Experiment No : 8

Aim: MNIST digit classification before and after shuffling Train CNN on Original Data Train CNN on shuffled data.

Theory:

MNIST Dataset: The MNIST dataset consists of 28x28 grayscale images of handwritten digits (0-9).
Each image is a grid of pixel values representing the intensity of ink at each pixel location.

Convolutional Neural Networks (CNNs):

CNNs are well-suited for image classification tasks due to their ability to capture spatial hierarchies and features in images.

Training a CNN:

CNNs consist of convolutional layers for feature extraction, pooling layers for downsampling, and fully connected layers for classification.
The network is trained to minimize a loss function (cross-entropy) by adjusting its weights and biases using backpropagation and an optimization algorithm (e.g., Adam).

Original Data:

When training a CNN on the original MNIST data, the images are presented to the network in their natural order.
The network learns patterns and features from the data as it is originally structured.

Shuffled Data:

In the second part of the experiment, the MNIST data is shuffled. This randomizes the order in which images are presented to the network during training.
Shuffling the data disrupts any inherent sequential patterns, potentially making the learning task more challenging.

Effects of Shuffling:

Shuffling the data introduces variability during training, which can help the model generalize better to unseen data.
It prevents the model from overfitting to any underlying order or patterns in the dataset.

Performance Comparison:

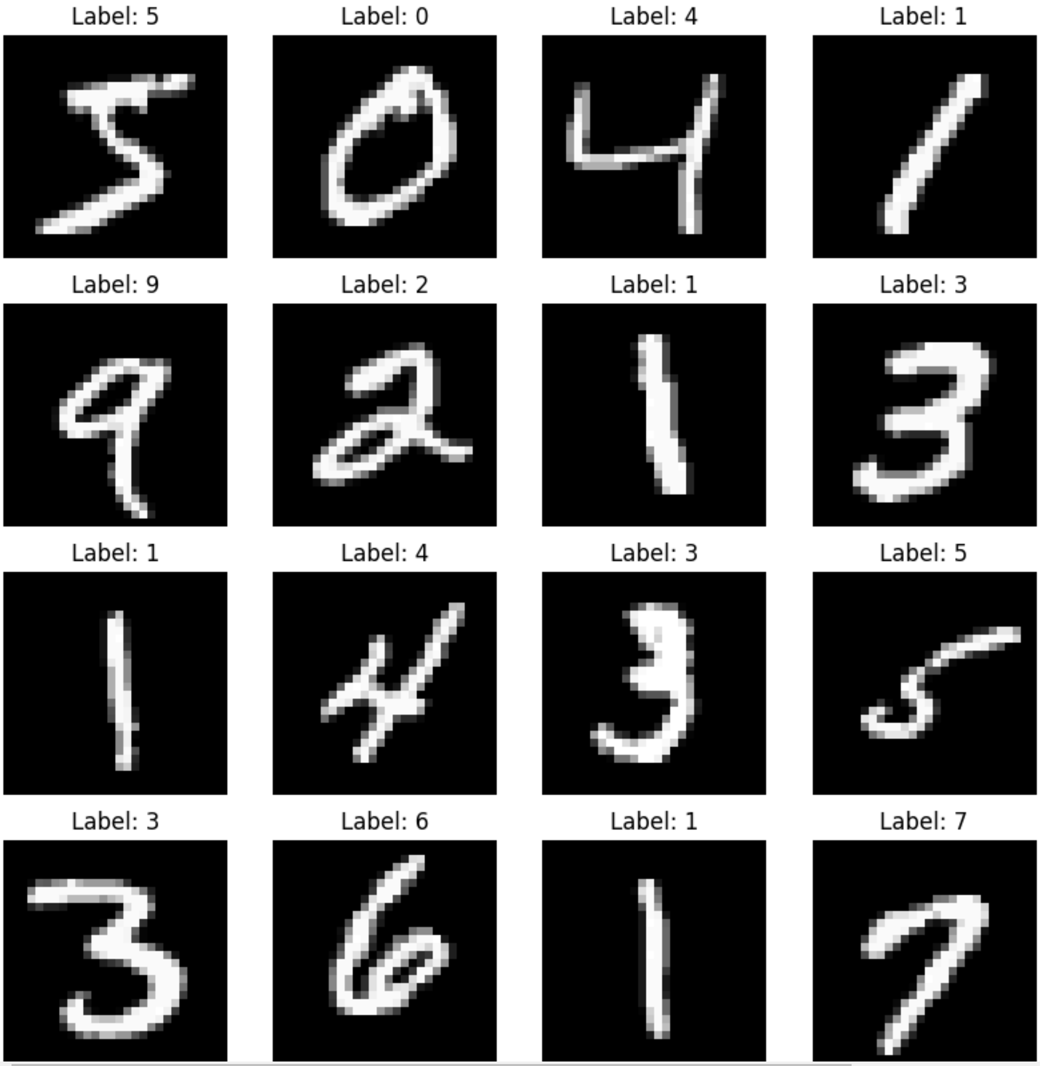
After training on the shuffled data, the CNN might exhibit different performance characteristics compared to the model trained on the original data.
It may demonstrate better generalization but potentially require more training epochs to achieve similar accuracy.

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

```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Display some sample images from the dataset
plt.figure(figsize=(10,10))
for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.show()
```




```
# Convert labels to categorical
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([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    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, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

```
# Train the model
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 56s 29ms/step - accuracy: 0.8902 - loss: 0.6145 - val_accuracy: 0.9712 - val_loss: 0.0905
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 29ms/step - accuracy: 0.9803 - loss: 0.0656 - val_accuracy: 0.9839 - val_loss: 0.0526
Epoch 3/5
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 54s 29ms/step - accuracy: 0.9857 - loss: 0.0477 - val_accuracy: 0.9844 - val_loss: 0.0557
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 54s 29ms/step - accuracy: 0.9878 - loss: 0.0404 - val_accuracy: 0.9758 - val_loss: 0.0886
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 30ms/step - accuracy: 0.9882 - loss: 0.0373 - val_accuracy: 0.9858 - val_loss: 0.0467
```

```python
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_acc}")
```

```
313/313 - 2s - 7ms/step - accuracy: 0.9858 - loss: 0.0467
Test accuracy: 0.98580002784729
```

```python
# Plot training & validation accuracy and loss
plt.figure(figsize=(12, 4))
```

```
<Figure size 1200x400 with 0 Axes>
```

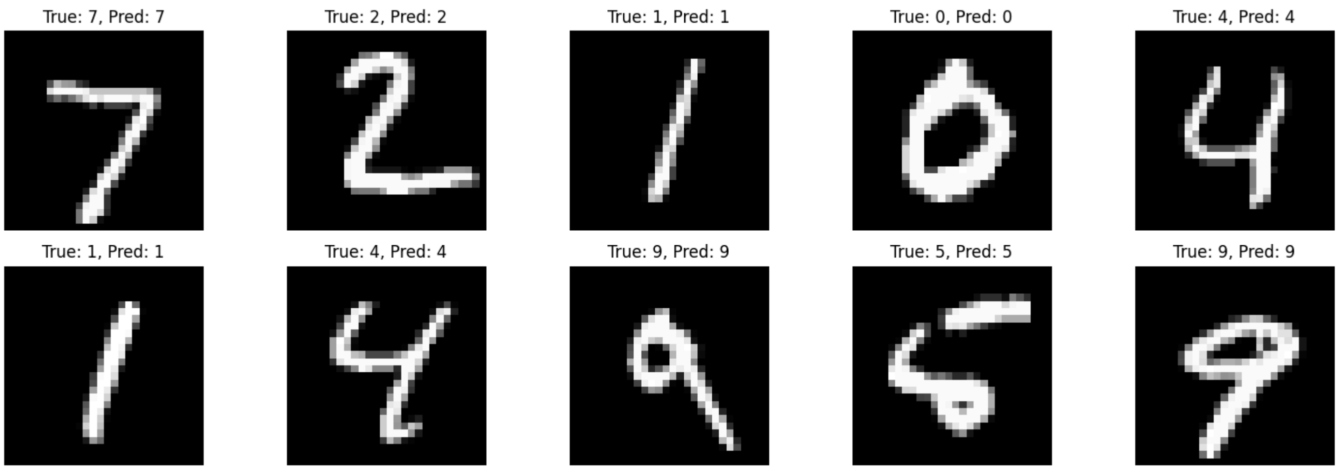
```python
# Make predictions on the test set
predictions = model.predict(x_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step
```

```python
# Display a few test images with predicted and true labels
num_images = 10
plt.figure(figsize=(15, 5))

for i in range(num_images):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    predicted_label = np.argmax(predictions[i])
    true_label = np.argmax(y_test[i])
    plt.title(f"True: {true_label}, Pred: {predicted_label}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```



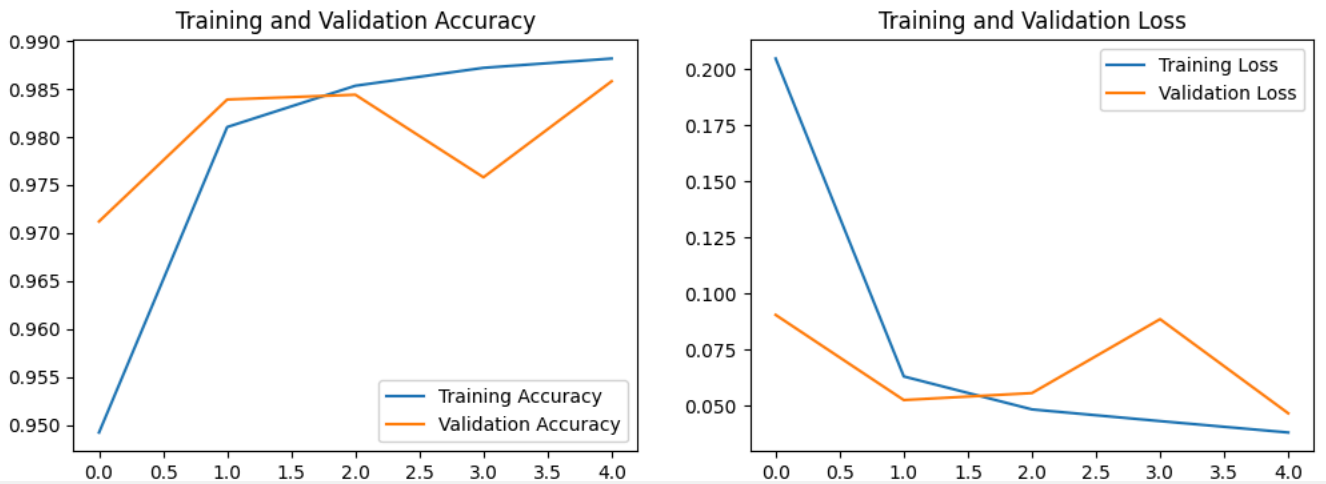

```python
plt.figure(figsize=(12, 4))

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.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.legend()

plt.show()
```



Conclusion:

Training a CNN on both original and shuffled data allows us to observe the impact of data order on the learning process.

Shuffling data is a common practice to ensure robustness and generalization in machine learning models, particularly for datasets with inherent order or bias.

The choice between shuffled and original data depends on the problem's requirements, and the experiment highlights the importance of data preprocessing in deep learning.