7/12/2024

DOCUMENTATION

WEEKLY ASSESSMENT

SHUHAIB
NSTI - CALICUT

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:

- Pc
- jupyterNotebook / vs code

PROCEDURE/CODE:

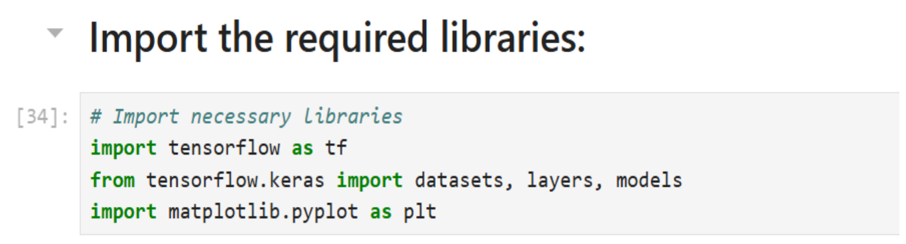
To implement a Convolutional Neural Network (CNN) for classifying images from the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 different classes.

Requirements:

- TensorFlow
- Keras (part of TensorFlow)
- CIFAR-10 dataset

Steps:-

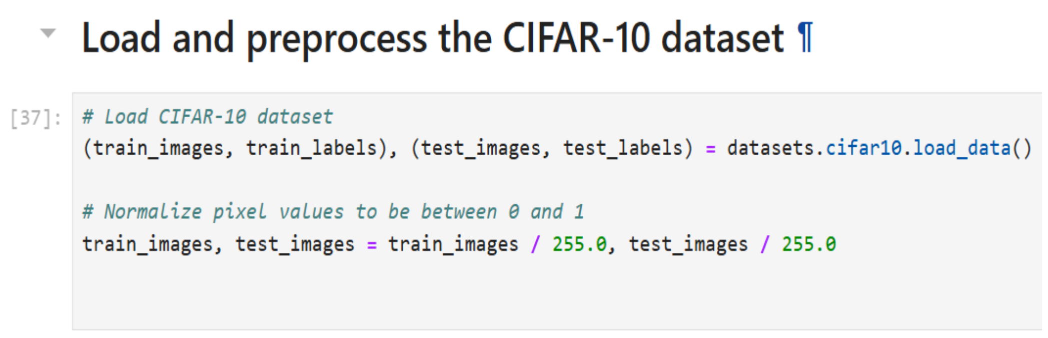
1.



Import the required libraries:

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

2.

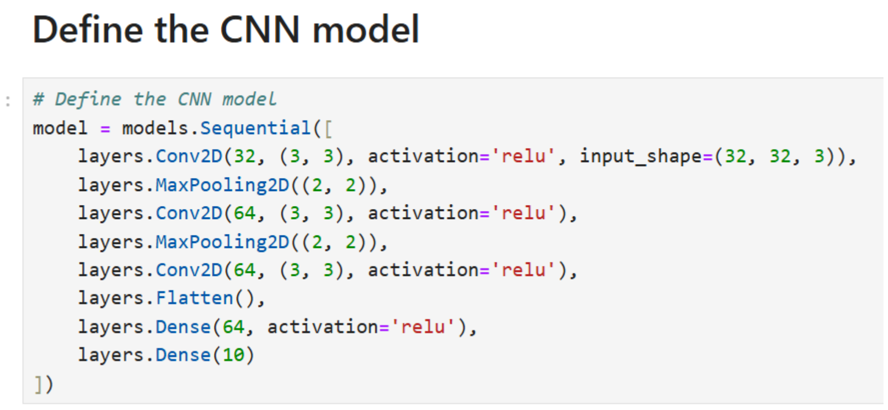


Load and preprocess the CIFAR-10 dataset ¶

```
[37]: # 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
```

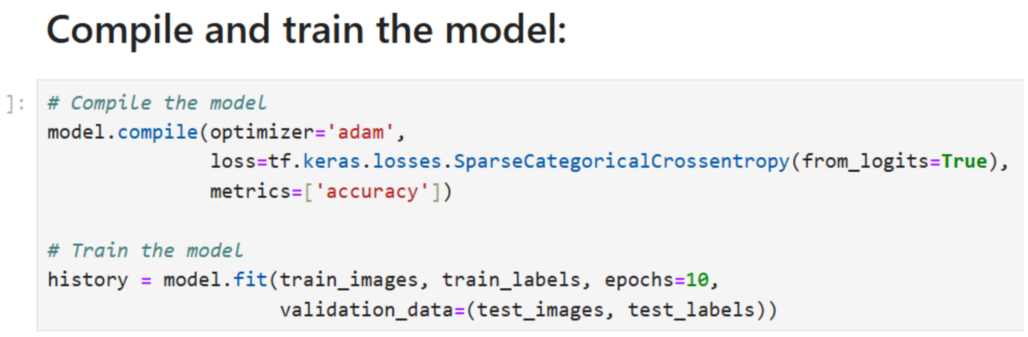
3.



Define the CNN model

```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)
])
```

4.

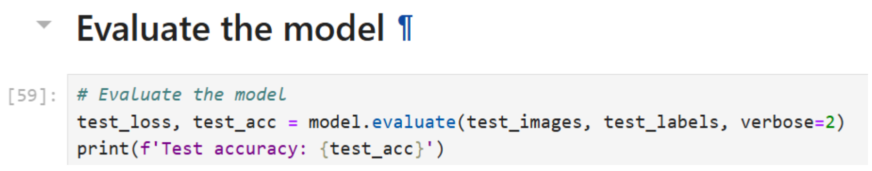


Compile and train the model:

```python
# 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))
```

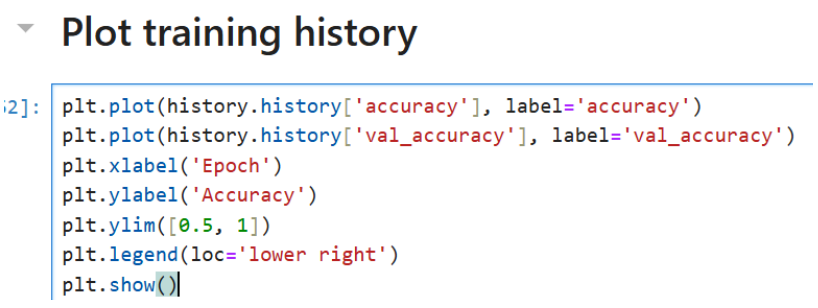
5.



Evaluate the model ¶

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

6.



Plot training history

```python
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

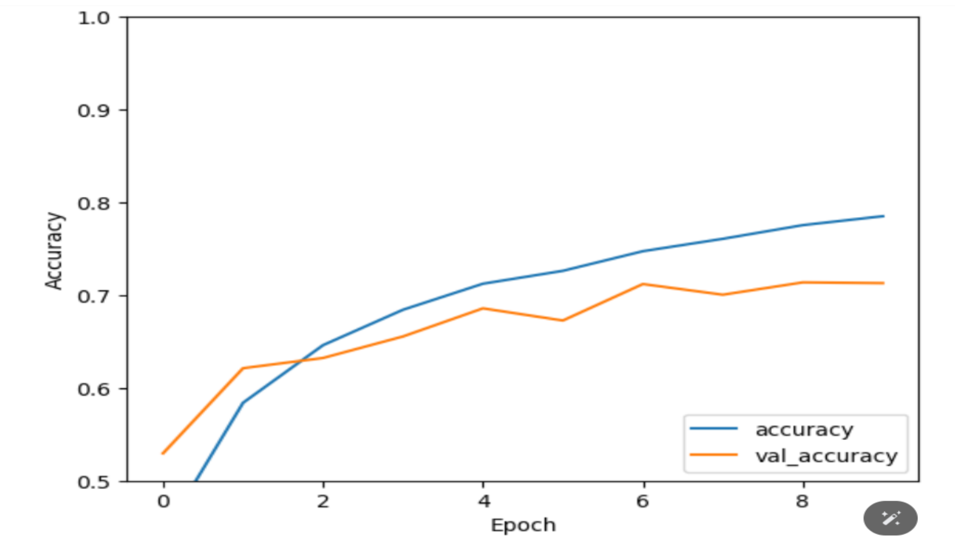
Compile and train the model:

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 32s 19ms/step - accuracy: 0.3445 - loss: 1.7589 - val_accuracy: 0.5295 - val_loss: 1.3499
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 27s 17ms/step - accuracy: 0.5670 - loss: 1.2132 - val_accuracy: 0.6212 - val_loss: 1.0723
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 32s 20ms/step - accuracy: 0.6376 - loss: 1.0278 - val_accuracy: 0.6322 - val_loss: 1.0547
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 32s 20ms/step - accuracy: 0.6801 - loss: 0.9073 - val_accuracy: 0.6554 - val_loss: 1.0033
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 20ms/step - accuracy: 0.7153 - loss: 0.8166 - val_accuracy: 0.6857 - val_loss: 0.9077
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 32s 21ms/step - accuracy: 0.7287 - loss: 0.7699 - val_accuracy: 0.6726 - val_loss: 0.9465
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.7495 - loss: 0.7118 - val_accuracy: 0.7119 - val_loss: 0.8371
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.7670 - loss: 0.6649 - val_accuracy: 0.7004 - val_loss: 0.8891
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 20ms/step - accuracy: 0.7787 - loss: 0.6249 - val_accuracy: 0.7137 - val_loss: 0.8525
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 20ms/step - accuracy: 0.7932 - loss: 0.5882 - val_accuracy: 0.7129 - val_loss: 0.8801
```

Evaluate the model

```
313/313 - 2s - 7ms/step - accuracy: 0.7129 - loss: 0.8801
Test accuracy: 0.7128999829292297
```

Plot training history



Result:-

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:

- Pc
- jupyterNotebook/ vs code

PROCEDURE/CODE:

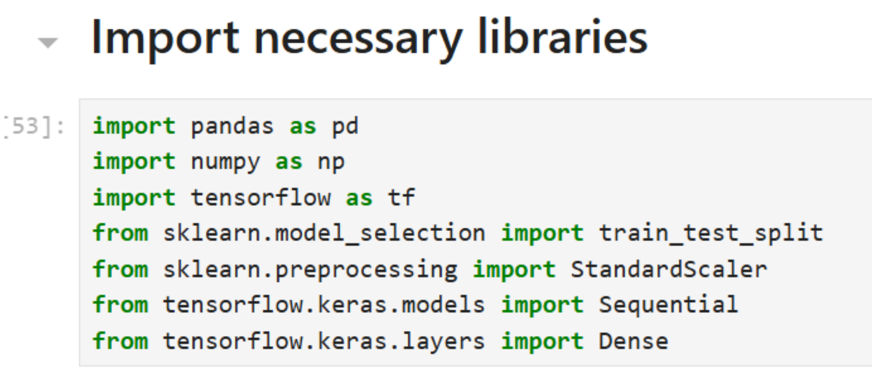
To implement a feedforward neural network to predict housing prices based on various features such as the number of bedrooms, bathrooms, square footage, location, and age of the house.

Requirements:

- TensorFlow
- Keras (part of TensorFlow)
- Pandas
- Scikit-learn

Steps:-

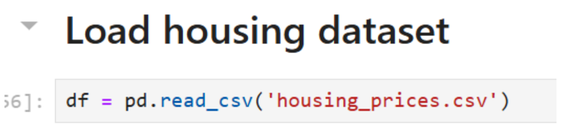
1.



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
```

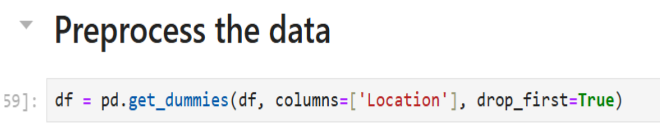
2.



Load housing dataset

```
df = pd.read_csv('housing_prices.csv')
```

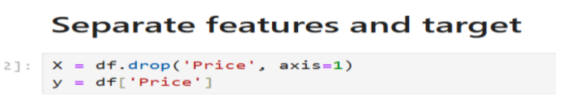
3.



Preprocess the data

```
df = pd.get_dummies(df, columns=['Location'], drop_first=True)
```

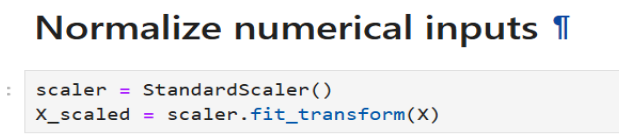
4.



Separate features and target

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

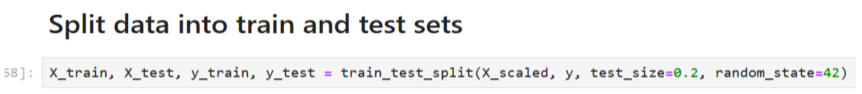
5.



Normalize numerical inputs ¶

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

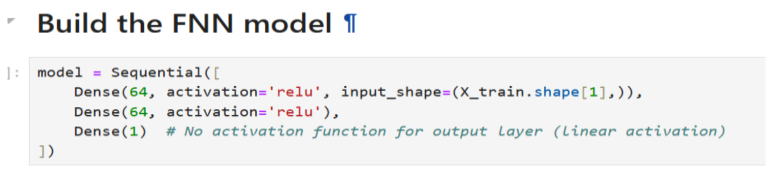
6.



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)
```

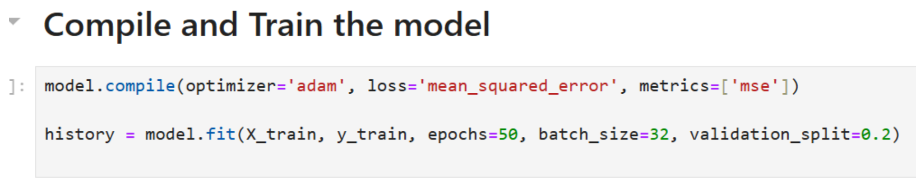
7.



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)
])
```

8.

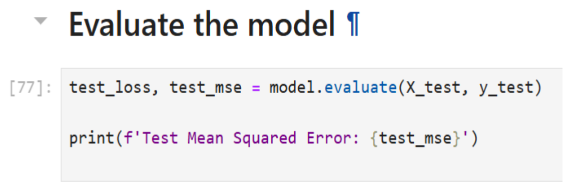


Compile and Train the model

```python
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mse'])

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

9.

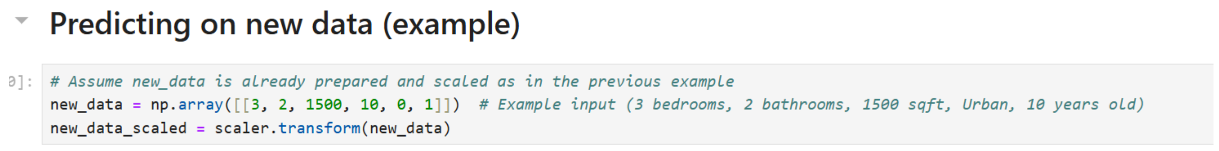


Evaluate the model ¶

```python
test_loss, test_mse = model.evaluate(X_test, y_test)

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

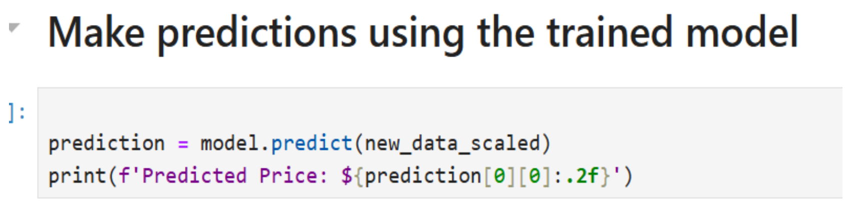
10.



Predicting on new data (example)

```python
# 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)
```

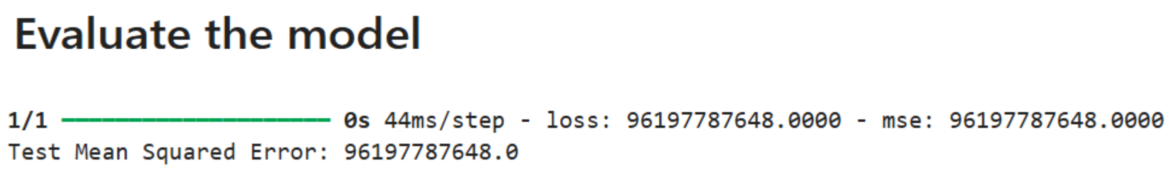
11.



Make predictions using the trained model

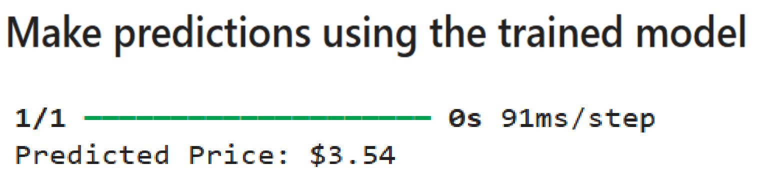
```python
prediction = model.predict(new_data_scaled)
print(f'Predicted Price: ${prediction[0][0]:.2f}')
```

OUTPUT



Evaluate the model

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - loss: 96197787648.0000 - mse: 96197787648.0000
Test Mean Squared Error: 96197787648.0
```



Make predictions using the trained model

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
Predicted Price: $3.54
```