Machine Learning Fundamentals

Group Members:

1. Abhishek Neerukonda - 101159833
2. Lakshmi Varaha Krishna Chittella - 101163679
3. Aditya Sambhara - 101162272
4. Sarada Satya Sai Kiran Malladi - 101163459
5. Pavan Venkata Sai Kalyan Gedala - 101144113
6. Suresh Boyapati - 101149244
7. Rajesh Reddy Guthikonda - 101179183
8. Mahesh Babu Kancham - 101164409
9. Sridhar Kancham - 101165877
10. Kalyan Kasinadhuni - 101165472

Contents:

CNN MODEL FOR OPTICAL RECOGNITION OF HANDWRITTEN DIGITS

Introduction:

1.Project Overview:

This project aims to develop a convolutional neural network (CNN) model for optical recognition of handwritten digits. The model is trained and evaluated using the Optical Recognition of Handwritten Digits dataset, obtained from the UCI Machine Learning Repository. The dataset contains samples of handwritten digits (0-9) in a format of 8x8 pixel images.

2.Dataset Explanation:

This dataset consists of samples of handwritten digits collected from a total of 43 people, 30 contributed to the training set, and 13 to the test set. Each sample is an 8x8 grayscale image of a handwritten digit (0 through 9), representing the numerical value of the digit. The task is to recognize these handwritten digits automatically.

3.Methodology:

Dataset Preparation:

The dataset preparation involves fetching the Optical Recognition of Handwritten Digits dataset from the UCI Machine Learning Repository and preprocessing it for model training. This includes loading the dataset, examining its metadata, converting it to numpy arrays, reshaping the features, normalizing the data, and one-hot encoding the target labels.

Model Architecture:

The model architecture consists of a convolutional neural network (CNN) designed using the TensorFlow Keras API. The architecture comprises convolutional layers followed by max-pooling and dropout layers for feature extraction, and dense layers for classification. The final layer uses softmax activation to output probabilities for each digit class. The Individual components are:

- **Convolutional layers:**
  layers.Conv2D: This layer creates a convolution kernel that is convolved with the input image to produce a tensor of outputs. It applies a specified number of filters (or kernels) to the input image, extracting features through convolution operations.

- **Max pooling layers:**
  layers.MaxPooling2D: This layer downsamples the input representation, reducing its dimensionality and computational complexity. It extracts the most important features by retaining the maximum value in each region of the input.

- **Dropout Layers:**
  layers.Dropout: This layer applies dropout regularization to the input, randomly setting a fraction of input units to zero during training to prevent overfitting.

- **Dense Layers:**
  layers.Dense: This layer is a densely connected neural network layer, where each neuron is connected to every neuron in the previous and next layers.

- **Activation Functions:**
  Activation functions introduce non-linearity into the neural network, enabling it to learn complex patterns in the data. Common activation functions include ReLU (Rectified Linear Unit) and softmax.

Training Process:

The training process involves k-fold cross-validation with k=5 to train the model. Each fold of the cross-validation splits the dataset into training and validation sets. The model is trained on the training set and evaluated on the validation set for multiple epochs. Training metrics such as loss and accuracy are monitored and recorded.

4.Results:

Training Results:

Training results include visualizations of training and validation loss, as well as training and validation accuracy across all folds. Additionally, average loss and accuracy metrics are calculated to assess the model's performance.

Evaluation Results:

Evaluation results comprise a confusion matrix and classification report, providing insights into the model's performance in classifying handwritten digits. The confusion matrix visualizes the model's predictions compared to the actual labels, while the classification report presents precision, recall, F1-score, and support for each class.

Error Analysis:

Error analysis involves examining misclassified samples and identifying patterns or common errors made by the model. This analysis helps in understanding the model's weaknesses and potential areas for improvement.

5.Conclusion:

The conclusion summarizes the project's objectives, methodology, findings, and implications. It discusses the effectiveness of the developed CNN model in recognizing handwritten digits and highlights key insights gained from the evaluation results and error analysis.

6.Future Works:

Future works outline potential directions for further research or improvements to the model. This may include experimenting with different architectures, optimizing hyperparameters, exploring advanced techniques such as data augmentation, or extending the model to handle more complex datasets or tasks. By organizing the project documentation into these sections, you can provide a clear and comprehensive overview of the dataset preparation, model architecture, training process, evaluation results, error analysis, conclusion, and suggestions for future works.

7.Reference:

https://github.com/Kmohamedalie/Optical-Recognition-of-Handwritten-Digits

MLF_CNN

May 1, 2024

[1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_openml
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras import layers, models, utils
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from tensorflow.keras.utils import to_categorical  # Correctly importing␣
 ↪to_categorical
from ucimlrepo import fetch_ucirepo
```

[2]:
```
# Fetch dataset
optical_recognition_of_handwritten_digits = fetch_ucirepo(id=80)

# Data as pandas dataframes
X = optical_recognition_of_handwritten_digits.data.features
y = optical_recognition_of_handwritten_digits.data.targets

# Metadata
print(optical_recognition_of_handwritten_digits.metadata)

# Variable information
print(optical_recognition_of_handwritten_digits.variables)

# Optionally, print some information about the data
print("Features shape:", X.shape)
print("Target shape:", y.shape)
print("Unique digits in target:", np.unique(y))
```

```
{'uci_id': 80, 'name': 'Optical Recognition of Handwritten Digits',
'repository_url': 'https://archive.ics.uci.edu/dataset/80/optical+recognition+of
+handwritten+digits', 'data_url':
'https://archive.ics.uci.edu/static/public/80/data.csv', 'abstract': 'Two
```

versions of this database available; see folder', 'area': 'Computer Science', 'tasks': ['Classification'], 'characteristics': ['Multivariate'], 'num_instances': 5620, 'num_features': 64, 'feature_types': ['Integer'], 'demographics': [], 'target_col': ['class'], 'index_col': None, 'has_missing_values': 'no', 'missing_values_symbol': None, 'year_of_dataset_creation': 1998, 'last_updated': 'Wed Aug 23 2023', 'dataset_doi': '10.24432/C50P49', 'creators': ['E. Alpaydin', 'C. Kaynak'], 'intro_paper': {'title': 'Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition', 'authors': 'C. Kaynak', 'published_in': 'MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University', 'year': 1995, 'url': None, 'doi': None}, 'additional_info': {'summary': 'We used preprocessing programs made available by NIST to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions.\r\n\r\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994.', 'purpose': None, 'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'All input attributes are integers in the range 0..16.\r\nThe last attribute is the class code 0..9', 'citation': None}}

```
           name     role         type demographic description units  \
0    Attribute1  Feature      Integer        None        None  None
1    Attribute2  Feature      Integer        None        None  None
2    Attribute3  Feature      Integer        None        None  None
3    Attribute4  Feature      Integer        None        None  None
4    Attribute5  Feature      Integer        None        None  None
..          …       …            …           …           …    …
60  Attribute61  Feature      Integer        None        None  None
61  Attribute62  Feature      Integer        None        None  None
62  Attribute63  Feature      Integer        None        None  None
63  Attribute64  Feature      Integer        None        None  None
64        class   Target  Categorical        None        None  None

   missing_values
0              no
1              no
2              no
3              no
4              no
..            …
60             no
61             no
62             no
```

```
63              no
64              no

[65 rows x 7 columns]
Features shape: (5620, 64)
Target shape: (5620, 1)
Unique digits in target: [0 1 2 3 4 5 6 7 8 9]
```

[3]:
```python
def create_model():
    model = models.Sequential([
        layers.Conv2D(32, kernel_size=(3, 3), activation='relu',␣
↪input_shape=(8, 8, 1)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.25),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
    return model

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 6, 6, 32)          320

 max_pooling2d (MaxPooling2  (None, 3, 3, 32)          0
 D)

 dropout (Dropout)           (None, 3, 3, 32)          0

 conv2d_1 (Conv2D)           (None, 1, 1, 64)          18496

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 128)               8320

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290
```

```
=================================================================
Total params: 28426 (111.04 KB)
Trainable params: 28426 (111.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[4]:
```
# Convert features and labels to numpy arrays (if they are not already)
X = np.array(X)
y = np.array(y)

# Reshape X to fit the model input shape: (n_samples, 8, 8, 1)
X = X.reshape(-1, 8, 8, 1)

# Normalize the feature data
X = X.astype('float32') / 16  # Assuming pixel values range from 0 to 16

# One-hot encode the target labels
y = to_categorical(y, num_classes=10)
```

[5]:
```
# Initialize lists to store test labels and predictions
y_true = []
y_pred = []
all_train_loss = []
all_val_loss = []
all_train_acc = []
all_val_acc = []

num_folds = 5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_no = 1

for train, test in kf.split(X):
    print(f'Training fold {fold_no}...')
    ␣
 ↪print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')

    # Fit data to model
    history = model.fit(X[train], y[train],
                        batch_size=32,
                        epochs=10,
                        verbose=1,
                        validation_data=(X[test], y[test]))

    # Append loss and accuracy
    all_train_loss.extend(history.history['loss'])
    all_val_loss.extend(history.history['val_loss'])
```

```python
    all_train_acc.extend(history.history['accuracy'])
    all_val_acc.extend(history.history['val_accuracy'])

    # Generate generalization metrics
    scores = model.evaluate(X[test], y[test], verbose=0)
    print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]};␣
 ↪{model.metrics_names[1]} of {scores[1]*100}%')

    # Predict on the test data
    predictions = model.predict(X[test], batch_size=32)
    y_pred.extend(np.argmax(predictions, axis=1))
    y_true.extend(np.argmax(y[test], axis=1))

    fold_no += 1
```

```
Training fold 1…
------------------------------------------------------------------------
Training for fold 1 …
Epoch 1/10
141/141 [==============================] - 1s 3ms/step - loss: 1.4540 -
accuracy: 0.5347 - val_loss: 0.3917 - val_accuracy: 0.9128
Epoch 2/10
141/141 [==============================] - 0s 2ms/step - loss: 0.4386 -
accuracy: 0.8648 - val_loss: 0.2316 - val_accuracy: 0.9288
Epoch 3/10
141/141 [==============================] - 0s 2ms/step - loss: 0.3113 -
accuracy: 0.9070 - val_loss: 0.1645 - val_accuracy: 0.9466
Epoch 4/10
141/141 [==============================] - 0s 2ms/step - loss: 0.2383 -
accuracy: 0.9315 - val_loss: 0.1366 - val_accuracy: 0.9573
Epoch 5/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1842 -
accuracy: 0.9464 - val_loss: 0.1118 - val_accuracy: 0.9671
Epoch 6/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1718 -
accuracy: 0.9475 - val_loss: 0.1018 - val_accuracy: 0.9751
Epoch 7/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1382 -
accuracy: 0.9584 - val_loss: 0.0866 - val_accuracy: 0.9786
Epoch 8/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1198 -
accuracy: 0.9646 - val_loss: 0.0820 - val_accuracy: 0.9742
Epoch 9/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1105 -
accuracy: 0.9682 - val_loss: 0.0727 - val_accuracy: 0.9786
Epoch 10/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1015 -
accuracy: 0.9693 - val_loss: 0.0658 - val_accuracy: 0.9795
```

```
Score for fold 1: loss of 0.0657673254609108; accuracy of 97.95373678207397%
36/36 [==============================] - 0s 777us/step
Training fold 2…
------------------------------------------------------------------------
Training for fold 2 …
Epoch 1/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0937 -
accuracy: 0.9693 - val_loss: 0.0420 - val_accuracy: 0.9867
Epoch 2/10
141/141 [==============================] - 0s 2ms/step - loss: 0.1025 -
accuracy: 0.9691 - val_loss: 0.0396 - val_accuracy: 0.9893
Epoch 3/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0853 -
accuracy: 0.9751 - val_loss: 0.0402 - val_accuracy: 0.9893
Epoch 4/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0697 -
accuracy: 0.9782 - val_loss: 0.0475 - val_accuracy: 0.9840
Epoch 5/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0749 -
accuracy: 0.9780 - val_loss: 0.0431 - val_accuracy: 0.9831
Epoch 6/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0745 -
accuracy: 0.9760 - val_loss: 0.0404 - val_accuracy: 0.9858
Epoch 7/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0705 -
accuracy: 0.9789 - val_loss: 0.0356 - val_accuracy: 0.9893
Epoch 8/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0592 -
accuracy: 0.9827 - val_loss: 0.0394 - val_accuracy: 0.9867
Epoch 9/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0593 -
accuracy: 0.9798 - val_loss: 0.0459 - val_accuracy: 0.9831
Epoch 10/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0518 -
accuracy: 0.9849 - val_loss: 0.0519 - val_accuracy: 0.9813
Score for fold 2: loss of 0.05190756916999817; accuracy of 98.13167452812195%
36/36 [==============================] - 0s 829us/step
Training fold 3…
------------------------------------------------------------------------
Training for fold 3 …
Epoch 1/10
141/141 [==============================] - 0s 3ms/step - loss: 0.0681 -
accuracy: 0.9775 - val_loss: 0.0187 - val_accuracy: 0.9920
Epoch 2/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0597 -
accuracy: 0.9804 - val_loss: 0.0165 - val_accuracy: 0.9929
Epoch 3/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0551 -
```

```
accuracy: 0.9824 - val_loss: 0.0185 - val_accuracy: 0.9938
Epoch 4/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0464 -
accuracy: 0.9878 - val_loss: 0.0223 - val_accuracy: 0.9902
Epoch 5/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0485 -
accuracy: 0.9840 - val_loss: 0.0211 - val_accuracy: 0.9920
Epoch 6/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0452 -
accuracy: 0.9844 - val_loss: 0.0193 - val_accuracy: 0.9920
Epoch 7/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0423 -
accuracy: 0.9858 - val_loss: 0.0226 - val_accuracy: 0.9938
Epoch 8/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0402 -
accuracy: 0.9878 - val_loss: 0.0248 - val_accuracy: 0.9920
Epoch 9/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0459 -
accuracy: 0.9851 - val_loss: 0.0206 - val_accuracy: 0.9929
Epoch 10/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0334 -
accuracy: 0.9893 - val_loss: 0.0232 - val_accuracy: 0.9929
Score for fold 3: loss of 0.02319428138434887; accuracy of 99.28825497627258%
36/36 [==============================] - 0s 798us/step
Training fold 4…
------------------------------------------------------------------------
Training for fold 4 …
Epoch 1/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0391 -
accuracy: 0.9878 - val_loss: 0.0063 - val_accuracy: 0.9991
Epoch 2/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0358 -
accuracy: 0.9907 - val_loss: 0.0115 - val_accuracy: 0.9956
Epoch 3/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0426 -
accuracy: 0.9873 - val_loss: 0.0125 - val_accuracy: 0.9947
Epoch 4/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0341 -
accuracy: 0.9889 - val_loss: 0.0088 - val_accuracy: 0.9964
Epoch 5/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0342 -
accuracy: 0.9898 - val_loss: 0.0122 - val_accuracy: 0.9956
Epoch 6/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0291 -
accuracy: 0.9915 - val_loss: 0.0160 - val_accuracy: 0.9947
Epoch 7/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0279 -
accuracy: 0.9902 - val_loss: 0.0130 - val_accuracy: 0.9964
```

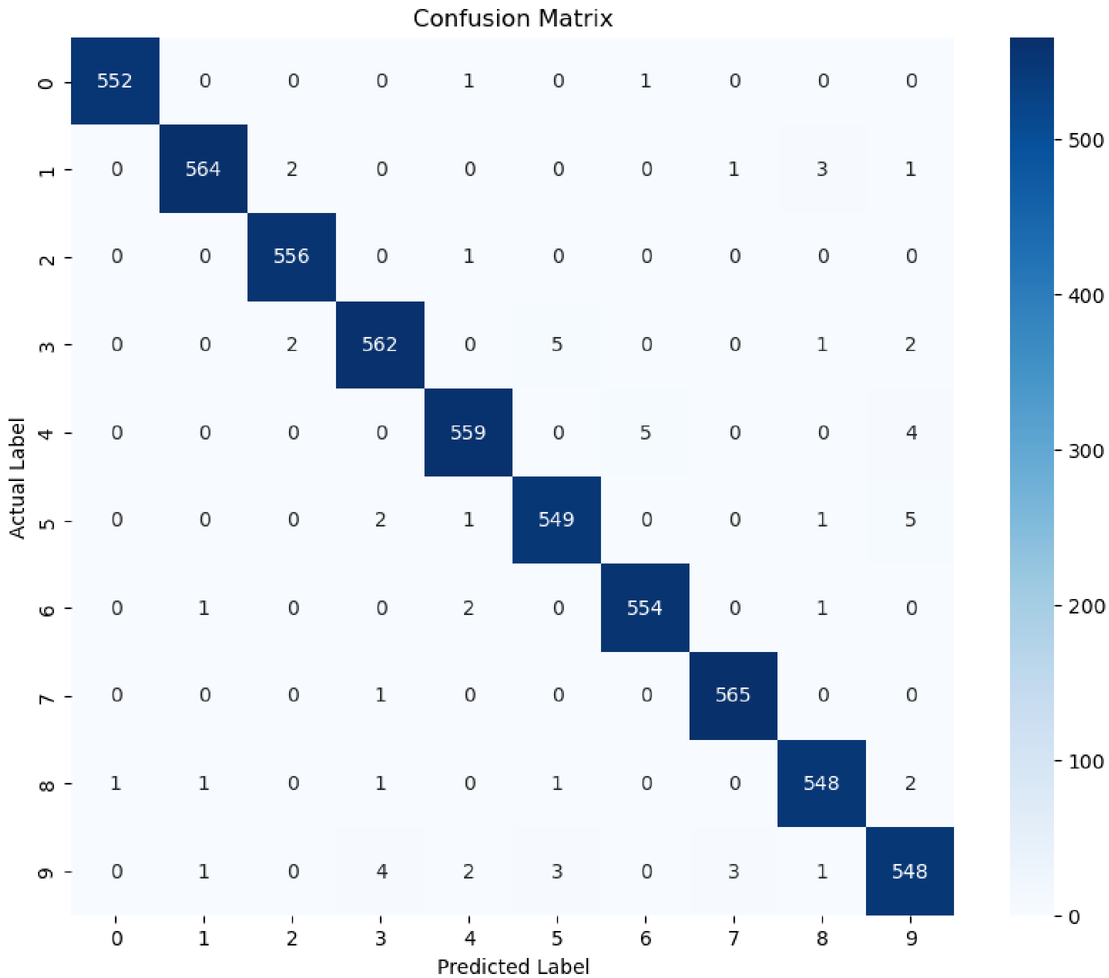
```
Epoch 8/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0298 -
accuracy: 0.9907 - val_loss: 0.0108 - val_accuracy: 0.9964
Epoch 9/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0298 -
accuracy: 0.9900 - val_loss: 0.0126 - val_accuracy: 0.9947
Epoch 10/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0227 -
accuracy: 0.9929 - val_loss: 0.0136 - val_accuracy: 0.9947
Score for fold 4: loss of 0.013570494949817657; accuracy of 99.46619272232056%
36/36 [==============================] - 0s 799us/step
Training fold 5…
------------------------------------------------------------------------
Training for fold 5 …
Epoch 1/10
141/141 [==============================] - 0s 3ms/step - loss: 0.0278 -
accuracy: 0.9913 - val_loss: 0.0040 - val_accuracy: 0.9982
Epoch 2/10
141/141 [==============================] - 0s 3ms/step - loss: 0.0207 -
accuracy: 0.9944 - val_loss: 0.0048 - val_accuracy: 0.9973
Epoch 3/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0309 -
accuracy: 0.9902 - val_loss: 0.0063 - val_accuracy: 0.9973
Epoch 4/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0286 -
accuracy: 0.9911 - val_loss: 0.0100 - val_accuracy: 0.9973
Epoch 5/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0285 -
accuracy: 0.9920 - val_loss: 0.0071 - val_accuracy: 0.9982
Epoch 6/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0235 -
accuracy: 0.9927 - val_loss: 0.0111 - val_accuracy: 0.9956
Epoch 7/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0245 -
accuracy: 0.9918 - val_loss: 0.0132 - val_accuracy: 0.9956
Epoch 8/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0217 -
accuracy: 0.9935 - val_loss: 0.0092 - val_accuracy: 0.9964
Epoch 9/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0274 -
accuracy: 0.9904 - val_loss: 0.0069 - val_accuracy: 0.9973
Epoch 10/10
141/141 [==============================] - 0s 2ms/step - loss: 0.0181 -
accuracy: 0.9944 - val_loss: 0.0086 - val_accuracy: 0.9956
Score for fold 5: loss of 0.008615042082965374; accuracy of 99.5551586151123%
36/36 [==============================] - 0s 794us/step
```

[6]:

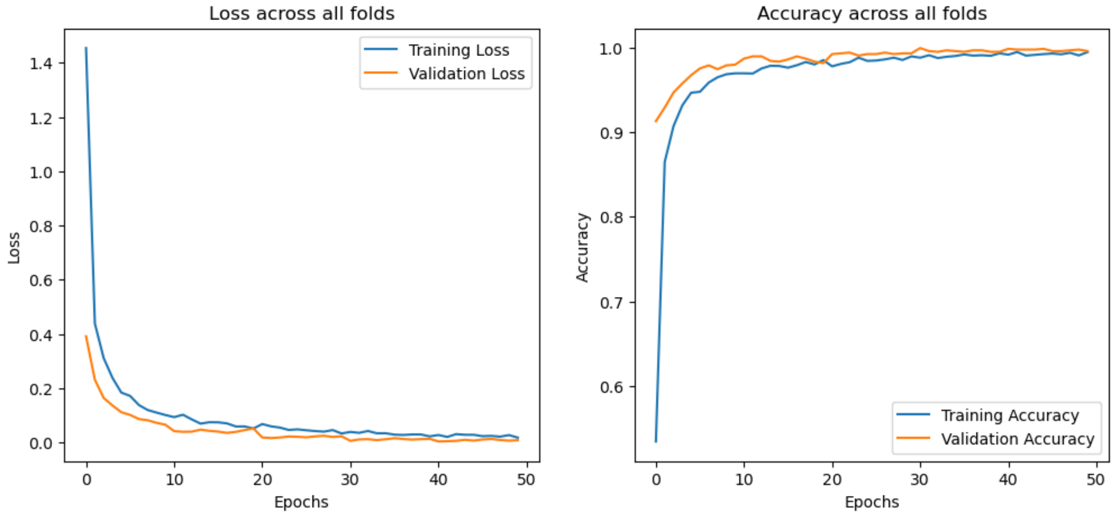
```python
# Generate the confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:")
print(cm)

# Classification report
report = classification_report(y_true, y_pred)
print("Classification Report:")
print(report)

# Plotting the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()

# Calculate average loss and accuracy
average_loss = np.mean(all_val_loss)
average_accuracy = np.mean(all_val_acc) * 100

print(f'Average Loss: {average_loss}')
print(f'Average Accuracy: {average_accuracy}%')

# Plot training & validation loss values
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(all_train_loss, label='Training Loss')
plt.plot(all_val_loss, label='Validation Loss')
plt.title('Loss across all folds')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(all_train_acc, label='Training Accuracy')
plt.plot(all_val_acc, label='Validation Accuracy')
plt.title('Accuracy across all folds')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
Confusion Matrix:
[[552   0   0   0   1   0   1   0   0   0]
```

```
 [  0 564   2   0   0   0   0   1   3   1]
 [  0   0 556   0   1   0   0   0   0   0]
 [  0   0   2 562   0   5   0   0   1   2]
 [  0   0   0   0 559   0   5   0   0   4]
 [  0   0   0   2   1 549   0   0   1   5]
 [  0   1   0   0   2   0 554   0   1   0]
 [  0   0   0   1   0   0   0 565   0   0]
 [  1   1   0   1   0   1   0   0 548   2]
 [  0   1   0   4   2   3   0   3   1 548]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       554
           1       0.99      0.99      0.99       571
           2       0.99      1.00      1.00       557
           3       0.99      0.98      0.98       572
           4       0.99      0.98      0.99       568
           5       0.98      0.98      0.98       558
           6       0.99      0.99      0.99       558
           7       0.99      1.00      1.00       566
           8       0.99      0.99      0.99       554
           9       0.98      0.98      0.98       562

    accuracy                           0.99      5620
   macro avg       0.99      0.99      0.99      5620
weighted avg       0.99      0.99      0.99      5620
```



Average Loss: 0.045532274013338518
Average Accuracy: 98.61743795871735%


Loss across all folds
Training Loss
Validation Loss
Loss
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
Accuracy across all folds
Training Accuracy
Validation Accuracy
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0
10
20
30
40
50