

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.
5. Reshape and transpose tensors.
6. Print resulting tensors and verify shapes.

Software Required:

Python 3.9+
Tensorflow
NumPy

(a) 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("Tensorflow 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.

Output :

Tensor creation :

Scalar (0-D) : 42

Vector (1-D) : [1 2 3 4 5]

Matrix (2-D) :

$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

$\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$

3-D Tensor shape : (2, 3, 4)

(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.

Output:

Element-wise operations

$a+b$:

$\begin{bmatrix} 6 & 8 \end{bmatrix}$

$\begin{bmatrix} 10 & 12 \end{bmatrix}$

$a-b$:

$\begin{bmatrix} -4 & -4 \end{bmatrix}$

$\begin{bmatrix} -4 & -4 \end{bmatrix}$

$a \times b$:

$\begin{bmatrix} 5 & 12 \end{bmatrix}$

$\begin{bmatrix} 21 & 32 \end{bmatrix}$

a/b :

$\begin{bmatrix} 0.2 & 0.333 \end{bmatrix}$

$\begin{bmatrix} 0.428 & 0.5 \end{bmatrix}$

© 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.

Output:

Matrix multiplication

a @ b :

[[19. 22.]

[43. 50.]]

(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:", transposed.numpy())
```


Result:

The vector was reshaped and transposed successfully.

Output:

Reshape and Transpose

Reshaped (5,1):

[1]

[2]

[3]

[4]

[5]

Transposed :

[1 4]

[2 5]

[3 6]

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.

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.
4. Standardise features using StandardScaler.
5. Initialise weight vector and bias.
6. Define the perceptron forward pass: weighted sum with 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.

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.
```

```
data = load_iris()
```

```
X = data.data
```

```
y = data.target
```

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

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

```
scaler = StandardScaler()
```

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

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

```
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])
```


EXPERIMENT-02

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

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

```
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}"
```

```
              f"loss {loss.numpy():.4f}"
```

```
              f"acc {acc.numpy():.4f}")
```



```
y_test_pred = tf.cast(perceptron(x_test) > 0.5,  
float32)  
test_acc = tf.reduce_mean(  
    tf.cast(y_test_pred == y_test, tf.float32)  
)
```

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

```
print("sample predictions: ")
```

```
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.

Output:

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 %
```


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. Normalise the features using Standard-Scales.
5. Build an MLP with two hidden layers (8 neurons each).
6. Train the MLP with ReLU, tanh, and sigmoid activation functions using Adam.
7. Train the MLP with Adam, SGD, and RMSprop optimizers using ReLU.
8. Evaluate test accuracy for each configuration and compare results.

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

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

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

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


EXPERIMENT-04

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

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

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

```
for opt in optimizers:  
    model = build_model(activation='relu', optimizer=opt)  
    model.fit(X_train, y_train, epochs=50, verbose=0)  
    _, 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 and optimizers was completed, demonstrating consistent accuracy above 95% for this binary classification task.

Output:

```
Terminal - 95x14 - %1
((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 %
```


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) 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.

Source Code:

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

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

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

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

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)

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



```
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']
)
```

```
history = model.fit(
    x_train, y_train, epochs=20, batch_size=64,
    validation_data=(x_test, y_test), verbose=0)
)
```

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

```
print(f"\ntest 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.

Output:

Terminal — 148x44 — 1

```
ufraan@CS298 dl-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
```


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 datasets using TF datasets.
2. Resize images to 128, 128, 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.

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, 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'),
    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.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, 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 with good accuracy and correctly predicted labels for new images, demonstrating the effectiveness of CNNs for multi-category image classification tasks.

Output:

Terminal - 124x26 - 100%

ufraan@CS298 d1-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
```


EXPERIMENT-06

CNN - Overfit, Underfit and Regularization

Aim:

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

Algorithm:

1. Load the MNIST dataset (10 digit classes).
2. Reshape images to $28 \times 28 \times 1$, normalize pixel values to $[0, 1]$, and one-hot encode labels.
3. 6(a): Build a deep CNN with 3 layers and a dense layer.
4. 6(b): Build a shallow CNN with single 8-filter.
5. 6(c): Build same deep CNNs as 6(a) but add Batch Normalization, Dropout and data augmentation.
6. Compare three approaches.

Software Required:

Python 3.9+

Tensorflow

a) CNN Overfitting

Source Code:

```
import tensorflow as tf
IMG_SIZE, NUM_CLASSES = 28, 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.

Output:

Terminal — 124x18 — 1

```
ufraan@CS298 dl-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
```


(b) CNN UNDERFITTINGSource Code:

```

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

Output:

```

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

```


(C) REGULARIZATION

Source Code:

```
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(),
])
```

```
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.

Output:

Terminal - 119x18 - ㄟ%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									
1875/1875	2s	1ms/step	- accuracy: 0.1551	- loss: 2.2645	- val_accuracy: 0.2528	- val_loss: 2.0991			
Epoch 2/5									
1875/1875	2s	1ms/step	- accuracy: 0.2666	- loss: 2.0564	- val_accuracy: 0.2931	- val_loss: 1.9423			
Epoch 3/5									
1875/1875	2s	1ms/step	- accuracy: 0.3153	- loss: 1.9174	- val_accuracy: 0.3446	- val_loss: 1.8365			
Epoch 4/5									
1875/1875	2s	1ms/step	- accuracy: 0.3506	- loss: 1.8146	- val_accuracy: 0.3743	- val_loss: 1.7612			
Epoch 5/5									
1875/1875	2s	1ms/step	- accuracy: 0.3648	- loss: 1.7549	- val_accuracy: 0.3742	- val_loss: 1.7017			
test accuracy: 0.3742									