7/12/2024

Weekly Assessment

MUHAMMED HIDASH

NSTI CALICUT

AIM:

Implement a Convolutional Neural Network (CNN) using a deep learning framework to classify images from the CIFAR-10 dataset.

Requirements:

- Pc/Laptop
- VS Code
- Chrome
- Python installed with TensorFlow, matplotlib

Learning Outcome:

- Understand the architecture of CNNs, including convolutional layers, pooling layers

Procedure:

Step-1: Data

```
# Import necessary libraries
```

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

Step-2: Model Architecture:

- Design a CNN architecture consisting of convolutional layers (with activation functions like ReLU), pooling layers (such as max pooling), and fully connected layers.

```python
# 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

# 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))
```

Step-4: Evaluation:

- Evaluate the trained model on the test set to measure its accuracy and other relevant metrics.
- Discuss any observed patterns in model performance and potential areas for improvement (e.g., regularization, data augmentation).

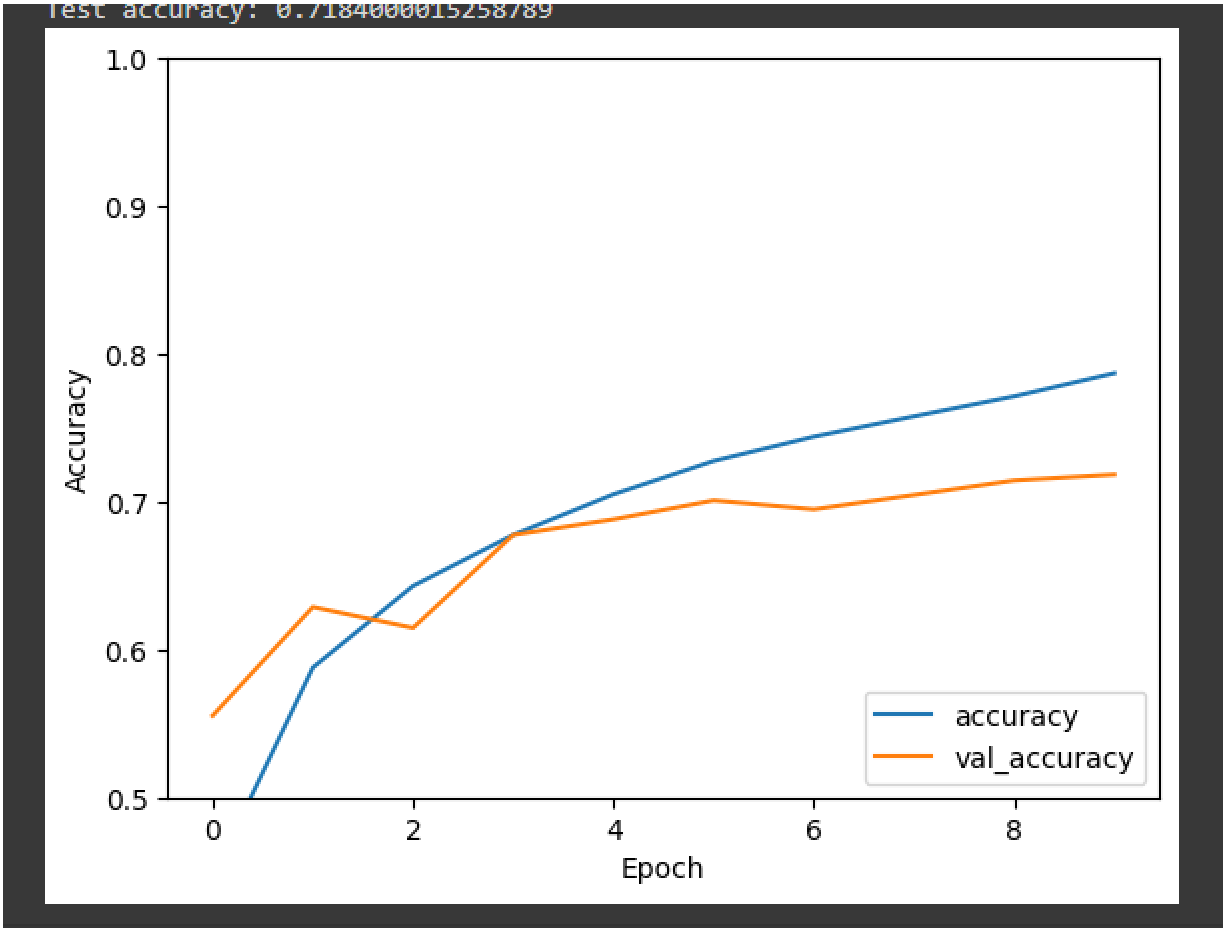
```python
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'Test accuracy: {test_acc}')

# 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()
```

Step-5: Improvements:

- Strategies to improve model performance, such as adjusting network architecture (adding more layers, changing filter sizes), optimizing hyperparameters, or applying advanced techniques like transfer learning.

Outputs:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 11s 0us/step
Epoch 1/10
1563/1563 [==============================] - 63s 40ms/step - loss: 1.5148 - accuracy: 0.4459 - val_loss: 1.2322 - val_accuracy: 0.5553
Epoch 2/10
1563/1563 [==============================] - 61s 39ms/step - loss: 1.1608 - accuracy: 0.5879 - val_loss: 1.0515 - val_accuracy: 0.6288
Epoch 3/10
1563/1563 [==============================] - 61s 39ms/step - loss: 1.0053 - accuracy: 0.6431 - val_loss: 1.1144 - val_accuracy: 0.6147
Epoch 4/10
1563/1563 [==============================] - 62s 39ms/step - loss: 0.9130 - accuracy: 0.6776 - val_loss: 0.9279 - val_accuracy: 0.6776
Epoch 5/10
1563/1563 [==============================] - 60s 39ms/step - loss: 0.8322 - accuracy: 0.7050 - val_loss: 0.8969 - val_accuracy: 0.6881
Epoch 6/10
1563/1563 [==============================] - 60s 39ms/step - loss: 0.7738 - accuracy: 0.7276 - val_loss: 0.8635 - val_accuracy: 0.7009
Epoch 7/10
1563/1563 [==============================] - 59s 38ms/step - loss: 0.7286 - accuracy: 0.7440 - val_loss: 0.8950 - val_accuracy: 0.6949
Epoch 8/10
1563/1563 [==============================] - 62s 39ms/step - loss: 0.6859 - accuracy: 0.7578 - val_loss: 0.8546 - val_accuracy: 0.7046
Epoch 9/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.6449 - accuracy: 0.7713 - val_loss: 0.8394 - val_accuracy: 0.7144
Epoch 10/10
1563/1563 [==============================] - 61s 39ms/step - loss: 0.6075 - accuracy: 0.7869 - val_loss: 0.8512 - val_accuracy: 0.7184
313/313 - 3s - loss: 0.8512 - accuracy: 0.7184 - 3s/epoch - 9ms/step
Test accuracy: 0.7184000015258789
```



AIM:

Construct a feedforward neural network to predict housing prices based on provided dataset.

Requirements:

- Pc/Laptop
- VS Code

Learning Outcome:

- Implement a feedforward neural network (FFNN) using a deep learning framework for regression tasks.
-

Procedure:

Step-1: Data:

- Created a CSV file named `housing_prices.csv` with columns: Bedrooms, Bathrooms, Square Footage,

```
# 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
```

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

Step-2: Model:

- Design a feedforward neural network with an appropriate number of hidden layers and neurons per layer.

```
(module) pd
# Loa        aset
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)
```

Step-3:

- Configure the training process: select optimizer (e.g., Adam), loss function (e.g., MSE), and possibly adjust learning rate.
- Split the dataset into training and validation sets.
- Train the model using backpropagation, iterating over the dataset for multiple epochs.

```
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)
])
Python
C:\Users\USER\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first l
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

# 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)
Python
```

Step-4: Evaluation:

- Evaluate the trained model using Mean Squared Error (MSE) on the validation set to assess its predictive performance.
- Interpret the MSE value in the context of the housing price predictions.

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

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - loss: 96195518464.0000 - mse: 96195518464.0000
Test Mean Squared Error: 96195518464.0
```

+ Code + Markdown

```
# 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}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
C:\Users\USER\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
Predicted Price: $3.44
```

Step-5: Improvements:

- Improvements sexperimenting with different network architectures (adding more layers, adjusting layer sizes), tuning hyperparameters (learning rate, batch size), or incorporating regularization techniques (e.g., dropout) to enhance model performance.

Outputs:


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - loss: 96195518464.0000 - mse: 96195518464.0000
Test Mean Squared Error: 96195518464.0
```

```
# 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}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
C:\Users\USER\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
Predicted Price: $3.44
```