USN NUMBER: 1RVU22CSE128
NAME: Reethu RG Thota

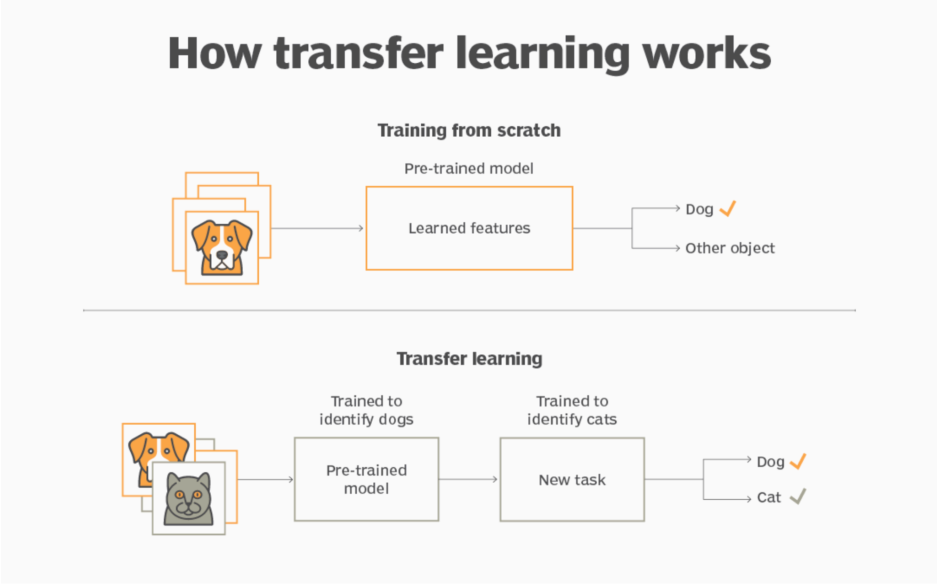
Ex No: 4 **Date:** 28/08/24	**CNN for Handwritten Digit Classification**

Objective

To implement and train a Convolutional Neural Network (CNN) to classify handwritten digits from the MNIST dataset and evaluate its performance.

Descriptions

In this experiment, we aim to implement a ***Convolutional Neural Network (CNN)*** for handwritten digit classification using the ***MNIST dataset***. The model will be built from scratch and trained to recognize digits from 0 to 9 based on images of handwritten characters. This experiment focuses on designing and training a CNN that can effectively learn and extract features from grayscale images, enabling accurate classification.



Convolutional Neural Networks are a specialized type of neural network designed to process data with a grid-like topology, such as images. They automatically and adaptively learn spatial hierarchies of features through backpropagation, making them highly effective for image recognition tasks. In this experiment, we use the MNIST dataset, which consists of ***60,000 training images and 10,000 test images of handwritten digits (0-9)***. The CNN model is designed with convolutional layers, pooling layers, and fully connected layers that work together to identify and classify the digits in the MNIST dataset. The model's performance is evaluated based on its accuracy in predicting the correct digit for each test image.

USN NUMBER: 1RVU22CSE128
NAME: Reethu RG Thota

Model

The CNN model is composed of the following layers:

1. **Convolutional Layer 1:** 32 filters with a kernel size of 3×3, using ReLU activation. This layer extracts low-level features from the input images.
2. **Convolutional Layer 2:** 64 filters with a kernel size of 3×3, using ReLU activation. This layer captures more complex patterns by combining the features learned in the first layer.
3. **MaxPooling Layer:** Pool size of 2×2 to reduce the spatial dimensions of the feature maps, retaining the most important features.
4. **Dropout Layer 1:** A dropout rate of 0.25 to prevent overfitting by randomly setting a fraction of input units to 0 during training.
5. **Flatten Layer:** Converts the 2D feature maps into a 1D vector to be used by the fully connected layers.
6. **Dense Layer 1:** 256 units with ReLU activation. This layer combines the features learned to make a classification decision.
7. **Dropout Layer 2:** A dropout rate of 0.5 to further prevent overfitting.
8. **Output Dense Layer:** 10 units with softmax activation, representing the 10 digit classes.

The model is compiled with the Adadelta optimizer, categorical cross-entropy as the loss function, and accuracy as the metric.

Building the parts of algorithm

The steps involved in building the CNN are:

1. **Data Preparation:**
    - **Load the MNIST dataset:**
      The MNIST dataset is a benchmark dataset of handwritten digits used for image classification tasks.
    - **Normalize the pixel values to the range [0, 1]:**
      Normalization scales pixel values to a range between 0 and 1, improving model performance and stability.
    - **Reshape the data to match the input shape required by the CNN:**
      Reshaping ensures that the input dimensions match what the CNN expects, which is necessary for proper processing.
2. **Model Architecture:**

```
model = Sequential()
model.add(Conv2D(32,    kernel_size=(3,    3),    activation='relu',
input_shape=(28, 28, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

3. **Compilation:**

   Compiles the model with categorical cross entropy loss for multi-class classification and the Adadelta optimizer for efficient training. Accuracy is used as the evaluation metric.

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

4. **Training the Model:**
   Trains the model on the training data for 10 epochs with a batch size of 128, and validates on the test data after each epoch to monitor performance.

   ```
   model.fit(x_train, y_train, batch_size=128, epochs=10,
   validation_data=(x_test, y_test))
   ```

5. **Evaluation:**
   Evaluates the model on the test data to obtain the final loss and accuracy metrics.

   ```
   score = model.evaluate(x_test, y_test, verbose=0)
   print('Test loss:', score[0])
   print('Test accuracy:', score[1])
   ```

Result

Train accuracy: 73.45%

Test accuracy: 83.32%

GitHub Link

https://github.com/reethuthota/Deep_Learning/tree/main/Lab4