Practical-3 img classification

---

1. Importing Required Libraries
```python
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
```

- **Libraries**:
  - **Numpy & Pandas**: Used for data manipulation.
  - **Matplotlib**: For data visualization.
  - **Scikit-Learn**: Provides functions for splitting data and calculating accuracy.
  - **TensorFlow & Keras**: For building and training the CNN model.

---

2. Loading the Dataset
```python
(images, labels), _ = mnist.load_data()
X = images
y = labels
```

- **MNIST Data**: Loads the MNIST dataset.
- **Variables**: Stores image data in `X` and labels in `y` for easy reference.

---

3. Loading and Preprocessing the Image Data
```python
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

- **Train/Test Split**: Loads MNIST dataset, splitting it into training and testing sets (`X_train`, `X_test` for images and `y_train`, `y_test` for labels).

---

4. Checking Data Shape
```python
print(X_train.shape)
```

- **Shape Check**: Displays the shape of the training dataset, which is `(60000, 28, 28)` (60,000 images, each 28x28 pixels).

---

5. Checking Pixel Intensity Range
```python
X_train[0].min(), X_train[0].max()
```

- **Range Check**: Confirms that pixel intensities in the images range from 0 to 255.

---

6. Normalizing the Data
```python
X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
```

- **Normalization**: Scales pixel values to a range of 0 to 1 by dividing by 255.0, which improves model performance.

---

7. Visualizing Some Digits
```python
def plot_digit(image, digit, plt, i):
    plt.subplot(4, 5, i + 1)
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.title(f"Digit: {digit}")
    plt.xticks([])
    plt.yticks([])
plt.figure(figsize=(16, 10))
for i in range(20):
    plot_digit(X_train[i], y_train[i], plt, i)
plt.show()
```

```

- **Plot Function**: Defines `plot_digit` to visualize images with labels.
- **Display 20 Images**: Plots 20 random images with their respective labels for inspection.

---

### 8. Reshaping Data for CNN
```python
X_train = X_train.reshape((X_train.shape + (1,)))
X_test = X_test.reshape((X_test.shape + (1,)))
```

- **Reshaping**: Adds an extra dimension to represent a single color channel (grayscale), resulting in `(28, 28, 1)` shape per image.

---

9. Checking Labels
```python
y_train[0:20]
```

- **Label Check**: Displays the first 20 labels in `y_train` to verify they correspond to the displayed digits.

---

10. Defining the Model Architecture
```python
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(100, activation="relu"),
    Dense(10, activation="softmax")
])
```

- **Sequential Model**: Creates a model with layers stacked sequentially.
  - **Conv2D Layer**: 32 filters, each 3x3, activated by ReLU to capture features.
  - **MaxPooling2D Layer**: Reduces spatial size to minimize computation and prevent overfitting.
  - **Flatten Layer**: Converts the 2D data into a 1D vector.
  - **Dense Layers**:
```

   - **Hidden Layer**: 100 neurons with ReLU activation.
   - **Output Layer**: 10 neurons with Softmax activation (for digit probabilities from 0-9).

---

11. Compiling the Model
```python
optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
    optimizer=optimizer,
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
```

- **Optimizer**: Uses Stochastic Gradient Descent (SGD) with momentum (0.9) to optimize training.
- **Loss Function**: `sparse_categorical_crossentropy` for multi-class classification.
- **Metrics**: Tracks accuracy during training.

---

12. Displaying Model Summary
```python
model.summary()
```

- **Model Summary**: Outputs model architecture, showing layers, shapes, and parameter counts.

---

13. Splitting Data for Training and Validation
```python
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, 
random_state=42)
```

- **Train/Validation Split**: Divides 80% of the data for training and 20% for validation to monitor model performance.

---

14. Training the Model
```python
```

Model_log = model.fit(
    X_train,
    y_train,
    epochs=10,
    batch_size=15,
    verbose=1,
    validation_data=(X_val, y_val)
)
```

- **Model Training**: Runs for 10 epochs with a batch size of 15, validating on `X_val` and `y_val` at each epoch.
- **Verbose**: Displays training progress.

---

### 15. Plotting Model Predictions on Random Test Images
```python
plt.figure(figsize=(16, 10))
for i in range(20):
    image = random.choice(X_test).squeeze()
    digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
    plot_digit(image, digit, plt, i)
plt.show()
```

- **Random Test Image Selection**: Picks 20 random images from `X_test` to visualize predictions.
- **Prediction**: `model.predict` returns class probabilities; `np.argmax` finds the class with the highest probability.

---

16. Making Predictions on Test Data
```python
predictions = np.argmax(model.predict(X_test), axis=-1)
accuracy_score(y_test, predictions)
```

- **Predictions**: Predicts labels for all test images in `X_test`.
- **Accuracy Calculation**: Compares predictions to `y_test` and calculates accuracy.

---

17. Visualizing a Specific Image
```python

n = random.randint(0, 9999)
plt.imshow(X_test[n])
plt.show()
```

- **Random Image Display**: Selects a random image from `X_test` and displays it.

---

18. Predicting the Selected Image's Digit
```python
predicted_value = model.predict(X_test)
print("Handwritten number in the image is = %d" %
np.argmax(predicted_value[n]))
```

- **Prediction for Single Image**: Predicts the class of the chosen image and displays the result.

---

19. Evaluating Model Performance
```python
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

- **Model Evaluation**: Calculates final loss and accuracy on `X_test` and `y_test`.
- **Outputs**: Displays loss and accuracy values as the final performance metrics.

——

Here are key questions for Assignment 3, which focuses on building an image classification model:

Important Questions and Answers

1. **What is an Image Classification problem?**
   - Image classification involves categorizing images into predefined classes based on visual content using deep learning models.

2. **Why use Deep Learning for Image Classification?**
```

   - Deep learning models, especially Convolutional Neural Networks (CNNs), excel in recognizing complex patterns in image data, offering high accuracy and efficiency.

3. **What is a Convolutional Neural Network (CNN)?**
   - CNN is a deep learning model designed to process data with a grid-like topology, such as images. It uses convolution layers to automatically and adaptively learn spatial hierarchies in data.

4. **Explain the Convolution operation in CNN.**
   - The convolution operation involves sliding a filter (kernel) across the image to detect specific features, creating a feature map that highlights important parts of the image.

5. **What is a Convolution Kernel?**
   - A kernel is a small matrix applied to an image through convolution, extracting features like edges and textures.

6. **Describe the types of convolution layers used in CNN.**
   - **1D Convolution**: Processes sequential data (e.g., text).
   - **2D Convolution**: Processes image data.
   - **3D Convolution**: Processes volumetric data (e.g., medical imaging).

7. **How does feature extraction occur in convolution layers?**
   - Convolution layers apply filters to input data to capture features like edges, shapes, and textures, forming feature maps used for classification.

8. **What are the key steps in preparing a dataset for training?**
   - Loading images, resizing, normalizing pixel values, and splitting into training, validation, and test sets.

9. **Explain the purpose of normalizing image data.**
   - Normalization scales pixel values to a consistent range (usually 0-1) to help the model train more efficiently and achieve faster convergence.

10. **What is the role of a pooling layer in CNN?**
    - Pooling layers down-sample feature maps, reducing dimensions and computational load while preserving important features.

11. **What is the difference between max pooling and average pooling?**
    - **Max Pooling**: Selects the maximum value in a pooling window, retaining strong features.
    - **Average Pooling**: Calculates the average value, providing a smoother representation.

12. **What is model evaluation in image classification?**
    - Evaluation involves using test data to assess the model's performance,

typically through accuracy, precision, recall, and F1-score metrics.

13. **Explain the purpose of the validation set.**
    - A validation set is used to fine-tune model hyperparameters and prevent overfitting by assessing model performance on unseen data during training.

14. **How does a CNN's architecture differ from a standard Feedforward Neural Network?**
    - CNNs use convolution and pooling layers for spatial data, whereas Feedforward networks typically only have fully connected layers, which may not capture spatial information effectively.

15. **What is data augmentation, and why is it used in image classification?**
    - Data augmentation artificially increases the training dataset by applying transformations (e.g., rotation, flipping), helping the model generalize better to new data.