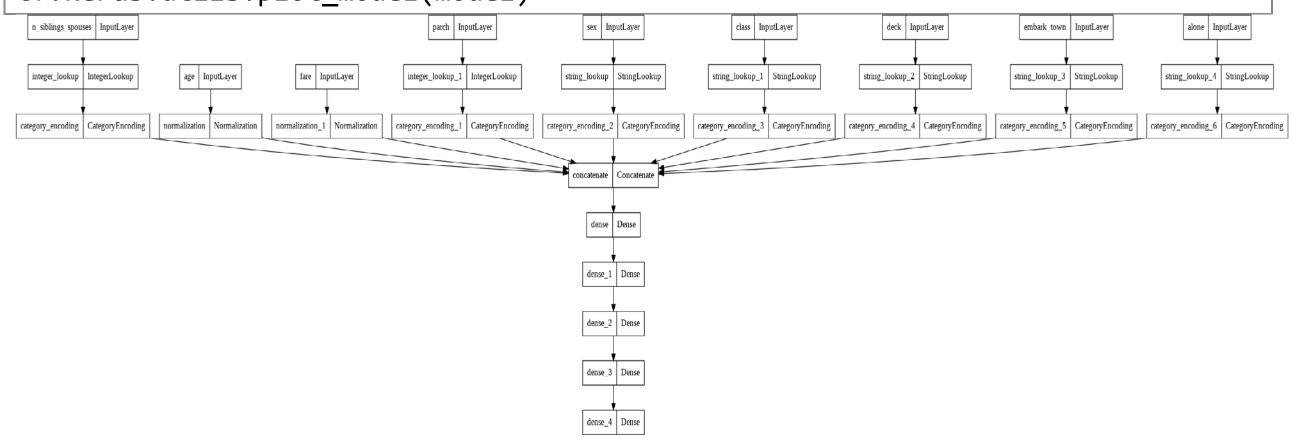
```
numerical_cols = ["age","fare"]
numerical_categorical_cols = ["n_siblings_spouses","parch"]
categorical_cols = ["sex","class","deck","embark_town","alone"]

all_inputs = []
encoded_features = []

# Numerical features.
for header in numerical_cols:
  numeric_col = tf.keras.Input(shape=(1,), name=header)
  normalization_layer = get_normalization_layer(header, train_ds) #
Normalization
  encoded_numeric_col = normalization_layer(numeric_col)
  all_inputs.append(numeric_col)
  encoded_features.append(encoded_numeric_col)

# Numerical Categorial features
for header in numerical_categorical_cols:
  categorical_col = tf.keras.Input(shape=(1,), name=header, dtype='int64')
  encoding_layer =
get_category_encoding_layer(name=header,dataset=train_ds,dtype='int64') #
encoding
  encoded_categorical_col = encoding_layer(categorical_col)
  all_inputs.append(categorical_col)
  encoded_features.append(encoded_categorical_col)

# Other categorical Features
```

```
for header in categorical_cols:
  categorical_col = tf.keras.Input(shape=(1,), name=header, dtype='string')
  encoding_layer =
get_category_encoding_layer(name=header,dataset=train_ds,dtype='string',max_t
okens=5) # encoding
  encoded_categorical_col = encoding_layer(categorical_col)
  all_inputs.append(categorical_col)
  encoded_features.append(encoded_categorical_col)
```

**Creating the model**

```
x  =  tf.keras.layers.concatenate(encoded_features)
x = tf.keras.layers.Dense(32, activation="relu")(x)
x = tf.keras.layers.Dense(8, activation="relu")(x)
x = tf.keras.layers.Dense(4, activation="relu")(x)
x = tf.keras.layers.Dense(2, activation="relu")(x)
outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)
model = tf.keras.Model(all_inputs,outputs)
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
    metrics=["accuracy"]
)
```

```
tf.keras.utils.plot_model(model)
```



**Training the model**

```
history = model.fit(train_ds,validation_data=val_ds,epochs=50)
```

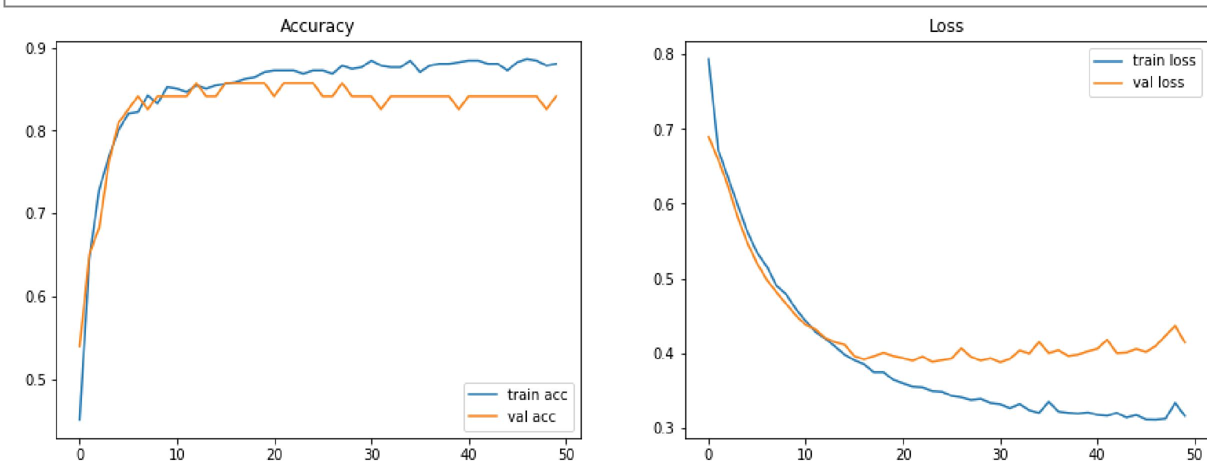
```
Epoch 1/50

inputs = self._flatten_to_reference_inputs(inputs)

51/51 [==============================] - 2s 11ms/step - loss: 0.7937 -
accuracy: 0.4511 - val_loss: 0.6897 - val_accuracy: 0.5397
Epoch 2/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6718 -
accuracy: 0.6447 - val_loss: 0.6590 - val_accuracy: 0.6508
Epoch 49/50
51/51 [==============================] - 0s 4ms/step - loss: 0.3333 -
```

```
accuracy: 0.8782 - val_loss: 0.4364 - val_accuracy: 0.8254
Epoch 50/50
51/51 [==============================] - 0s 4ms/step - loss: 0.3159 -
accuracy: 0.8802 - val_loss: 0.4143 - val_accuracy: 0.8413
```

**Ploting learing curves**

```
history = history.history
```

```
plt.figure(figsize=(15,5))
plt.subplot(121)
plt.title("Accuracy")
plt.plot(history["accuracy"],label="train acc")
plt.plot(history["val_accuracy"],label="val acc")
plt.legend()
plt.subplot(122)
plt.title("Loss")
plt.plot(history["loss"],label="train loss")
plt.plot(history["val_loss"],label="val loss")
plt.legend()
plt.show()
```




**Testing the model on test set**

```
loss ,accuracy = model.evaluate(test_ds)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.6833 - accuracy:
0.7143
```

```
print("test loss :",loss)
print("test accuracy :",accuracy)
```

```
test loss : 0.68325275182724
test accuracy : 0.7142857313156128
```

## Result

Thus Deep Neural Networks has been Implemented using Tensorflow.

**Ex No : 4**
**Date :**

# Sentence Classification using 1D CNN

## Aim

To Implement Sentence Classification using 1D CNN.

## Source Code

### Importing the required modules

```
import matplotlib.pyplot as plt
import os
import re
import shutil
import string
import tensorflow as tf
from tensorflow.keras import layers
```

### Dataset

#### Downloading the dataset

```
url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

dataset = tf.keras.utils.get_file(
    "aclImdb_v1",  url,
    untar=True, cache_dir='.',
    cache_subdir=''
)

dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')
```

#### Exploring the dataset directory

```
os.listdir(dataset_dir)
```

```
['imdb.vocab', 'test', 'README', 'imdbEr.txt', 'train']
```

```
train_dir = os.path.join(dataset_dir,"train")
os.listdir(train_dir)
```

```
['neg',
 'pos',
 'urls_neg.txt',
 'labeledBow.feat',
 'unsupBow.feat',
 'urls_unsup.txt',
```

```
 'unsup',
 'urls_pos.txt']
```

Here **pos** and **neg** are the folders containing positive and negative reviews.

**Removing unwanted folders from training folder**

```
shutil.rmtree(os.path.join(train_dir,"unsup"))
```

**Loading the dataset**

```
batch_size = 32
seed = 42

raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 20000 files for training.
```

```
raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 5000 files for validation.
```

```
raw_test_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/test',
    batch_size=batch_size)
```

```
Found 25000 files belonging to 2 classes.
```

## Preprocessing the text

**Standardization Function**

```
def custom_standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ')
    return tf.strings.regex_replace(
        stripped_html,
        f'[{re.escape(string.punctuation)}]',
        ''
    )
```

**Text Vectorization**

```
max_features = 10000
sequence_length = 250

vectorize_layer = layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)
```

```
train_text = raw_train_ds.map(lambda x, y: x)
vectorize_layer.adapt(train_text)
```

```
def vectorize_text(text,label):
    text = tf.expand_dims(text,-1) # add an extra dimension to the text
    return vectorize_layer(text),label
```

**Applying Text vectroization to the dataset and configuring it for performance**

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds =
raw_train_ds.map(vectorize_text).cache().prefetch(buffer_size=AUTOTUNE)
val_ds =
raw_val_ds.map(vectorize_text).cache().prefetch(buffer_size=AUTOTUNE)
test_ds =
raw_test_ds.map(vectorize_text).cache().prefetch(buffer_size=AUTOTUNE)
```

## Creating the model

```
embedding_dim = 16

model = tf.keras.Sequential([
  layers.Embedding(max_features + 1, embedding_dim),
  layers.Conv1D(8,7,activation="relu"),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(8,activation="relu"),
  layers.Dense(1)])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 16)          160016

 conv1d (Conv1D)             (None, None, 8)           904

 global_average_pooling1d (G  (None, 8)                0
 lobalAveragePooling1D)
```

```
 dropout (Dropout)           (None, 8)                 0

 dense (Dense)               (None, 8)                 72

 dense_1 (Dense)             (None, 1)                 9

=================================================================
Total params: 161,001
Trainable params: 161,001
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

```
model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir="logs")
    ]
)
```

```
Epoch 1/10
625/625 [==============================] - 12s 16ms/step - loss: 0.5317 -
accuracy: 0.6665 - val_loss: 0.3289 - val_accuracy: 0.8526
Epoch 2/10
625/625 [==============================] - 5s 7ms/step - loss: 0.3112 -
accuracy: 0.8682 - val_loss: 0.2876 - val_accuracy: 0.8684
...

...
Epoch 9/10
625/625 [==============================] - 4s 7ms/step - loss: 0.1075 -
accuracy: 0.9590 - val_loss: 0.3699 - val_accuracy: 0.8718
Epoch 10/10
625/625 [==============================] - 4s 7ms/step - loss: 0.0970 -
accuracy: 0.9632 - val_loss: 0.3856 - val_accuracy: 0.8712
```

## Visualizing the training process with tensorboard

```
%load_ext tensorboard
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 1142), started 0:11:31 ago. (Use '!kill
1142' to kill it.)

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

### Evaluvating the model

```
loss, accuracy = model.evaluate(test_ds)

print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
782/782 [==============================] - 9s 11ms/step - loss: 0.4483 -
accuracy: 0.8452
Loss:  0.44825002551078796
Accuracy:  0.8451600074768066
```

### Exporting the model

```
export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])

export_model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
optimizer="adam", metrics=['accuracy']
)


loss, accuracy = export_model.evaluate(raw_test_ds)
print(accuracy)
```

```
782/782 [==============================] - 9s 11ms/step - loss: 0.4482 -
accuracy: 0.8515
0.8515200018882751
```

```
export_model.save("sentence_classification_model")
```

```
INFO:tensorflow:Assets written to: sentence_classification_model/assets
```

## Result

Thus Sentence Classification using 1D CNN is implemented.

**Ex No : 5**
**Date :**

# Age Prediction Using 2D CNN

## Aim

To implement Age Prediction Using 2D CNN.

## Source Code

### Importing Modules

```
import tensorflow as tf
import tensorflow.keras.layers as tfl
import os
```

### Dataset

```
url = "https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-
wiki/static/wiki_crop.tar"

dataset = tf.keras.utils.get_file(
    "wiki_crop", url,
    untar=True, cache_dir='.',
    cache_subdir=''
)
dataset_dir = os.path.join(os.path.dirname(dataset), 'wiki_crop')
```

```
Downloading data from https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-
wiki/static/wiki_crop.tar
811319296/811315200 [==============================] - 67s 0us/step
811327488/811315200 [==============================] - 67s 0us/step
```

### Loading and extracting the age from the meta data file

```
import scipy.io
# extract data from wiki.mat
mat = scipy.io.loadmat(os.path.join(dataset_dir,'wiki.mat'))

import numpy as np
import datetime


mat
```

```
{'__globals__': [],
 '__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Sat Jan
16 16:25:20 2016',
 '__version__': '1.0',
 'wiki': array([[(array([[723671, 703186, 711677, ..., 720620, 723893,
713846]], dtype=int32), array([[2009, 1964, 2008, ..., 2013, 2011, 2008]],
```

```
dtype=uint16), array([[array(['17/10000217_1981-05-05_2009.jpg'],
dtype='<U31'),
                 array(['48/10000548_1925-04-04_1964.jpg'], dtype='<U31'),
                 array(['12/100012_1948-07-03_2008.jpg'], dtype='<U29'), ...,
                 array(['09/9998109_1972-12-27_2013.jpg'], dtype='<U30'),
                 array(['00/9999400_1981-12-13_2011.jpg'], dtype='<U30'),
                 array(['80/999980_1954-06-11_2008.jpg'], dtype='<U29')]],
               dtype=object), array([[1., 1., 1., ..., 1., 1., 0.]]),
array([[array(['Sami Jauhojärvi'], dtype='<U15'),
                 array(['Dettmar Cramer'], dtype='<U14'),
                 array(['Marc Okrand'], dtype='<U11'), ...,
                 array(['Michael Wiesinger'], dtype='<U17'),
                 array(['Johann Grugger'], dtype='<U14'),
                 array(['Greta Van Susteren'], dtype='<U18')]],
dtype=object), array([[array([[111.29109473, 111.29109473, 252.66993082,
252.66993082]]),
                 array([[252.4833023 , 126.68165115, 354.53192596,
228.73027481]]),
                 array([[113.52, 169.84, 366.08, 422.4 ]]), ...,
                 array([[169.88839786,  74.31669472, 235.2534231 ,
139.68171997]]),
                 array([[1, 1, 1, 1]], dtype=uint8),
                 array([[ 92.72633235,  62.0435549 , 230.12083087,
199.43805342]])]],
               dtype=object), array([[4.30096239, 2.6456395 , 4.32932883,
..., 3.49430317,        -inf,
                 5.48691655]]), array([[       nan, 1.94924791,        nan,
...,        nan,        nan,
                        nan]]))
]],
       dtype=[('dob', 'O'), ('photo_taken', 'O'), ('full_path', 'O'),
('gender', 'O'), ('name', 'O'), ('face_location', 'O'), ('face_score', 'O'),
('second_face_score', 'O')])}
```

```python
mat["wiki"]["dob"][0][0][0]
```

```
array([723671, 703186, 711677, ..., 720620, 723893, 713846], dtype=int32)
```

```python
dob = np.vectorize(lambda x: datetime.datetime.fromordinal(x).year)(
    mat["wiki"]["dob"][0][0][0]
)
photo_taken = mat["wiki"]["photo_taken"][0][0][0]

age = (photo_taken-dob).astype(np.float32)

age
```

```
array([27., 38., 59., ..., 40., 29., 53.], dtype=float32)
```

```python
mat["wiki"]["full_path"][0][0][0]
```

```
array([array(['17/10000217_1981-05-05_2009.jpg'], dtype='<U31'),
       array(['48/10000548_1925-04-04_1964.jpg'], dtype='<U31'),
       array(['12/100012_1948-07-03_2008.jpg'], dtype='<U29'), ...,
       array(['09/9998109_1972-12-27_2013.jpg'], dtype='<U30'),
       array(['00/9999400_1981-12-13_2011.jpg'], dtype='<U30'),
       array(['80/999980_1954-06-11_2008.jpg'], dtype='<U29')],
      dtype=object)
```

```
file_path = np.vectorize(lambda x : os.path.join(dataset_dir,x[0]))(
    mat["wiki"]["full_path"][0][0][0]
)

file_path
```

```
array(['./wiki_crop/17/10000217_1981-05-05_2009.jpg',
       './wiki_crop/48/10000548_1925-04-04_1964.jpg',
       './wiki_crop/12/100012_1948-07-03_2008.jpg', ...,
       './wiki_crop/09/9998109_1972-12-27_2013.jpg',
       './wiki_crop/00/9999400_1981-12-13_2011.jpg',
       './wiki_crop/80/999980_1954-06-11_2008.jpg'], dtype='<U49')
```

```
file_age_ds = tf.data.Dataset.from_tensor_slices((file_path,age))
```

```
def parse_function(filename, label):
    image_string = tf.io.read_file(filename)
    image_decoded = tf.io.decode_jpeg(image_string,channels=1)
    image = tf.image.resize(image_decoded, [256, 256])
    return image, tf.expand_dims(label,0)
```

```
image_age_ds=file_age_ds.map(parse_function).shuffle(seed=2,buffer_size=64)
image_age_ds
```

```
<ShuffleDataset element_spec=(TensorSpec(shape=(256, 256, 1),
dtype=tf.float32, name=None), TensorSpec(shape=(1,), dtype=tf.float32,
name=None))>
```

```
dataset_size = image_age_ds.cardinality().numpy()

AUTOTUNE = tf.data.AUTOTUNE
train_ds = image_age_ds.take(dataset_size*.6).batch(32).prefetch(AUTOTUNE)
val_ds =
image_age_ds.skip(dataset_size*.6).take(dataset_size*.2).batch(32).prefetch(A
UTOTUNE)
test_ds =
image_age_ds.skip(dataset_size*.8).take(dataset_size*.2).batch(32).prefetch(A
UTOTUNE)
```

## Creating the model

```
model = tf.keras.Sequential([

tfl.Conv2D(32,(7,7),padding="valid",activation="relu",input_shape=(256,256,1)
```

```
),
    tfl.MaxPool2D((4,4),strides = 4),
    tfl.Conv2D(64,(3,3),padding = "valid",activation="relu"),
    tfl.MaxPool2D((4,4),strides = 4),
    tfl.Conv2D(128,(3,3),padding = "valid",activation="relu"),
    tfl.MaxPool2D((2,2),strides = 2),
    tfl.Conv2D(256,(1,1),padding= "valid",activation="relu",),
    tfl.MaxPool2D((2,2),strides = 2),
    tfl.Flatten(),
    tfl.Dense(64,activation="relu"),
    tfl.Dense(1)
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 250, 250, 32)      1600

 max_pooling2d (MaxPooling2D  (None, 62, 62, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 60, 60, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 15, 15, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 6, 6, 256)         33024

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 64)                147520

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 274,561
Trainable params: 274,561
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.MeanAbsoluteError(),
    metrics=['MAE']
)

model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir="logs")
    ]
)
```

```
Epoch 1/10
1169/1169 [==============================] - 116s 91ms/step - loss: 14.2855 - MAE: 14.2855 - val_loss: 12.7515 - val_MAE: 12.7515
Epoch 2/10
1169/1169 [==============================] - 106s 91ms/step - loss: 13.4699 - MAE: 13.4699 - val_loss: 12.2243 - val_MAE: 12.2243
...
Epoch 9/10
1169/1169 [==============================] - 105s 90ms/step - loss: 11.7158 - MAE: 11.7158 - val_loss: 12.5963 - val_MAE: 12.5963
Epoch 10/10
1169/1169 [==============================] - 105s 89ms/step - loss: 11.6125 - MAE: 11.6125 - val_loss: 11.1081 - val_MAE: 11.1081

<keras.callbacks.History at 0x7f7dd0a1f210>
```

### Visualizing the training process with tensorboard

```
%load_ext tensorboard
%tensorboard --logdir logs
```

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

### Evaluating the model

```
loss, accuracy = model.evaluate(test_ds)

print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
390/390 [==============================] - 42s 30ms/step - loss: 11.4170 - MAE: 11.4170
Loss:  11.41700267791748
Accuracy:  11.41700267791748
```

## Result

Thus Age Prediction Using 2D CNN is Implemented.

**Ex No : 6**

**Date :**

# Transfer Learning and Fine Tuning

## Aim

To implement Transfer Learning and Fine Tuning using Tensorflow.

## Source Code

### Importing Modules

```
import tensorflow as tf
import tensorflow.keras.layers as tfl
import os
```

### Dataset

```
from tensorflow.keras.datasets import cifar10

train_ds, test_ds = cifar10.load_data() # Load the datset
split = int(train_ds[0].shape[0]*.8)
train_ds,val_ds =
(train_ds[0][:split],train_ds[1][:split]),(train_ds[0][split:],train_ds[1][sp
lit:])
data_augumentation = tf.keras.Sequential([
    tfl.Rescaling(1./255),
    tfl.RandomFlip(mode="horizontal"),
    tfl.RandomRotation(factor=.1),
    tfl.RandomContrast(factor=(.2,.9),),
    tfl.RandomZoom(height_factor=(.1,.3))
])

def preprocess(x,y):
    return data_augumentation(x), tf.one_hot(tf.squeeze(y),10)

train_ds =
tf.data.Dataset.from_tensor_slices(train_ds).map(preprocess).batch(64)
val_ds = tf.data.Dataset.from_tensor_slices(val_ds).map(preprocess).batch(64)
test_ds =
tf.data.Dataset.from_tensor_slices(test_ds).map(preprocess).batch(64)
```

```
train_ds
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 32, 32, 3),
dtype=tf.uint8, name=None), TensorSpec(shape=(None, 10), dtype=tf.float32,
name=None))>
```

## Loading weights from a pretrained model

```
base_model = tf.keras.applications.VGG19(include_top=False,weights =
"imagenet",input_shape=(32,32,3))

base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv4 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv4 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808
```

```
 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv4 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

```
for layer in base_model.layers[:-5]:
    layer.trainable=False

base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv4 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808
```

```
 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv4 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv4 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 9,439,232
Non-trainable params: 10,585,152
_________________________________________________________________
```

```
model = tf.keras.Sequential([
    base_model,
    tfl.Flatten(),
    tfl.Dense(256,activation="relu"),
    tfl.Dense(128,activation="relu"),
    tfl.Dense(64,activation="relu"),
    tfl.Dense(10,activation = "softmax")
])

model.summary()
```

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten_17 (Flatten)        (None, 512)               0

 dense_63 (Dense)            (None, 256)               131328

 dense_64 (Dense)            (None, 128)               32896

 dense_65 (Dense)            (None, 64)                8256

 dense_66 (Dense)            (None, 10)                650

=================================================================
```

```
Total params: 20,197,514
Trainable params: 9,612,362
Non-trainable params: 10,585,152
_________________________________________________________________
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=['accuracy']
)

model.fit(
    train_ds,
    validation_data = val_ds,
    epochs=50,
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir="logs")
    ]
)
```

```
Epoch 1/50
625/625 [==============================] - 18s 27ms/step - loss: 1.3345 -
accuracy: 0.5190 - val_loss: 1.1002 - val_accuracy: 0.6241
Epoch 2/50
625/625 [==============================] - 17s 27ms/step - loss: 0.9370 -
accuracy: 0.6873 - val_loss: 0.9605 - val_accuracy: 0.6811
...

...
accuracy: 0.9754 - val_loss: 1.8799 - val_accuracy: 0.7307
Epoch 49/50
625/625 [==============================] - 18s 28ms/step - loss: 0.0811 -
accuracy: 0.9770 - val_loss: 1.9967 - val_accuracy: 0.7363
Epoch 50/50
625/625 [==============================] - 17s 28ms/step - loss: 0.0802 -
accuracy: 0.9775 - val_loss: 2.1469 - val_accuracy: 0.7304

<keras.callbacks.History at 0x7f24f65f2dd0>
```

### Visualizing the training process with tensorboard

```
%load_ext tensorboard
%tensorboard --logdir logs
```

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

### Evaluating the model

```
loss, accuracy = model.evaluate(test_ds)

print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
157/157 [==============================] - 2s 14ms/step - loss: 2.2584 -
accuracy: 0.7221
Loss:  2.2583987712860107
Accuracy:  0.722100019454956
```

### Result

Thus Transfer Learning and Fine Tuning is implemented.s

**Ex No : 7**
**Date :**

# Document Classification Using RNN

## Aim

To implement Document Classification Using RNN.

## Source Code

### Importing the required modules

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import re
import shutil
import string
```

### Dataset

#### Downloading the dataset

```
url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

dataset = tf.keras.utils.get_file(
    "aclImdb_v1",  url,
    untar=True, cache_dir='.',
    cache_subdir=''
)

dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')
```

#### Exploring the dataset directory

```
os.listdir(dataset_dir)
```

```
['test', 'README', 'imdbEr.txt', 'train', 'imdb.vocab']
```

```
train_dir = os.path.join(dataset_dir,"train")
os.listdir(train_dir)
```

```
['neg',
 'unsupBow.feat',
 'urls_neg.txt',
 'unsup',
 'urls_unsup.txt',
 'urls_pos.txt',
```

```
 'pos',
 'labeledBow.feat']
```

**Removing unwanted folders from training folder**

```
shutil.rmtree(os.path.join(train_dir,"unsup"))
```

**Loading the dataset**

```
batch_size = 128
seed = 42

raw_train_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 20000 files for training.
```

```
raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 5000 files for validation.
```

```
raw_test_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/test',
    batch_size=batch_size)
```

```
Found 25000 files belonging to 2 classes.
```

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = raw_train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = raw_val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = raw_test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

## Preprocessing the text

**Standardization Function**

```
def custom_standardization(input_data):
    lowercase = tf.strings.lower(input_data)
    stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ')
    return tf.strings.regex_replace(
        stripped_html,
        f'[{re.escape(string.punctuation)}]',
```

```
        ''
    )
```

**Text Vectorization**

```
max_features = 10000
sequence_length = 250

vectorize_layer = tf.keras.layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)
vectorize_layer.adapt(raw_train_ds.map(lambda x, y: x))
```

## Creating the Model

```
model = tf.keras.Sequential([
    vectorize_layer,
    tf.keras.layers.Embedding(
        input_dim=len(vectorize_layer.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,return_sequences=True))
,
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization_1 (TextV  (None, 250)              0
 ectorization)

 embedding_1 (Embedding)    (None, 250, 64)           640000

 bidirectional_2 (Bidirectio  (None, 250, 128)         66048
 nal)

 bidirectional_3 (Bidirectio  (None, 64)               41216
 nal)
```

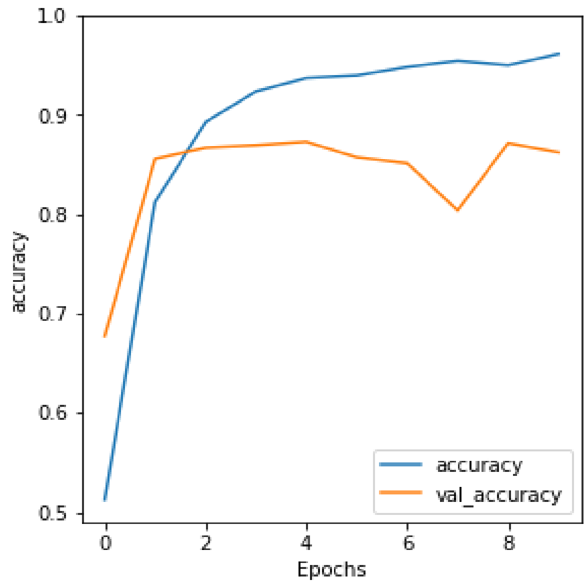
```
 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 751,489
Trainable params: 751,489
Non-trainable params: 0
_________________________________________________________________
```

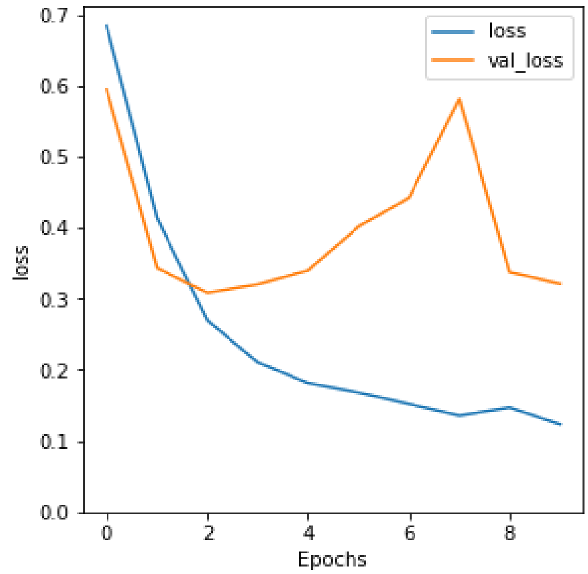
```
history = model.fit(train_ds, epochs=10,
                    validation_data=val_ds,
                    validation_steps=30)
```

```
Epoch 1/10
157/157 [==============================] - 214s 1s/step - loss: 0.6844 -
accuracy: 0.5127 - val_loss: 0.5949 - val_accuracy: 0.6773
Epoch 2/10
157/157 [==============================] - 194s 1s/step - loss: 0.4138 -
accuracy: 0.8120 - val_loss: 0.3435 - val_accuracy: 0.8557
...

...

Epoch 9/10
157/157 [==============================] - 193s 1s/step - loss: 0.1469 -
accuracy: 0.9499 - val_loss: 0.3377 - val_accuracy: 0.8714
Epoch 10/10
157/157 [==============================] - 193s 1s/step - loss: 0.1235 -
accuracy: 0.9611 - val_loss: 0.3213 - val_accuracy: 0.8622
```

```
def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
```

```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.ylim(None, 1)
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
plt.ylim(0,  None)
plt.show()
```




## Evaluating the model

```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

```
196/196 [==============================] - 69s 349ms/step - loss: 0.3627 -
accuracy: 0.8426
Test Loss: 0.36270952224731445
Test Accuracy: 0.8426399827003479
```

## Sample Predictions

```
import numpy as np

samples = np.array([
  'The movie was awesome, wonderful and amazing.',
  "The Movies is the bad and waste of time."
])
predictions = model.predict(samples)

predictions
```

```
array([[ 1.3299981],
       [-1.356227 ]], dtype=float32)
```

## Result

Thus Document Classification Using RNN is implemented.