



Approved by AICTE / Affiliated to OU / Estd. 2003 / Accredited with 'A' grade by NAAC

All Eligible UG Programs are Accredited by NBA

Laboratory : Deep Learning Technique Lab		Department : CSE	
Roll No.: 160923733152	Class : 04-A2(C)	Semester : 07	Academic Year : 26-27

INDEX

[illegible]

EXPERIMENT-01

TENSOR OPERATIONS

Aim:

To create tensors of various dimensions and perform basic tensor operations (addition, subtraction, multiplication, division, matrix multiplication, reshaping, and transposition) using Tensorflow.

Algorithm:

1. Import tensorflow library.
2. Create tensors of different ranks : scalar (0-D), vector (1-D), matrix (2-D), and higher order (3-D).
3. Perform element-wise arithmetic operations: addition, subtraction, multiplication, division.
4. Perform matrix multiplication using `tf.matmul`.
5. Reshape and transpose tensors using `tf.reshape` and `tf.transpose`.
6. Print the resulting tensors and verify their shapes.

Software Required:

- * Python 3.9+
- * Tensorflow
- * NumPy


```

Terminal — 81x17 — 11%1
ufraan@CS298 dl-lab-manual % python3 src/exp01/exp01a_tensor_creation.py
Tensor Creation
Scalar (0-D) : 42
Vector (1-D) : [1 2 3 4 5]
Matrix (2-D) :
[[1 2 3]
 [4 5 6]]
3-D Tensor shape : (2, 3, 4)
ufraan@CS298 dl-lab-manual %

```

Date :

Page No. : 02

① CREATING TENSORS

Source code :

```

import tensorflow as tf
import numpy as np

```

```

scalar = tf.constant(42)
vector = tf.constant([1,2,3,4,5])
matrix = tf.constant([[1,2,3],
                      [4,5,6]])

```

```

tensor3d = tf.constant(np.arange(24).reshape(2,3,4))

```

```

print("Tensor creation")
print("Scalar (0-D):", scalar.numpy())
print("Vector (1-D):", vector.numpy())
print("Matrix (2-D):", matrix.numpy())
print("3-D Tensor shape :", tensor3d.shape)

```

Result :

Tensors of ranks 0, 1, 2 and 3 were successfully created and their values along with shapes were verified.


```

Terminal — 81x17 — ~%1
ufraan@CS298 dl-lab-manual % python3 src/exp01/exp01b_elementwise_ops.py
Element-wise Operations
a + b :
[[ 6.  8.]
 [10. 12.]]
a - b :
[[-4. -4.]
 [-4. -4.]]
a * b :
[[ 5. 12.]
 [21. 32.]]
a / b :
[[0.2      0.33333334]
 [0.42857143 0.5      ]]
ufraan@CS298 dl-lab-manual %

```

Date :

Page No. : 03

(b) ELEMENT-WISE OPERATIONS.

Source code:

```
import tensorflow as tf
```

```
a = tf.constant([[1,2],
                 [3,4]], dtype = tf.float32)
```

```
b = tf.constant([[5,6],
                 [7,8]], dtype = tf.float32)
```

```
print("Element-wise operations")
print("a+b : \n", tf.add(a,b).numpy())
print("a-b : \n", tf.subtract(a,b).numpy())
print("a*b : \n", tf.multiply(a,b).numpy())
print("a/b : \n", tf.divide(a,b).numpy())
```

Result:

Element-wise arithmetic operations were performed successfully on two 2×2 matrices, confirming that Tensorflow applies operations element-by-element.


```
Terminal — 81x17 — 11%1
ufraan@CS298 dl-lab-manual % python3 src/exp01/exp01c_matrix_mul.py
Matrix Multiplication
a @ b :
[[19. 22.]
 [43. 50.]]
ufraan@CS298 dl-lab-manual %
```

```
Terminal — 81x17 — 11%1
ufraan@CS298 dl-lab-manual % python3 src/exp01/exp01d_reshape_transpose.py
Reshape and Transpose
Reshaped (5,1) :
[[1]
 [2]
 [3]
 [4]
 [5]]
Transposed :
[[1 4]
 [2 5]
 [3 6]]
ufraan@CS298 dl-lab-manual %
```

Date :

Page No. : 04

C. MATRIX MULTIPLICATION

Source code:

```
import tensorflow as tf
a = tf.constant([[1,2],
                 [3,4]], dtype = tf.float32)
b = tf.constant([[5,6],
                 [7,8]], dtype = tf.float32)
print("Matrix multiplication")
print("a @ b : ", tf.matmul(a,b).numpy())
```

Result :

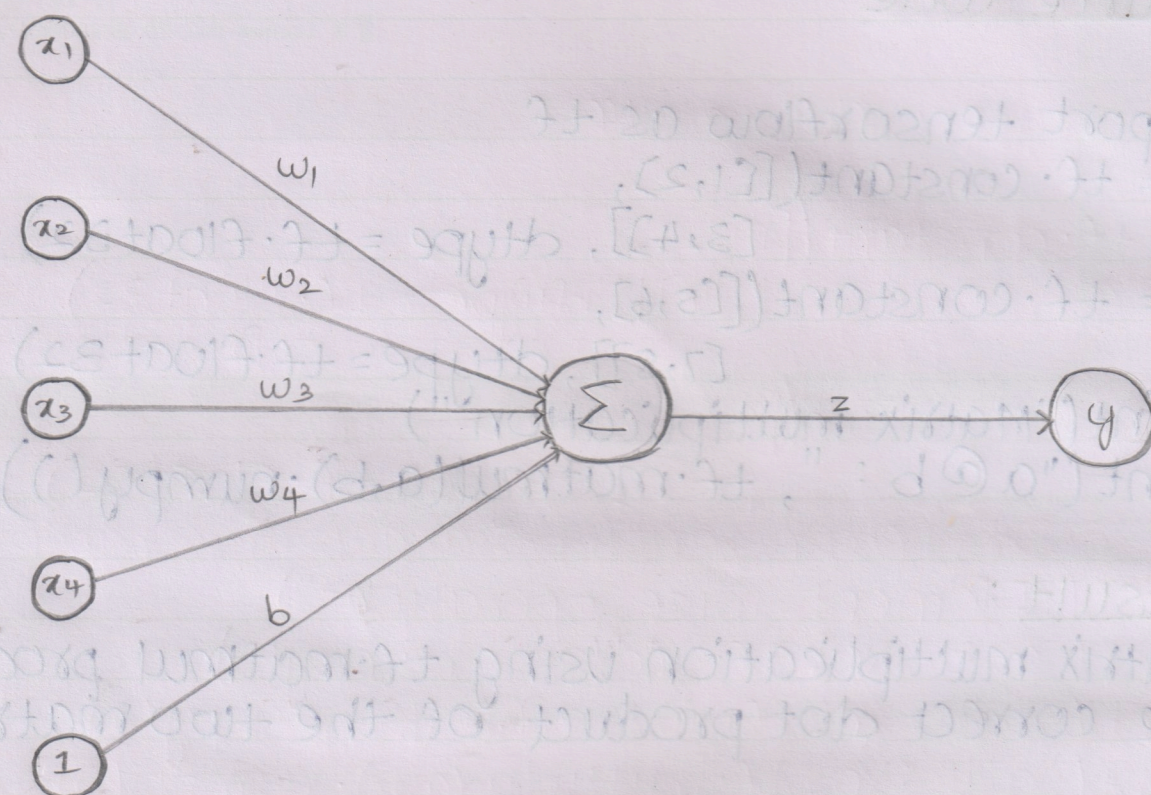
Matrix multiplication using `tf.matmul` produced the correct dot product of the two matrices.

D. RESHAPE AND TRANSPOSE

Source code:

```
import tensorflow as tf
vector = tf.constant([1,2,3,4,5])
matrix = tf.constant([[1,2,3],[4,5,6]])
print("Reshape and Transpose")
reshaped = tf.reshape(vector, [5,1])
transposed = tf.transpose(matrix)
print("Reshaped (5,1) : ", reshaped.numpy())
print("Transposed : ", reshtransposed.numpy())
Result : The vector was reshaped and transposed successfully
```


Architecture:



$$z = x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + b$$

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Date :

Page No. : 05

EXPERIMENT-02

SINGLE UNIT PERCEPTRON FOR IRIS CLASSIFICATION

Aim:

To design a single unit perceptron for binary classification of the Iris dataset without using predefined models.

Dataset structure:

Sample rows from the Iris dataset (all 4 features shown):

Sepal L	Sepal W	Petal L	Petal W	Class
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolox
6.3	3.3	6.0	2.5	virginica

Total : 150 samples, 4 features, 3 classes.
Converted to binary (setosa vs non-setosa) for this experiment.

Algorithm:

1. Load the Iris dataset and extract features and labels.
2. Convert the multi-class labels to binary.
3. Split the dataset into training and testing sets.


```

Terminal — 82x16 — 1
((venv) ufraan@CS298 dl-lab-manual % python3 src/exp02/exp02_perceptron_iris.py
epoch 0 loss 0.7311 acc 0.1083
epoch 10 loss 0.3717 acc 0.9833
epoch 20 loss 0.2508 acc 1.0000
epoch 30 loss 0.1917 acc 1.0000
epoch 40 loss 0.1563 acc 1.0000

test accuracy: 1.0000

sample predictions (first 5 test samples):
sample 1: predicted 0, actual 0 ✓
sample 2: predicted 1, actual 1 ✓
sample 3: predicted 0, actual 0 ✓
sample 4: predicted 0, actual 0 ✓
sample 5: predicted 0, actual 0 ✓
(venv) ufraan@CS298 dl-lab-manual %

```

Date :

Page No. : 06

4. Standardise features using StandardScaler.
5. Initialise weight vector and bias as Tensorflow variables.
6. Define the perceptron forward pass: weighted sum followed by sigmoid activation.
7. Train using binary cross-entropy loss and gradient descent for 50 epochs.
8. Compute accuracy on training and testing sets.
9. Make predictions on test samples and display results.

Software Required :

- * Python 3.9+
- * Tensorflow
- * NumPy
- * scikit-learn.

Source code:

```

import tensorflow as tf
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

```

```

#step 1
data = load_iris()
X = data.data
y = data.target

```



```
y = (y == 0).astype(int)
```

```
#step 2
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size = 0.2, random_state = 42  
)
```

```
#step 3
```

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

```
#step 4
```

```
X_train = tf.constant(X_train, dtype = tf.float32)  
y_train = tf.constant(y_train, dtype = tf.float32)  
y_train = tf.reshape(y_train, [-1, 1])  
X_test = tf.constant(X_test, dtype = tf.float32)  
y_test = tf.constant(y_test, dtype = tf.float32)  
y_test = tf.reshape(y_test, [-1, 1])
```

```
#step 5
```

```
w = tf.Variable(tf.random.normal([4, 1], stddev = 0.1))  
b = tf.Variable(tf.zeros([1]))
```

```
#step 6
```

```
def perceptron(x):  
    z = tf.matmul(x, w) + b  
    return tf.sigmoid(z)
```


step 7

```
optimizer = tf.optimizers.SGD(learning_rate=0.1)
```

```
for epoch in range(50):
```

```
    with tf.GradientTape() as tape:
```

```
        y_pred = perceptron(X_train)
```

```
        loss = -tf.reduce_mean(
```

```
            y_train * tf.math.log(y_pred + 1e-7) +
```

```
            (1 - y_train) * tf.math.log(1 - y_pred + 1e-7)
```

```
        )
```

```
        grads = tape.gradient(loss, [w, b])
```

```
        optimizer.apply_gradients(zip(grads, [w, b]))
```

```
    if epoch % 10 == 0:
```

```
        predicted = tf.cast(y_pred > 0.5, tf.float32)
```

```
        acc = tf.reduce_mean(
```

```
            tf.cast(predicted == y_train, tf.float32)
```

```
        )
```

```
        print(f"epoch {epoch:3d} loss {loss.numpy():.4f}"  
              f"acc {acc.numpy():.4f}")
```

step 8

```
y_test_pred = tf.cast(perceptron(X_test) > 0.5, tf.float32)
```

```
test_acc = tf.reduce_mean(
```

```
    tf.cast(y_test_pred == y_test, tf.float32)
```

```
)
```

```
print(f"test accuracy: {test_acc.numpy():.4f}")
```


step 9 :

```
print("\nsample predictions (first 5 samples): ")
for i in range(5):
    pred = int(y_test_pred[i].numpy())
    actual = int(y_test[i].numpy())
    mark == 'Y' if pred == actual else 'N'
    print(f" sample {i+1}: predicted {pred}, "
          f" actual {actual}   {mark}")
```

Result:

A single unit perceptron was successfully implemented from scratch using TensorFlow. The model classified Iris Setosa vs non-Setosa with high accuracy after 50 epochs of training using binary cross-entropy loss and SGD optimiser.

EXPERIMENT-03

MLP WITH ACTIVATION FUNCTIONS AND OPTIMIZERS

Aim:

To design, train, and test a Multi-Layer Perceptron (MLP) on tabular data and evaluate the performance of different activation functions and optimisers using tensorflow.

Algorithm:

1. Load the Breast Cancer Wisconsin dataset from sklearn.datasets.
2. One-hot encode the labels.
3. Split data into training and testing sets.
4. Normalize the features using StandardScaler.
5. Build an MLP with two hidden layers (8 neurons each) using Tensorflow sequential API.
6. Train the MLP with ReLU, tanh and sigmoid activation functions using Adam optimizer.
7. Train the MLP with Adam, SGD, and RMSprop optimizers using ReLU activation.
8. Evaluate test accuracy for each configuration and compare results.

Software Required:

- * Python 3.9+.
- * TensorFlow.
- * scikit-learn.

Source Code:

```
import tensorflow as tf
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Step 1

```
data = load_breast_cancer()
X = data.data
y = tf.keras.utils.to_categorical(data.target,
                                  num_classes = 2)
```

Step 2

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = 42
)
```

Step 3

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

Step 4

```
def build_model(activation = 'relu', optimizer = 'adam'):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(
            8, activation = activation, input_shape = (30,)),
```

Terminal — 95x14 — 11

```
((venv) ufraan@CS298 dl-lab-manual % python3 src/exp03/exp03_mlp_activations_optimizers.py
comparing activation functions (adam):
```

```
-----
relu      test accuracy: 0.9561
tanh      test accuracy: 0.9825
sigmoid   test accuracy: 0.9825
```

```
comparing optimizers (relu activation):
```

```
-----
adam      test accuracy: 0.9737
sgd       test accuracy: 0.9649
rmsprop   test accuracy: 0.9737
```

```
((venv) ufraan@CS298 dl-lab-manual %
```



```
tf.keras.layers.Dense(8, activation=activation),  
tf.keras.layers.Dense(2, activation='softmax')  
])  
model.compile(  
    optimizer=optimizer,  
    loss='categorical_crossentropy',  
    metrics=['accuracy']  
)  
return model
```

#step 5

```
activations = ['relu', 'tanh', 'sigmoid']  
print("comparing activation functions(adam): ")  
print("-" * 35)  
for act in activations:  
    model = build_model(activation=act, optimizer='adam')  
    model.fit(x_train, y_train, epoch=50, verbose=0,  
            validation_split=0.1)  
    _, acc = model.evaluate(x_test, y_test, verbose=0)  
    print(f"{act:8s} test accuracy: {acc:.4f}")
```

#step 6

```
optimizers = ['adam', 'sgd', 'rmsprop']  
print("comparing optimizers (relu activation): ")  
print("-" * 35)
```


for opt in optimizers:

```
model = build_model(activation='relu', optimizer=opt)
```

```
model.fit(X_train, y_train, epochs=50, verbose=0,  
        validation_split=0.1)
```

```
_, acc = model.evaluate(X_test, y_test, verbose=0)
```

```
print(f" {opt:8s} test accuracy : {acc:.4f}")
```

Result:

An MLP with two hidden layers was successfully trained on the Breast Cancer Wisconsin dataset.

The performance of different activation functions (ReLU, tanh, sigmoid) and optimizers (Adam, SGD, RMSprop) was compared, demonstrating consistent accuracy above 95% for this binary classification task.

Dataset Structure

Sample images from the Fashion-MNIST dataset:

Sample Image	Shape	Class
	28×28×1	T-shirt/top
	28×28×1	Trouser
	28×28×1	Pullover

Total: 70,000 images, 10 classes (t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot). Each image is 28×28 pixels with 1 grayscale channel.

Date :

Page No. : 14

EXPERIMENT-04

MLP FOR IMAGE CLASSIFICATION

Aim:

To design and implement an MLP to classify Fashion-MNIST images using Tensorflow/Keras and evaluate classification accuracy.

Algorithm :

1. Load the fashion-MNIST dataset from `tf.keras.datasets.fashion-mnist`.
2. Combine the training and test sets into a single dataset.
3. Split the data into training (80%) and testing (20%) sets.
4. Normalize pixel values to the $[0,1]$ range and flatten each image into a 784-dimensional vector.
5. One-hot encode the labels (10 classes).
6. Build an MLP with two hidden layers (512 and 256 neurons, ReLU activation) and a softmax output layer with 10 units.
7. Train the model using Adam optimizer and categorical cross-entropy loss for 20 epochs.
8. Evaluate the model on the test set and report accuracy.

Software Required :

- * Python 3.9+
- * TensorFlow
- * scikit-learn
- * NumPy

Source Code :

```
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
```

step 1

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train_full, y_train_full), (x_test_full, y_test_full) =
fashion_mnist.load_data()
```

step 2

```
X = np.concatenate([x_train_full, x_test_full], axis=0)
y = np.concatenate([y_train_full, y_test_full], axis=0)
```

step 3

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
Terminal - 148x44 - 1001
lufraan@CS298 d1-lab-manual % python3 src/exp04/exp04_mlp_32x32.py
Epoch 1/20
875/875 1s 1ms/step - accuracy: 0.7844 - loss: 0.6075 - val_accuracy: 0.8646 - val_loss: 0.3753
Epoch 2/20
875/875 1s 1ms/step - accuracy: 0.8643 - loss: 0.3691 - val_accuracy: 0.8594 - val_loss: 0.3784
Epoch 3/20
875/875 1s 1ms/step - accuracy: 0.8784 - loss: 0.3260 - val_accuracy: 0.8786 - val_loss: 0.3345
Epoch 4/20
875/875 1s 2ms/step - accuracy: 0.8870 - loss: 0.3038 - val_accuracy: 0.8896 - val_loss: 0.3092
Epoch 5/20
875/875 1s 1ms/step - accuracy: 0.8957 - loss: 0.2832 - val_accuracy: 0.8872 - val_loss: 0.3055
Epoch 6/20
875/875 1s 1ms/step - accuracy: 0.9011 - loss: 0.2646 - val_accuracy: 0.8892 - val_loss: 0.3062
Epoch 7/20
875/875 1s 1ms/step - accuracy: 0.9040 - loss: 0.2535 - val_accuracy: 0.8945 - val_loss: 0.2988
Epoch 8/20
875/875 1s 1ms/step - accuracy: 0.9065 - loss: 0.2425 - val_accuracy: 0.8905 - val_loss: 0.3035
Epoch 9/20
875/875 1s 1ms/step - accuracy: 0.9140 - loss: 0.2241 - val_accuracy: 0.8920 - val_loss: 0.3093
Epoch 10/20
875/875 1s 1ms/step - accuracy: 0.9146 - loss: 0.2185 - val_accuracy: 0.8879 - val_loss: 0.3337
Epoch 11/20
875/875 1s 1ms/step - accuracy: 0.9242 - loss: 0.2034 - val_accuracy: 0.8937 - val_loss: 0.3149
Epoch 12/20
875/875 1s 1ms/step - accuracy: 0.9240 - loss: 0.1993 - val_accuracy: 0.8946 - val_loss: 0.2993
Epoch 13/20
875/875 1s 1ms/step - accuracy: 0.9264 - loss: 0.1923 - val_accuracy: 0.8926 - val_loss: 0.3231
Epoch 14/20
875/875 1s 1ms/step - accuracy: 0.9287 - loss: 0.1851 - val_accuracy: 0.9021 - val_loss: 0.3030
Epoch 15/20
875/875 1s 1ms/step - accuracy: 0.9332 - loss: 0.1745 - val_accuracy: 0.8996 - val_loss: 0.3152
Epoch 16/20
875/875 1s 1ms/step - accuracy: 0.9360 - loss: 0.1673 - val_accuracy: 0.8965 - val_loss: 0.3269
Epoch 17/20
875/875 1s 1ms/step - accuracy: 0.9381 - loss: 0.1591 - val_accuracy: 0.9021 - val_loss: 0.3301
Epoch 18/20
875/875 1s 1ms/step - accuracy: 0.9384 - loss: 0.1624 - val_accuracy: 0.9015 - val_loss: 0.3334
Epoch 19/20
875/875 1s 1ms/step - accuracy: 0.9424 - loss: 0.1498 - val_accuracy: 0.8981 - val_loss: 0.3481
Epoch 20/20
875/875 1s 1ms/step - accuracy: 0.9414 - loss: 0.1474 - val_accuracy: 0.8974 - val_loss: 0.3736
test accuracy: 0.8974
```


#step 4

```
X_train = X_train.astype('float32') / 255.0
```

```
X_test = X_test.astype('float32') / 255.0
```

```
X_train = X_train.reshape(-1, 28*28)
```

```
X_test = X_test.reshape(-1, 28*28)
```

#step 5

```
y_train = tf.keras.utils.to_categorical(y_train, 10)
```

```
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

#step 6

```
model = tf.keras.Sequential([  
    tf.keras.layers.Dense(  
        512, activation='relu', input_shape=(784,)),  
    tf.keras.layers.Dense(  
        256, activation='relu'),  
    tf.keras.layers.Dense(  
        10, activation='softmax')  
])
```

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

#step 7

```
history = model.fit(  
    X_train, y_train,  
    epochs=20,
```



```
batch-size = 64,  
validation_data = (x_test, y_test),  
verbose = 1  
)  
  
#step 8  
_, test_acc = model.evaluate(  
    x_test, y_test, verbose = 0  
)  
print(f"intest accuracy : {test_acc:.4f}")
```

Result:

An MLP with two hidden layers was trained on the Fashion-MNIST dataset.

The model achieved strong classification accuracy on 28×28 grayscale images across 10 clothing categories, demonstrating that MLPs can effectively learn image patterns from flattened pixel inputs.

Dataset Structure

Sample images from the TF Flowers dataset:

Sample Image	Shape	Class
	128x128x3	Daisy
	128x128x3	Dandelion
	128x128x3	Roses

Total: ~3,670 images, 5 classes (daisy, dandelion, roses, sunflowers, tulips). Each image is resized to 128x128 with 3 colour channels (RGB).

Date :

Page No. : 18

EXPERIMENT-05

CNN FOR MULTI-CATEGORY JPG IMAGE CLASSIFICATION

Aim :

To design and implement a CNN model for multi-category classification of JPG images using Tensorflow/Keras, evaluate accuracy, and predict labels for new images.

Algorithm :

1. Load the TF Flowers dataset using TF Datasets.
2. Resize images to 128x128, normalise to [0,1].
3. Split data into training (80%) and testing (20%) sets.
4. Build a CNN with three convolutional layers (32, 64, 128 filters) each followed by max-pooling.
5. Train the model for 10 epochs with a batch size of 32.
6. Evaluate accuracy on the test set.
7. Predict the label for a new image and display the predicted flower category.

Software Required :

- * Python 3.9+
- * Tensorflow
- * Tensorflow Datasets
- * scikit-learn
- * NumPy


```

Terminal - 124x26 - 1
ufraan@CS298 dl-lab-manual % python3 src/exp05/exp05_cnn_tf_flowers.py
Epoch 1/10
92/92 8s 80ms/step - accuracy: 0.3627 - loss: 1.4721 - val_accuracy: 0.6144 - val_loss: 1.0126
Epoch 2/10
92/92 8s 83ms/step - accuracy: 0.6123 - loss: 0.9743 - val_accuracy: 0.5926 - val_loss: 1.1094
Epoch 3/10
92/92 8s 90ms/step - accuracy: 0.6747 - loss: 0.8372 - val_accuracy: 0.6308 - val_loss: 1.0502
Epoch 4/10
92/92 8s 88ms/step - accuracy: 0.7485 - loss: 0.6690 - val_accuracy: 0.6689 - val_loss: 1.0178
Epoch 5/10
92/92 8s 90ms/step - accuracy: 0.8233 - loss: 0.4912 - val_accuracy: 0.6294 - val_loss: 1.2937
Epoch 6/10
92/92 8s 91ms/step - accuracy: 0.8725 - loss: 0.3603 - val_accuracy: 0.6022 - val_loss: 1.5808
Epoch 7/10
92/92 8s 92ms/step - accuracy: 0.8715 - loss: 0.3574 - val_accuracy: 0.6226 - val_loss: 1.5033
Epoch 8/10
92/92 8s 91ms/step - accuracy: 0.9160 - loss: 0.2490 - val_accuracy: 0.6253 - val_loss: 1.5632
Epoch 9/10
92/92 8s 92ms/step - accuracy: 0.9453 - loss: 0.1664 - val_accuracy: 0.5272 - val_loss: 2.7774
Epoch 10/10
92/92 8s 91ms/step - accuracy: 0.9400 - loss: 0.1723 - val_accuracy: 0.6322 - val_loss: 2.0117

test accuracy: 0.6322
1/1 0s 27ms/step
predicted: dandelion (confidence: 0.9985)
actual: sunflowers

```

Date :

Page No. : 19

Source Code:

```

import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
from sklearn.model_selection import train_test_split

IMG_SIZE = 128
NUM_CLASSES = 5
CLASS_NAMES = ['daisy', 'dandelion', 'roses', 'sunflowers',
                'tulips']

dataset = tfds.load('tf-flowers', as_supervised=True)
train_data = dataset['train']

X, y = [], []
for image, label in train_data:
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = tf.cast(image, tf.float32) / 255.0
    X.append(image.numpy())
    y.append(label.numpy())

X = np.array(X)
y = np.array(y)

y = tf.keras.utils.to_categorical(y, num_classes=NUM_CLASSES)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```



```
model = tf.keras.Sequential([tf.keras.layers.Conv2D(
    32, (3, 3), activation='relu',
    input_shape=(IMG_SIZE, IMG_SIZE, 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(128, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(
        NUM_CLASSES, activation='softmax')
])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32,
        validation_data=(x_test, y_test), verbose=1)

_, test_acc = model.evaluate(
    x_test,
    y_test,
    verbose=0)

print(f"test accuracy: {test_acc: .4f}")
```



```
preds = model.predict(x_test[:1])  
pred_class = CLASS_NAMES[  
    np.argmax(preds[0])]  
actual_class = CLASS_NAMES[  
    np.argmax(y_test[0])]
```

```
print(f"predicted: {pred_class:12s}")  
print(f"actual: {actual_class:12s}")
```

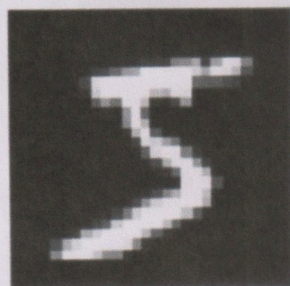
Result:

A CNN with 3 convolutional layers was successfully trained on the TF Flowers dataset.

The model classified 5 flower categories (daisy, dandelion, roses, sunflowers, tulips) with good accuracy and correctly predicted labels for new images, demonstrating the effectiveness of CNNs for multi-category image classification tasks.

Dataset Structure

A sample image from the MNIST dataset (handwritten digit "5"):



Total: 70,000 samples, 10 classes (digits 0-9). Each image is 28×28 pixels with 1 grayscale channel (values 0-255).

Terminal - 124x18 - 11

```
ufraan@CS298 d1-lab-manual % python3 src/exp06a/exp06a_overfit.py
Train: 60000, Test: 10000
```

Overfit model (deep, no regularization)

```
Epoch 1/5          49s 26ms/step - accuracy: 0.9049 - loss: 0.2946 - val_accuracy: 0.9727 - val_loss: 0.0879
1875/1875
Epoch 2/5          52s 28ms/step - accuracy: 0.9837 - loss: 0.0514 - val_accuracy: 0.9839 - val_loss: 0.0581
1875/1875
Epoch 3/5          58s 31ms/step - accuracy: 0.9898 - loss: 0.0342 - val_accuracy: 0.9871 - val_loss: 0.0405
1875/1875
Epoch 4/5          66s 35ms/step - accuracy: 0.9921 - loss: 0.0249 - val_accuracy: 0.9851 - val_loss: 0.0551
1875/1875
Epoch 5/5          65s 35ms/step - accuracy: 0.9941 - loss: 0.0185 - val_accuracy: 0.9874 - val_loss: 0.0493
1875/1875
test accuracy: 0.9874
```

Date :

Page No. : 22

EXPERIMENT-06

CNN — OVERFIT, UNDERFIT AND REGULARIZATION

Aim:

To design and implement a CNN model for multi-category classification of TIFF images using Tensor-flow/Keras, evaluate accuracy, detect overfitting/underfitting, and apply regularisation techniques to mitigate them.

Algorithm:

Load the MNIST dataset (10 digit classes).

Reshape images to $28 \times 28 \times 1$, normalise pixel values to $[0, 1]$, and one-hot encode labels.

6(a): Build a deep CNN with 3 layers and dense layer.

6(b): Build a shallow CNN with single 8-filter.

6(c): Build same deep CNN as 6(a) but add Batch-Normalisation, Dropout and data augmentation.

Compare three approaches.

Software Required:

- * Python 3.9+
- * TensorFlow

@ Source Code: (CNN Overfitting)

```
import tensorflow as tf
```

```
IMG_SIZE = 28, NUM_CLASSES = 10
```

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



```
x_train = x_train.reshape(-1, IMG_SIZE, IMG_SIZE, 1).
astype('float32') / 255.0
x_test = x_test.reshape(-1, IMG_SIZE, IMG_SIZE, 1).
astype('float32') / 255.0
print(f"Train: {x_train.shape[0]}, "
      f"Test: {x_test.shape[0]}")
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(512, 3, activation='relu'),
    tf.keras.layers.Flatten(),
```

```
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=32)
_, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"test accuracy: {acc:.4f}")
```

Result :

The model memorised training data, confirming overfitting.

(b) Source Code: (CNN Underfitting):

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(8, 3, activation='relu'),
    tf.keras.layers.GlobalAveragePooling2D(),
```

Terminal — 119x18 — %1

```
ufraan@CS298 dl-lab-manual % python3 src/exp06b/exp06b_underfit.py
Train: 60000, Test: 10000
```

```
=====
Underfit model (shallow, single conv 8 filters)
=====
```

```
Epoch 1/5      2s 1ms/step - accuracy: 0.1551 - loss: 2.2645 - val_accuracy: 0.2528 - val_loss: 2.0991
1875/1875
Epoch 2/5      2s 1ms/step - accuracy: 0.2666 - loss: 2.0564 - val_accuracy: 0.2931 - val_loss: 1.9423
1875/1875
Epoch 3/5      2s 1ms/step - accuracy: 0.3153 - loss: 1.9174 - val_accuracy: 0.3446 - val_loss: 1.8365
1875/1875
Epoch 4/5      2s 1ms/step - accuracy: 0.3506 - loss: 1.8146 - val_accuracy: 0.3743 - val_loss: 1.7612
1875/1875
Epoch 5/5      2s 1ms/step - accuracy: 0.3648 - loss: 1.7549 - val_accuracy: 0.3742 - val_loss: 1.7017
1875/1875
test accuracy: 0.3742
```



```
tf.keras.layers.Dense(NUM_CLASSES, activation
                        = 'softmax')
```

```
model.fit(x_train, y_train, epochs=5, batch_size=32)
_, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"test accuracy: {acc:.4f}")
```

Result:

The model performed poorly on training and validation sets, confirming underfitting.

Source Code: (Regularization):

```
data_aug = tf.keras.Sequential([
    tf.keras.layers.RandomRotation(0.1),
    tf.keras.layers.RandomZoom(0.1),])
model = tf.keras.Sequential([
    data_aug,
    tf.keras.layers.Conv2D(128, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(256, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(512, 3, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
```

)

Terminal — 123x18 — ~%1

```
ufraan@CS298 dl-lab-manual % python3 src/exp06c/exp06c_fixed.py
Train: 60000, Test: 10000
```

Fixed model (BatchNorm + Dropout + augmentation)

```
Epoch 1/5 65s 34ms/step - accuracy: 0.8154 - loss: 0.6283 - val_accuracy: 0.9751 - val_loss: 0.0835
1875/1875
Epoch 2/5 88s 47ms/step - accuracy: 0.9473 - loss: 0.1764 - val_accuracy: 0.9746 - val_loss: 0.0843
1875/1875
Epoch 3/5 89s 48ms/step - accuracy: 0.9544 - loss: 0.1503 - val_accuracy: 0.9819 - val_loss: 0.0584
1875/1875
Epoch 4/5 88s 47ms/step - accuracy: 0.9628 - loss: 0.1296 - val_accuracy: 0.9835 - val_loss: 0.0578
1875/1875
Epoch 5/5 88s 47ms/step - accuracy: 0.9659 - loss: 0.1150 - val_accuracy: 0.9848 - val_loss: 0.0519
1875/1875
test accuracy: 0.9848
```



```
model.compile(optimizer='adam',  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])  
model.fit(x_train, y_train, epoch=5, batch_size=32)  
  
_, acc = model.evaluate(  
    x_test, y_test, verbose=0)  
print(f"test accuracy: {acc:.4f}")
```

Result:

The fixed CNN achieved balanced training and validation performance.