Project Report: Hand Sign Classification Using Neural Networks

Introduction

This project aims to build and train a neural network model capable of classifying hand signs based on images. The model is trained on a dataset consisting of images of different hand signs, and it uses a multi-layer perceptron (MLP) architecture to learn and predict the corresponding classes.

Libraries and Dependencies

The following Python libraries were used in the project: - `numpy`: For numerical operations and array manipulation. - `os`: For interacting with the file system to load the dataset. - `matplotlib`: For plotting images and visualizing the loss curve. - `tensorflow`: For building and training the neural network model. - `sklearn`: For calculating the accuracy of the model.

```
import numpy as np
import os
import matplotlib.pyplot as plt
import matplotlib.image as img
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
```

Data Loading and Preparation

The dataset is organized into training and testing sets, each containing subfolders corresponding to different hand sign classes. The data is loaded into the model by reading image files, converting them into arrays, and flattening them into vectors.

Data Paths

```
TRAIN_DATA_PATH = 'd:\\Projects\\hand-sign-ml\\data\\train'
TEST_DATA_PATH = 'd:\\Projects\\hand-sign-ml\\data\\test'
```

Listing Classes

```
os.chdir(TRAIN_DATA_PATH)
letters  = os.listdir()
```

Data Seeding Function

The seed function is responsible for loading images, flattening them into 1D arrays (28x28 pixels), and associating each image with its corresponding label.

```
def seed(path):
    X_raw = []
    Y_raw = []

    for letter in letters:
        dir = path + "\\"+ letter
        letter_index = letters.index(letter)
        os.chdir(dir)
        files = os.listdir()

        for file in files:
            file_path = dir + "\\" + file
            X_raw.append(img.imread(file_path).reshape(28*28))
            Y_raw.append(letter_index)

    X = np.array(X_raw)
    Y = np.array(Y_raw).reshape(X.shape[0],1)

    return X,Y
```

Loading Data

```
X_train, Y_train = seed(TRAIN_DATA_PATH)
X_test, Y_test = seed(TEST_DATA_PATH)
```

Model Architecture

A Sequential neural network model is defined using TensorFlow and Keras. The model consists of six fully connected layers (Dense layers) with varying numbers of neurons and activation functions.

Model Definition

```
tf.random.set_seed(1234)
```

```
model = Sequential(
    [
        tf.keras.Input(shape=(784,)),
        Dense(units=640,activation="relu",name="L1"),
        Dense(units=360,activation="relu",name="L2"),
        Dense(units=180,activation="relu",name="L3"),
        Dense(units=96,activation="relu",name="L4"),
        Dense(units=56,activation="relu",name="L5"),
        Dense(units=24,activation="linear",name="L6")
    ], name = "hand_sign_model"
)
```

Model Summary

The model summary provides a detailed overview of each layer, including the output shape and the number of parameters.

```
model.summary()
```

Output:

```
Model: "hand_sign_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 L1 (Dense)                  (None, 640)               502400
 L2 (Dense)                  (None, 360)               230760
 L3 (Dense)                  (None, 180)               64980
 L4 (Dense)                  (None, 96)                17376
 L5 (Dense)                  (None, 56)                5432
 L6 (Dense)                  (None, 24)                1368
=================================================================
Total params: 822,316
Trainable params: 822,316
Non-trainable params: 0
_________________________________________________________________
```

Model Compilation and Training

The model is compiled with the following parameters: - **Loss Function**: `SparseCategoricalCrossentropy` - suitable for multi-class classification problems where labels are provided as integers. - **Optimizer**: `Adam` - a popular optimization algorithm that adapts the learning rate. - **Learning Rate**: `0.001`

Compilation

```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)
```

Training

The model is trained for 40 epochs using the training dataset.

```
history = model.fit(
    X_train,Y_train,
    epochs=40
)
```

Prediction and Evaluation

The model's performance is evaluated using the test dataset. The predicted index function computes the softmax probabilities and selects the class with the highest probability as the prediction.

Prediction Function

```
def predicted_index(x):
    prediction = model.predict(x.reshape(1,28*28))
    prediction_p = tf.nn.softmax(prediction)
    return np.argmax(prediction_p)
```

Generating Predictions

```
y_predicted = list(map(predicted_index, X_test))
```

Accuracy Calculation

The accuracy of the model is calculated using the `accuracy_score` function from `sklearn`.

```
accuracy =
        accuracy_score(Y_test.reshape(Y_test.shape[0]),y_predicted)
print(f'Model has accuracy of {100*accuracy:.2f}%')
```

Output:

```
Model has accuracy of 87.15%
```

Loss Visualization

The training loss is plotted against epochs to visualize the model’s learning process.

Plotting Loss

```
plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()
```


Loss Curve
Loss Curve

Conclusion

The neural network model was successfully trained to classify hand signs with an accuracy of approximately 87.15%. The architecture consists of multiple dense layers, each contributing to the model’s ability to learn complex patterns in the input images. The loss curve indicates the model’s learning over time, and the accuracy suggests a strong performance, though there may still be room for improvement.