WEEKLY ASSESSMENT WEEK 6&7

Navani P

Activity 2:

AIM: Using a deep learning framework of your choice (TensorFlow, PyTorch, etc.), implement a CNN to classify images from the CIFAR-10 dataset. Ensure your network includes convolutional layers, pooling layers, and fully connected layers. Evaluate the performance of your model and discuss any improvements you could make.

REQUIREMENTS:

Laptop/Computer

VS Code, Jupiter Notebook,Google Colab.

Procedure:

```
#Question 2

# Import necessary libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```python
# Load housing dataset
df = pd.read_csv('housing_prices.csv')

# Preprocess the data
# One-hot encode 'Location' column
df = pd.get_dummies(df, columns=['Location'], drop_first=True)

# Separate features and target
X = df.drop('Price', axis=1)
y = df['Price']

# Normalize numerical inputs
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```python
# Build the FNN model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1)  # No activation function for output layer (linear activation)
])



# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mse'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)


# Evaluate the model
test_loss, test_mse = model.evaluate(X_test, y_test)

print(f'Test Mean Squared Error: {test_mse}')
```

```python
# Predicting on new data (example)
# Assume new_data is already prepared and scaled as in the previous example

new_data = np.array([[3, 2, 1500, 10, 0, 1]])  # Example input (3 bedrooms, 2 bathrooms, 1500 sqft, Urban, 10 years old)
new_data_scaled = scaler.transform(new_data)

# Make predictions using the trained model
prediction = model.predict(new_data_scaled)
print(f'Predicted Price: ${prediction[0][0]:.2f}')
```

OUTPUT

+ Code + Text

```
Epoch 33/50
1/1 [==============================] - 0s 35ms/step - loss: 108098650112.0000 - mse: 108098650112.0000 - val_loss: 228047421440.0000 - val_mse: 2280474214
Epoch 34/50
1/1 [==============================] - 0s 34ms/step - loss: 108098584576.0000 - mse: 108098584576.0000 - val_loss: 228047323136.0000 - val_mse: 228047323136.0000
Epoch 35/50
1/1 [==============================] - 0s 32ms/step - loss: 108098519040.0000 - mse: 108098519040.0000 - val_loss: 228047192064.0000 - val_mse: 228047192064.0000
Epoch 36/50
1/1 [==============================] - 0s 33ms/step - loss: 108098461696.0000 - mse: 108098461696.0000 - val_loss: 228047060992.0000 - val_mse: 228047060992.0000
Epoch 37/50
1/1 [==============================] - 0s 39ms/step - loss: 108098379776.0000 - mse: 108098379776.0000 - val_loss: 228046880768.0000 - val_mse: 228046880768.0000
Epoch 38/50
1/1 [==============================] - 0s 32ms/step - loss: 108098314240.0000 - mse: 108098314240.0000 - val_loss: 228046733312.0000 - val_mse: 228046733312.0000
Epoch 39/50
1/1 [==============================] - 0s 34ms/step - loss: 108098232320.0000 - mse: 108098232320.0000 - val_loss: 228046602240.0000 - val_mse: 228046602240.0000
Epoch 40/50
1/1 [==============================] - 0s 33ms/step - loss: 108098150400.0000 - mse: 108098150400.0000 - val_loss: 228046438400.0000 - val_mse: 228046438400.0000
Epoch 41/50
1/1 [==============================] - 0s 35ms/step - loss: 108098068480.0000 - mse: 108098068480.0000 - val_loss: 228046307328.0000 - val_mse: 228046307328.0000
Epoch 42/50
1/1 [==============================] - 0s 42ms/step - loss: 108097970176.0000 - mse: 108097970176.0000 - val_loss: 228046127104.0000 - val_mse: 228046127104.0000
Epoch 43/50
1/1 [==============================] - 0s 42ms/step - loss: 108097888256.0000 - mse: 108097888256.0000 - val_loss: 228045946880.0000 - val_mse: 228045946880.0000
Epoch 44/50
1/1 [==============================] - 0s 59ms/step - loss: 108097789952.0000 - mse: 108097789952.0000 - val_loss: 228045766656.0000 - val_mse: 228045766656.0000
Epoch 45/50
1/1 [==============================] - 0s 41ms/step - loss: 108097699840.0000 - mse: 108097699840.0000 - val_loss: 228045570048.0000 - val_mse: 228045570048.0000
Epoch 46/50
1/1 [==============================] - 0s 47ms/step - loss: 108097593344.0000 - mse: 108097593344.0000 - val_loss: 228045357056.0000 - val_mse: 228045357056.0000
Epoch 47/50
1/1 [==============================] - 0s 60ms/step - loss: 108097495040.0000 - mse: 108097495040.0000 - val_loss: 228045193216.0000 - val_mse: 228045193216.0000
Epoch 48/50
1/1 [==============================] - 0s 43ms/step - loss: 108097396736.0000 - mse: 108097396736.0000 - val_loss: 228044980224.0000 - val_mse: 228044980224.0000
Epoch 49/50
1/1 [==============================] - 0s 33ms/step - loss: 108097282048.0000 - mse: 108097282048.0000 - val_loss: 228044750848.0000 - val_mse: 228044750848.0000
Epoch 50/50
1/1 [==============================] - 0s 34ms/step - loss: 108097167360.0000 - mse: 108097167360.0000 - val_loss: 228044537856.0000 - val_mse: 228044537856.0000
1/1 [==============================] - 0s 25ms/step - loss: 96196714496.0000 - mse: 96196714496.0000
Test Mean Squared Error: 96196714496.0
1/1 [==============================] - 0s 80ms/step
Predicted Price: $2.87
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

ACTIVITY 1:

AIM: Construct a feedforward neural network to predict housing prices based on the provided

dataset. Include input normalization, hidden layers with appropriate activation

functions, and an output layer. Train the network using backpropagation and evaluate its

performance using Mean Squared Error (MSE).

Bedrooms,Bathrooms,SquareFootage,Location,Age,Price

3,2,1500,Urban,10,300000

4,3,2000,Suburban,5,400000

2,1,800,Rural,20,150000

3,2,1600,Urban,12,310000

4,3,2200,Suburban,8,420000

2,1,900,Rural,25,160000

5,4,3000,Urban,3,600000

3,2,1400,Suburban,15,290000

3,2,1300,Rural,30,180000

4,3,2500,Urban,7,500000

You can copy this data into a CSV file named housing_prices.csv

REQUIREMENTS:

Laptop/Computer

VS Code, Jupiter Notebook,Google Colab.

PROCEDURE:

```
[1]  #Question 1

     # Import necessary libraries
     import tensorflow as tf
     from tensorflow.keras import datasets, layers, models
     import matplotlib.pyplot as plt
```

```
# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

1m

```python
# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

11m

```python
# Train the model
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'Test accuracy: {test_acc}')

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()
```

OUTPUT

+ Code + Text

```
1563/1563 [==============================] - 62s 39ms/step - loss: 0.8588 - accuracy: 0.6996 - val_loss: 0.9233 - val_accuracy: 0.6826
Epoch 6/10
1563/1563 [==============================] - 65s 41ms/step - loss: 0.7992 - accuracy: 0.7194 - val_loss: 0.8938 - val_accuracy: 0.6956
Epoch 7/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.7558 - accuracy: 0.7345 - val_loss: 0.9030 - val_accuracy: 0.6846
Epoch 8/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.7115 - accuracy: 0.7531 - val_loss: 0.8599 - val_accuracy: 0.7107
Epoch 9/10
1563/1563 [==============================] - 63s 40ms/step - loss: 0.6724 - accuracy: 0.7664 - val_loss: 0.9259 - val_accuracy: 0.6953
Epoch 10/10
1563/1563 [==============================] - 62s 40ms/step - loss: 0.6381 - accuracy: 0.7754 - val_loss: 0.8827 - val_accuracy: 0.7077
313/313 - 4s - loss: 0.8827 - accuracy: 0.7077 - 4s/epoch - 11ms/step
Test accuracy: 0.7077000141143799
```