EXPERIMENT 5

Aim: Build a Convolution Neural Network for MNIST Hand written Digit Classification.

Theory:

This code implements a Convolutional Neural Network (CNN) for classifying handwritten digits using the MNIST dataset. The MNIST dataset consists of 28×28 grayscale images of digits (0-9) and their corresponding labels. CNNs are particularly well-suited for image recognition tasks due to their ability to extract spatial features.

Data Loading and Preprocessing

- The dataset is stored in a compressed archive and extracted using the zipfile module.
- The load_mnist_images and load_mnist_labels functions read the binary data and reshape the images into (28,28,1) while normalizing pixel values to the range [0,1].
- Training and test images/labels are loaded from the extracted files.

CNN Architecture

The CNN model is built using keras.Sequential with the following layers:

- **Conv2D (32 filters, 3×3 kernel, ReLU activation)**: Extracts local features from input images.
- **MaxPooling2D (2×2 pool size)**: Reduces spatial dimensions, enhancing computational efficiency.
- **Conv2D (64 filters, 3×3 kernel, ReLU activation)**: Captures more complex patterns.
- **MaxPooling2D (2×2 pool size)**: Further reduces dimensions.
- **Flatten Layer**: Converts the 2D feature maps into a 1D vector.
- **Dense (128 neurons, ReLU activation)**: Fully connected layer for learning abstract representations.
- **Dense (10 neurons, Softmax activation)**: Outputs probabilities for 10 digit classes (0-9).

Compilation and Training

- **Loss Function**: sparse_categorical_crossentropy is used since labels are integer values.
- **Optimizer**: adam, an adaptive learning rate optimizer, enhances convergence speed.
- **Metrics**: Model performance is evaluated using accuracy.
- The model is trained for 5 epochs with a batch size of 64.

Model Evaluation and Prediction

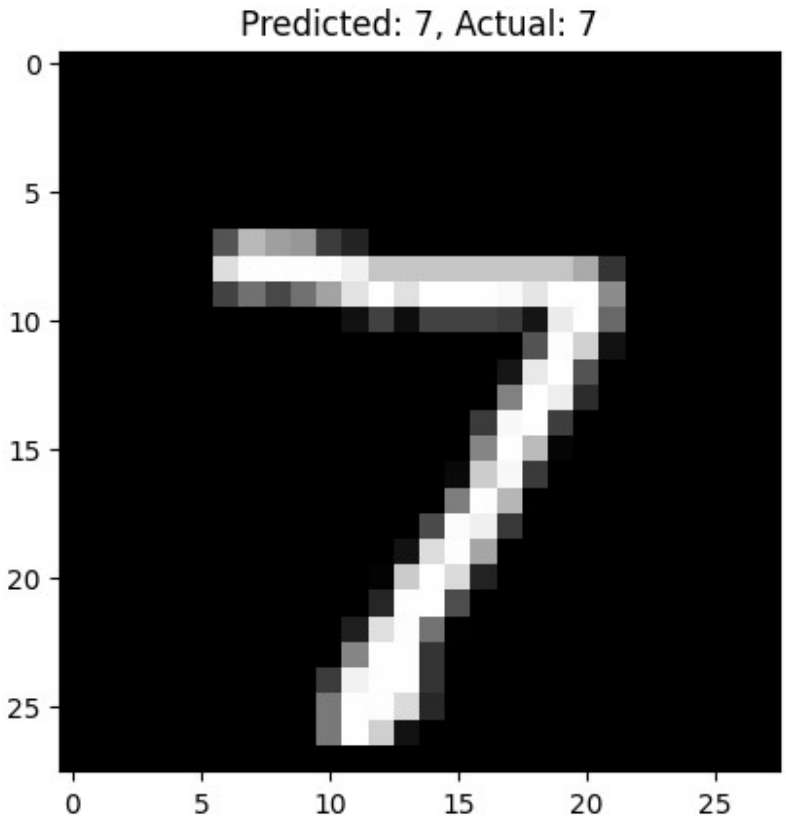
- The model is tested on unseen MNIST images, and test accuracy is printed.
- A sample test image is displayed using Matplotlib, along with the predicted and actual labels.

Code:

```
import  zipfile
import zipfile
import os
import numpy as np
import struct
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
zip_path = "/MNIST/archive.zip"
extract_path = "/MNIST"
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
zip_ref.extractall(extract_path)
def load_mnist_images(filename):
with open(filename, 'rb') as f:
_, num, rows, cols = struct.unpack(">IIII", f.read(16))
return np.fromfile(f, dtype=np.uint8).reshape(num, rows, cols, 1) / 255.0
def load_mnist_labels(filename):
with open(filename, 'rb') as f:
_, num = struct.unpack(">II", f.read(8))
return np.fromfile(f, dtype=np.uint8)
train_images = load_mnist_images(os.path.join(extract_path, "train-images.idx3-
ubyte"))
train_labels = load_mnist_labels(os.path.join(extract_path, "train-labels.idx1-
ubyte"))
test_images = load_mnist_images(os.path.join(extract_path, "t10k-images.idx3-
ubyte"))
test_labels = load_mnist_labels(os.path.join(extract_path, "t10k-labels.idx1-
ubyte"))
model = keras.Sequential([layers.Conv2D(32, kernel_size=(3,3), activation='relu',
input_shape=(28,28,1)),
layers.MaxPooling2D(pool_size=(2,2)),
layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
layers.MaxPooling2D(pool_size=(2,2)),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, validation_data=(test_images, test_labels),
epochs=5, batch_size=64)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print("Test Accuracy:", test_acc)
predictions = model.predict(test_images)
import matplotlib.pyplot as plt
index = 0
plt.imshow(test_images[index].reshape(28, 28), cmap='gray')
plt.title(f"Predicted: {np.argmax(predictions[index])}, Actual:
{test_labels[index]}")
plt.show()
```

Output:

```
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 52s 55ms/step - accuracy: 0.8887 - loss: 0.3688 - val_accuracy: 0.9810 - val_loss: 0.0580
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9843 - loss: 0.0529 - val_accuracy: 0.9872 - val_loss: 0.0381
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 79s 46ms/step - accuracy: 0.9897 - loss: 0.0323 - val_accuracy: 0.9887 - val_loss: 0.0320
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 82s 46ms/step - accuracy: 0.9922 - loss: 0.0244 - val_accuracy: 0.9897 - val_loss: 0.0320
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 44s 46ms/step - accuracy: 0.9941 - loss: 0.0182 - val_accuracy: 0.9903 - val_loss: 0.0332
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9868 - loss: 0.0423
Test Accuracy: 0.9902999997138977
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
```



Learning Outcomes:

EXPERIMENT 6

Aim: Design a neural network for classifying movie reviews (Binary Classification) using IMDB dataset.

Theory:

This code is designed for sentiment analysis using the IMDB movie reviews dataset. It builds and trains a neural network to classify movie reviews as either positive or negative. The model utilizes a deep learning approach with embedding layers for natural language processing (NLP).

Importing Required Libraries

- tensorflow and keras: Used for building and training the neural network.
- imdb dataset: Contains preprocessed movie reviews labeled as positive (1) or negative (0).
- pad_sequences: Ensures uniform input length by padding or truncating reviews.
- numpy: For numerical operations.

Loading and Preprocessing Data

- The imdb.load_data(num_words=10000) function loads the IMDB dataset, keeping.
- pad_sequences is applied to ensure all input sequences are of equal length (max_len=200). This step helps in maintaining consistency in input size for training the model.

Building the Neural Network Model

The model follows a sequential architecture with the following layers:

Embedding Layer: Converts words (represented as integers) into dense vectors of fixed size (128-dimensional). It helps in capturing semantic meaning.

Flatten Layer: Converts the multi-dimensional output from the embedding layer into a one-dimensional vector.

Dense Layers:

- Three fully connected (Dense) layers with ReLU activation function, which helps in learning complex patterns.
- An output layer with a single neuron and sigmoid activation function for binary classification.

Compilation and Training

- adam optimizer: Used for efficient optimization and faster convergence.

- binary_crossentropy loss function: Suitable for binary classification problems.
- accuracy metric: Evaluates model performance.
- The model is trained for 5 epochs with a batch size of 64 and validated on test data.

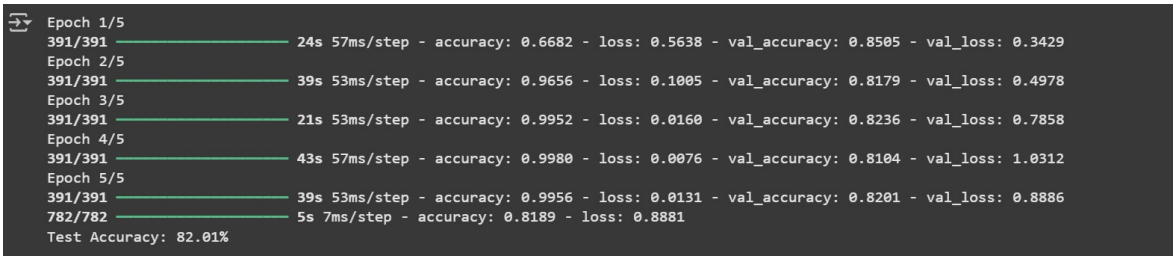
Model Evaluation

The model is tested on unseen test data.The final test accuracy is printed, indicating how well the model performs on movie review classification.

Code:

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad_sequences

import numpy as np

num_words = 10000

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)

max_len = 200

x_train = pad_sequences(x_train, maxlen=max_len, padding='post',

truncating='post') x_test = pad_sequences(x_test, maxlen=max_len, padding='post',

truncating='post') model = keras.Sequential([

    keras.layers.Embedding(input_dim=num_words,                    output_dim=128,
input_length=max_len),

    keras.layers.Flatten(),

    keras.layers.Dense(128, activation='relu'),

    keras.layers.Dense(64, activation='relu'),

    keras.layers.Dense(32, activation='relu'),

    keras.layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary_crossentropy',

metrics=['accuracy']) epochs = 5

batch_size = 64

model.fit(x_train,       y_train,       epochs=epochs,       batch_size=batch_size,
validation_data=(x_test, y_test))

loss, accuracy = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Output:



Learning Outcomes: