DEEP LEARNING

CASE STUDY 1

B UDAY KUMAR

HU21CSEN0100964

Design a Convolutional Neural Network to classify the digits in the MNIST digit dataset and evaluate its performance using different metrics.

A Convolutional Neural Network (CNN) stands as a pivotal model in the realm of deep learning, garnering significant acclaim for its effectiveness, especially in image-related tasks.

Its rise in popularity can be attributed to its capacity to leverage multilayer perceptrons for intricate computational tasks.

Unlike traditional image classification algorithms, CNNs require minimal pre-processing, a distinguishing feature that sets them apart.

Instead of relying on manually engineered features, CNNs learn directly from the data through filters.

This self-learning capability allows CNNs to automatically identify and extract hierarchical features, making them highly suited for image processing tasks.

In recent years, the prominence of CNNs has grown as they demonstrate superior performance in various applications.

Their ability to learn hierarchical representations directly from raw pixel data makes them well-suited for image classification, object detection, and other computer vision tasks.

This is particularly evident in scenarios where hand-engineering features might be challenging or impractical.

One of the notable applications of CNNs is on the MNIST dataset, a collection of 28×28 grayscale images depicting hand-written digits from 0 to 9.

With a training set of 60,000 examples and a test set of 10,000 examples, MNIST serves as a standard benchmark for demonstrating the capabilities of CNNs in image classification.

The network learns intricate patterns through filters, eliminating the need for manually engineered features, making CNNs a well-suited choice for image-related tasks.

Designing the Architecture for a Convolutional Neural Network (CNN):

1. **Input Layer**: The initial layer of the CNN is responsible for receiving grayscale images with dimensions of 28x28 pixels. In the context of MNIST, each pixel denotes the intensity of a digit.

2. **Convolutional Layers**: Serving as the fundamental components, convolutional layers play a pivotal role in a CNN. These layers specialize in learning local patterns and features within the input images. Each convolutional layer employs filters (or kernels) that systematically slide across the input image, conducting convolutions to generate feature maps. Activation functions, such as Rectified Linear Unit (ReLU), are commonly applied post-convolution to introduce non-linearities.

3. **Pooling Layers**: Typically, MaxPooling layers are implemented after convolutional layers to reduce the spatial dimensions of the generated feature maps. Pooling facilitates the retention of crucial information while concurrently diminishing computational complexity.

4. **Flatten Layer**: Following the convolutional and pooling layers, the Flatten layer serves to transform the 2D feature maps into a 1D vector. This transformation is imperative as it readies the data for subsequent fully connected layers.

5. **Fully Connected Layers**: Positioned after flattening, dense (fully connected) layers are employed to establish connections among all the features extracted from prior layers. These layers are adept at discerning global patterns and relationships present in the feature maps.

6. **Output Layer**: The concluding layer, the output layer, is composed of nodes equivalent to the number of classes—in the case of MNIST, 10 for digits ranging from 0 to 9. Typically leveraging the softmax activation function, the output layer converts raw scores into class probabilities, facilitating the classification of the input images.

Model Compilation and Training :

1. **Loss Function Selection**: The choice of a loss function holds significant importance, particularly in multi-class classification tasks such as MNIST.
   Categorical cross entropy is a widely adopted loss function for its effectiveness in quantifying the disparity between predicted and actual class probabilities.

2. **Optimizer Application**: Optimizers, such as Adam or Stochastic Gradient Descent (SGD), play a crucial role in the training process. Their primary function is to iteratively adjust the weights of the neural network to minimize the chosen loss function. This iterative adjustment aids in the convergence of the model during training.

3. **Metrics for Evaluation**: Beyond accuracy, a comprehensive model evaluation involves the consideration of additional metrics such as precision, recall, and F1-score.These metrics provide a more nuanced understanding of the model's performance, going beyond a simple accuracy measure.

4. **Training Procedure**: The model undergoes training on the designated training set, and this involves the iterative adjustment of weights through techniques like back propagation and gradient descent. To prevent over fitting, common training practices include the implementation of dropout for regularization and data augmentation. Dropout helps in preventing the model from relying too heavily on specific neurons, while data augmentation introduces variability into the training data, enhancing the model's ability to generalize to unseen examples. This holistic training approach ensures that the model learns robust representations from the training data.

Evaluation:

1. **Testing Process**: The assessment of the model's performance occurs on an independent test set that was not part of the training data. This distinct dataset allows for an unbiased evaluation of the model's ability to generalize to new, unseen instances.

2. **Performance Metrics**:

- Accuracy: This metric signifies the proportion of correctly predicted instances out of the total instances in the test set.

- Precision: Precision denotes the ratio of accurately predicted positive observations to the total instances predicted as positive. It emphasizes the model's ability to avoid false positives.

- Recall: Recall represents the ratio of correctly predicted positive observations to the total actual positive instances. It highlights the model's capacity to capture all relevant positives.

- F1-Score: The F1-Score is a weighted average of precision and recall, providing a balanced measure that considers both false positives and false negatives. This metric is particularly useful when seeking a trade-off between precision and recall.

Confusion Matrix:

A confusion matrix provides a detailed breakdown of model performance by presenting the counts of True Positives, True Negatives, False Positives, and False Negatives. This matrix is instrumental in gaining insights into the accuracy and error rates of a classification model.

Visualization:

The visualization of sample predictions and misclassifications serves as a valuable tool for comprehending the model's performance on individual instances. By inspecting these visual representations, one can gain a nuanced understanding of how well the model distinguishes between different classes and identify areas where improvements or adjustments may be necessary.

Technologies Used:

1. **TensorFlow**: A widely-used open-source machine learning framework developed by the Google Brain team, utilized for constructing and training machine learning models.
2. **Keras**: A high-level neural networks API built on top of TensorFlow, simplifying the definition and training of deep learning models.
3. **MNIST Dataset**: A dataset comprising 28x28 pixel grayscale images of handwritten digits (0 through 9). This dataset serves as a common introduction to image classification tasks in machine learning.
4. **Scikit-Learn Metrics**: A machine learning library that provides efficient tools for data analysis and modeling. It includes functions for computing accuracy, generating classification reports, and constructing confusion matrices.
5. **Matplotlib**: A Python plotting library employed for visualizing predictions, particularly useful for creating informative visual representations.
6. **NumPy**: A Python library that supports large, multidimensional arrays and matrices. It also provides mathematical functions for operations on these arrays.
7. **Loading and Preprocessing MNIST Dataset**: The mnist.load_data() function loads the MNIST dataset, and image pixel values are normalized to the range [0, 1] by dividing by 255 to expedite convergence during training.
8. **Building the CNN Model**: The code defines a CNN model using the Keras Sequential API, incorporating convolutional layers with max-pooling, a flatten layer, and dense (fully connected) layers. ReLU activation functions are applied to convolutional layers, while softmax is used for the output layer, suitable for multi-class classification.

9. **Compiling the Model**: The model is compiled with the Adam optimizer, sparse categorical cross entropy loss (appropriate for integer-encoded class labels), and accuracy as the evaluation metric.
10. **Reshaping the Data**: Input data is reshaped to align with the model's expected input shape, including an additional dimension for the channel.
11. **Training the Model**: The model is trained on the training data (x_train and y_train) for 5 epochs.
12. **Evaluating the Model**: The trained model undergoes evaluation on the test data (x_test and y_test), and the resulting test accuracy is printed.
13. **Making Predictions**: The model is utilized to make predictions on the test data, with np.argmax employed to extract predicted class labels.
14. **Performance Metrics**: Performance metrics, including the confusion matrix, accuracy, precision, recall, and F1-score, are computed and displayed to assess the model's performance on the test set.
15. **Visualizing Predictions**: A subplot of images from the test set is generated to visually represent true labels, predicted labels, and the corresponding images, aiding in the interpretation of model predictions.

Graphs :

Training and Validation Accuracy:

This graph shows the training and validation accuracy over different epochs during the training process.

X-axis: Epochs (the number of times the model has seen the entire training dataset).

Y-axis: Accuracy.

Lines:

Blue line represents the training accuracy, showing how well the model performs on the training data during each epoch.

Orange line represents the validation accuracy, showing how well the model generalizes to unseen validation data during each epoch.

Training and Validation Loss:

This graph displays the training and validation loss over different epochs.

X-axis: Epochs.

Y-axis: Loss (a measure of how well the model is performing; lower is better).

Lines:

Blue line represents the training loss, showing how well the model fits the training data during each epoch.

Orange line represents the validation loss, showing how well the model generalizes to unseen validation data during each epoch.

Understanding Multi Class Classification Metrics

In multi-class classification metrics, the evaluation goes beyond binary outcomes and involves distinguishing between more than two classes. Common metrics include accuracy, precision, recall, and F1-Score, calculated across all classes. Accuracy measures the overall correctness of predictions, while precision, recall, and F1-Score provide insights into the model's performance for individual classes, considering false positives and false negatives. Understanding multi-class metrics is essential for assessing the model's effectiveness across diverse categories in scenarios such as image classification or natural language processing tasks.

Creating confusion matrix for multi class classification and calculating different metrics like precision, recall, Accuracy and F1 Score

To create a confusion matrix for multi-class classification, the matrix dimensions expand to cover interactions between all classes. Each row represents the actual class, and each column corresponds to the predicted class. Precision, recall, accuracy, and F1-Score are derived from the confusion matrix. Precision is calculated as the ratio of true positives to the sum of true positives and false positives for a specific class. Recall is the ratio of true positives to the sum of true positives and false negatives. Accuracy is the sum of diagonal elements divided by the total sum of the matrix. F1-Score is the harmonic mean of precision and recall. These metrics offer a comprehensive evaluation of the model's performance across multiple classes.

Difference between Binary classification and Multi class

The fundamental difference between binary and multi-class classification lies in the number of classes involved. Binary classification deals with distinguishing between two classes, typically denoted as positive and negative. In contrast, multi-class classification involves distinguishing between more than two classes. Binary classification often uses metrics like precision, recall, and accuracy, while multi-class classification extends these metrics to cover multiple classes simultaneously. Understanding this distinction is crucial for selecting appropriate evaluation metrics and interpreting the model's performance based on the task at hand.

Create confusion matrix for binary classification and multi class classification

Creating a confusion matrix for binary classification involves a 2x2 table with entries for true positives, true negatives, false positives, and false negatives. For multi-class classification, the confusion matrix expands to accommodate the interactions between all classes. Each row represents the actual class, and each column represents the predicted class. In binary classification, the matrix has two rows and two columns, while in multi-class classification, the matrix dimensions correspond to the number of classes. The confusion matrix visually summarizes the model's performance, aiding in the calculation of various metrics.

Calculate various metrics from the confusion matrix

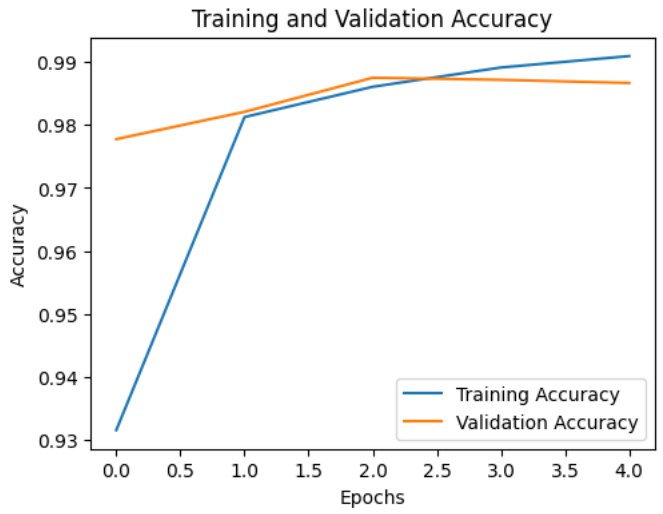
Various metrics can be calculated from the confusion matrix to assess the model's performance. Accuracy is computed as the sum of diagonal elements divided by the total sum of the matrix. Precision for a specific class is the ratio of true positives to the sum of true positives and false positives. Recall is the ratio of true positives to the sum of true positives and false negatives. F1-Score is the harmonic mean of precision and recall. These metrics provide a comprehensive understanding of the model's strengths and weaknesses across different classes, facilitating informed decision-making in a multi-class classification scenario.

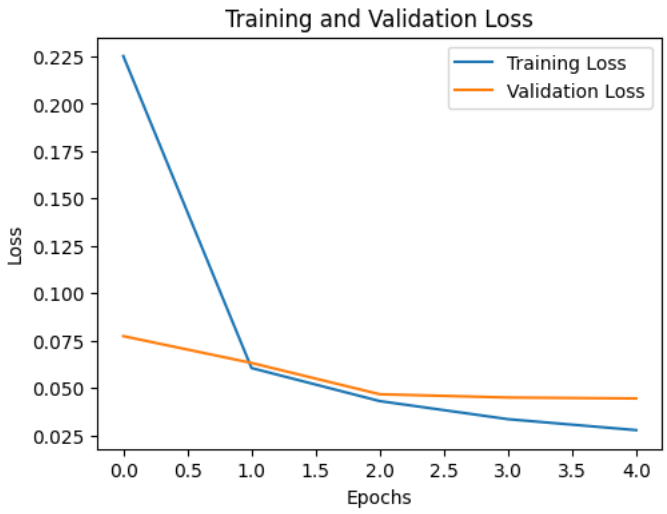
Google Colab :

https://colab.research.google.com/drive/1m0OSjP3TuhvjAaV6Vty97O254g30wMJ9?usp=sharing

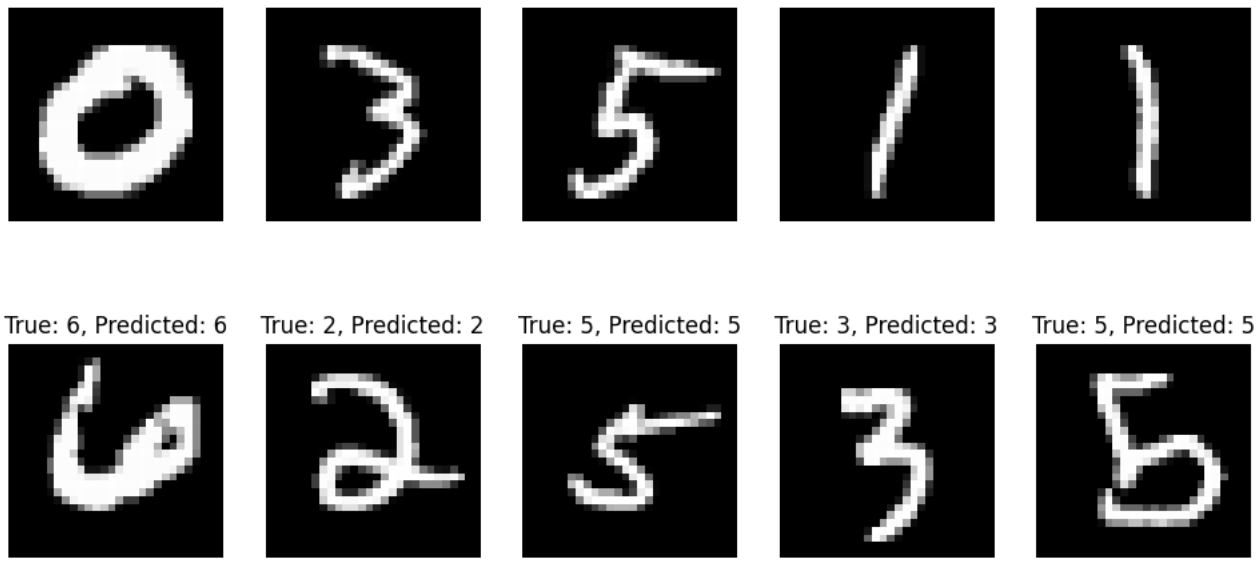
```python
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_recall_fscore_support
import matplotlib.pyplot as plt
import numpy as np

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize pixel values to between 0 and 1

# Reshape the data to fit the model
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Save the model
model.save('mnist_cnn_model.h5')

# Plot accuracy and loss graphs
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {test_acc * 100:.2f}%")

# Select random images from the test set
num_images = 10
random_indices = np.random.choice(len(x_test), num_images)

# Make predictions
predictions = np.argmax(model.predict(x_test[random_indices]), axis=-1)

# Plot the images along with their true and predicted labels
plt.figure(figsize=(12, 6))
for i, idx in enumerate(random_indices):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[idx].reshape(28, 28), cmap='gray')
    plt.title(f'True: {np.argmax(y_test[idx])}, Predicted: {predictions[i]}')
    plt.axis('off')
plt.show()

# Evaluate performance using different metrics
print("\nConfusion Matrix:")
print(confusion_matrix(np.argmax(y_test, axis=1), np.argmax(model.predict(x_test), axis=-1)))

print("\nClassification Report:")
print(classification_report(np.argmax(y_test, axis=1), np.argmax(model.predict(x_test), axis=-1)))

# Calculate overall accuracy and class-wise accuracies
overall_accuracy = accuracy_score(np.argmax(y_test, axis=1), np.argmax(model.predict(x_test), axis=-1))
class_accuracies = np.sum(np.argmax(y_test, axis=1) == np.argmax(model.predict(x_test), axis=-1)) / len(np.argmax(model.predict(x_test), axis=-1))

print(f"\nOverall Accuracy: {overall_accuracy * 100:.2f}%")
print(f"Class-wise Accuracy: {class_accuracies * 100:.2f}%")

# Calculate precision, recall, and F1 Score for each class
precision, recall, f1_score, _ = precision_recall_fscore_support(np.argmax(y_test, axis=1), np.argmax(model.predict(x_test), axis=-1), average=None)

print("\nPrecision for each class:")
for i in range(10):
    print(f"Class {i}: {precision[i]:.4f}")

print("\nRecall for each class:")
for i in range(10):
    print(f"Class {i}: {recall[i]:.4f}")

print("\nF1 Score for each class:")
for i in range(10):
    print(f"Class {i}: {f1_score[i]:.4f}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
750/750 [==============================] - 63s 79ms/step - loss: 0.2252 - accuracy: 0.9315 - val_loss: 0.0774 - val_accuracy: 0.9778
Epoch 2/5
750/750 [==============================] - 39s 52ms/step - loss: 0.0605 - accuracy: 0.9812 - val_loss: 0.0632 - val_accuracy: 0.9821
Epoch 3/5
750/750 [==============================] - 42s 56ms/step - loss: 0.0431 - accuracy: 0.9861 - val_loss: 0.0467 - val_accuracy: 0.9875
Epoch 4/5
750/750 [==============================] - 41s 54ms/step - loss: 0.0335 - accuracy: 0.9891 - val_loss: 0.0450 - val_accuracy: 0.9872
Epoch 5/5
750/750 [==============================] - 40s 54ms/step - loss: 0.0278 - accuracy: 0.9909 - val_loss: 0.0445 - val_accuracy: 0.9867
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recc
  saving_api.save_model(
```






```
313/313 [==============================] - 3s 8ms/step - loss: 0.0366 - accuracy: 0.9884

Test Accuracy: 98.84%
1/1 [==============================] - 0s 121ms/step
```




```
Confusion Matrix:
313/313 [==============================] - 2s 8ms/step
[[ 971    0    1    1    0    2    2    0    2    1]
 [   0 1134    0    1    0    0    0    0    0    0]
 [   1    4 1026    0    0    0    0    1    0    0]
 [   0    0    2 1006    0    2    0    0    0    0]
 [   0    1    1    0  972    0    0    1    0    7]
 [   1    0    0   10    0  879    1    0    1    0]
 [   4    3    0    0    2    3  944    0    2    0]
 [   0    9   13    3    0    1    0  990    0   12]
 [   1    0    4    2    0    2    1    0  962    2]
 [   0    2    1    0    2    2    1    0    1 1000]]

Classification Report:
313/313 [==============================] - 3s 9ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.98      1.00      0.99      1135
           2       0.98      0.99      0.99      1032
           3       0.98      1.00      0.99      1010
           4       1.00      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       1.00      0.96      0.98      1028
           8       0.99      0.99      0.99       974
           9       0.98      0.99      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

313/313 [==============================] - 2s 8ms/step
313/313 [==============================] - 2s 8ms/step
313/313 [==============================] - 3s 11ms/step

Overall Accuracy: 98.84%
Class-wise Accuracy: 98.84%
313/313 [==============================] - 2s 8ms/step

Precision for each class:
Class 0: 0.9928
Class 1: 0.9835
Class 2: 0.9790
Class 3: 0.9834
Class 4: 0.9959
Class 5: 0.9865
Class 6: 0.9947
Class 7: 0.9980
Class 8: 0.9938
Class 9: 0.9785

Recall for each class:
Class 0: 0.9908
Class 1: 0.9991
Class 2: 0.9942
Class 3: 0.9960
Class 4: 0.9898
Class 5: 0.9854
Class 6: 0.9854
Class 7: 0.9630
Class 8: 0.9877
Class 9: 0.9911
```

Class 9: 0.9911

F1 Score for each class:
Class 0: 0.9918
Class 1: 0.9913
Class 2: 0.9865
Class 3: 0.9897
Class 4: 0.9928
Class 5: 0.9860
Class 6: 0.9900
Class 7: 0.9802
Class 8: 0.9907
Class 9: 0.9847