Week-6: Construct convolution neural network and perform the classification using MNIST dataset using K10 cross validation.

Convolutional Neural Network: Convolutional networks (LeCun, 1989), also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology.

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation.

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Pooling: A typical layer of a convolutional network consists of three stages. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage. In the third stage, we use a pooling function to modify the output of the layer further.

Pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, **the max pooling**

K10 cross validation: K-fold cross-validation is a technique for evaluating predictive models. The dataset is divided into k subsets or folds. The model is trained and evaluated k times, using a different fold as the validation set each time. Performance metrics from each fold are averaged to estimate the model’s generalization performance. This method aids in model assessment, selection, and hyperparameter tuning, providing a more reliable measure of a model’s effectiveness.

To achieve this K-Fold Cross Validation, we have to split the data set into three sets, Training, Testing, and Validation, with the challenge of the volume of the data.

```
Example: kf = KFold(n_splits=2)
```

Process-1: Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import mnist
from sklearn.metrics import accuracy_score

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

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Expand dimensions to add a channel dimension
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# Define the CNN model with 4 layers
def create_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.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='sgd',
                  loss='mse',
                  metrics=['accuracy'])
    return model

# Print model summary
model=create_model()
model.summary()
```

```
# Perform K-fold cross-validation
k_folds = 10
kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)

accuracies = []

for train_index, val_index in kf.split(x_train):
    x_train_fold, x_val_fold = x_train[train_index], x_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    model = create_model()
    # Train the model
    model.fit(x_train_fold, y_train_fold, epochs=5, batch_size=64, verbose=0)

    # Evaluate on the validation set
    val_predictions = np.argmax(model.predict(x_val_fold), axis=-1)
    accuracy = accuracy_score(y_val_fold, val_predictions)
    accuracies.append(accuracy)

    #Print the Accuracy over the K fold
    print('Accuracy is : ',accuracy)

# Print the average accuracy over the K folds
print(f'Average Accuracy: {np.mean(accuracies)}')
```

<u>Output:</u>

```
188/188 [==============================] - 2s 7ms/step
Accuracy is :  0.0925
188/188 [==============================] - 1s 7ms/step
Accuracy is :  0.14716666666666667
188/188 [==============================] - 2s 12ms/step
Accuracy is :  0.114
188/188 [==============================] - 2s 11ms/step
Accuracy is :  0.05483333333333333
188/188 [==============================] - 2s 8ms/step
Accuracy is :  0.10116666666666667
188/188 [==============================] - 1s 7ms/step
Accuracy is :  0.08783333333333333
188/188 [==============================] - 1s 7ms/step
Accuracy is :  0.1005
188/188 [==============================] - 2s 10ms/step
Accuracy is :  0.036
188/188 [==============================] - 2s 8ms/step
Accuracy is :  0.07216666666666667
188/188 [==============================] - 1s 7ms/step
Accuracy is :  0.07233333333333333


Average Accuracy: 0.08785000000000001
```

Process-2: Code:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

%cd C:\Users\Zai\Untitled Folder

# Load your own training and test datasets
# Make sure to normalize the pixel values to be between 0 and 1
# Also, make sure to expand dimensions to add a channel dimension
if needed

train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Extract features and labels
x_train, y_train = train_data.drop('label', axis=1), train_data['label']
x_test, y_test = train_test_split(test_data,test_size=0.2,random_state=2)

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# If needed, expand dimensions to add a channel dimension
# For example, if your data is grayscale, you may need to add a chann
el dimension
x_train = np.expand_dims(x_train.values.reshape(-1, 28, 28), axis=-1)
x_test = np.expand_dims(x_test.values.reshape(-1, 28, 28), axis=-1)

# Define the CNN model with 4 layers
def create_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.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Perform K-fold cross-validation
k_folds = 10
kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)

accuracies = []

for train_index, val_index in kf.split(x_train):
    x_train_fold, x_val_fold = x_train[train_index], x_train[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

    model = create_model()

    # Train the model
    model.fit(x_train_fold, y_train_fold, epochs=5, batch_size=128, verbose=
0)

    # Evaluate on the validation set
    val_predictions = np.argmax(model.predict(x_val_fold), axis=-1)
    accuracy = accuracy_score(y_val_fold, val_predictions)
    accuracies.append(accuracy)

    #Print the Accuracy over the K fold
    print('Accuracy is : ',accuracy)

# Print the average accuracy over the K folds
print(f'Average Accuracy: {np.mean(accuracies)}')
```

Output:

```
132/132 [==============================] - 1s 8ms/step
Accuracy is :  0.9873809523809524
132/132 [==============================] - 2s 13ms/step
Accuracy is :  0.9864285714285714
132/132 [==============================] - 1s 10ms/step
Accuracy is :  0.9888095238095238
132/132 [==============================] - 2s 13ms/step
Accuracy is :  0.9873809523809524
132/132 [==============================] - 1s 8ms/step
Accuracy is :  0.9835714285714285
132/132 [==============================] - 2s 11ms/step
Accuracy is :  0.9835714285714285
132/132 [==============================] - 1s 6ms/step
Accuracy is :  0.9854761904761905
132/132 [==============================] - 1s 8ms/step
Accuracy is :  0.9852380952380952
132/132 [==============================] - 1s 6ms/step
Accuracy is :  0.9845238095238096
132/132 [==============================] - 1s 7ms/step
Accuracy is :  0.9847619047619047
Average Accuracy: 0.9857142857142858
```