Fractal-3 Assignment
MACHINE LEARNING

Problem 2: Learning to implement Neural Network

Mayur Mehta
M22AI577
29 April 2023

https://github.com/M22AI577/ML_Assigmnet_FACTORL3/blob/21162b688331d1898de5ffd9241edd4c798aa2d1/Preblem_2_ML_Gurumukhi_digit.ipynb

Learning to implement Neural Network

Gurmukhi Handwritten Digit Classification: Gurmukhi is one of the popular Indian scripts widely used in Indian state of Punjab. In this part of the assignment, our goal is to develop a neural network solution (a simple NN, not a CNN) for classifyingGurmukhi digits. We provide you Handwritten Gurmukhi digit dataset.

First we have to load data from folder, in folder data is of each and every digit stored as digit folder. First we load the training data from folder using load_data function.

```
def load_data(sourcePath = '',labelArray = [],imageArray = []) :

  # Loop over each folder from '0' to '9'
  for numberLabel in range(10):
    number_folder = os.path.join(sourcePath, str(numberLabel))
    # Loop over each image in the folder
    print("Loading",format(numberLabel))
    for file in os.listdir(number_folder):
      file_path = os.path.join(number_folder, file)
      if file_path.endswith(('.tiff','.bmp')):
        # read the image i graay scale and resize it to the imageSize size
        img = cv2.imread(file_path, 0)
        img = cv2.resize(img, imageSize)
        # Append the image and label to the lists
        imageArray.append(img)
        labelArray.append(numberLabel)

# Loading Traiing dataset
load_data(train_path,train_numberLabels,train_Images)

# Convert the lists to NumPy arrays
train_Images = np.array(train_Images)
train_numberLabels = np.array(train_numberLabels)

# Save the arrays in NumPy format
np.save('image_train.npy', train_Images)
np.save('label_train.npy', train_numberLabels)


image_train = np.load('image_train.npy')
label_train = np.load('label_train.npy')


# Loading Testing dataset
load_data(val_path,test_numberLabels,test_Images)


# Convert the lists to NumPy arrays
test_Images = np.array(test_Images)
```

We are storing data using nupmy in train_Images and train_numberLabels.

Same way we are loading data of test data using load_data function and storing it image_test and label_test . The images are loaded from two directories, 'train' and 'val', which contain subdirectories for each digit label.The images are resized to 32x32 pixels and converted to grayscale.

```
# Loading Testing dataset
load_data(val_path,test_numberLabels,test_Images)


# Convert the lists to NumPy arrays
test_Images = np.array(test_Images)
test_numberLabels = np.array(test_numberLabels)

# Save the arrays in NumPy format
np.save('image_test.npy', test_Images)
np.save('label_test.npy', test_numberLabels)


image_test = np.load('image_test.npy')
label_test = np.load('label_test.npy')


model = keras.Sequential([
keras.layers.Flatten(),
keras.layers.Dense(10, input_shape=(1024,),activation = 'sigmoid')
])


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



# train the model
# some 10 iterations done here
early_stopping_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history1 = model.fit(image_train, label_train,epochs= 10, validation_data=(image_test, label_test),callbacks=[early_stopping_callback])
# history1 = model.fit(image_train, label_train,epochs= 10, validation_data=(image_test, label_test))
plot_accuracy_loss(history1)
```

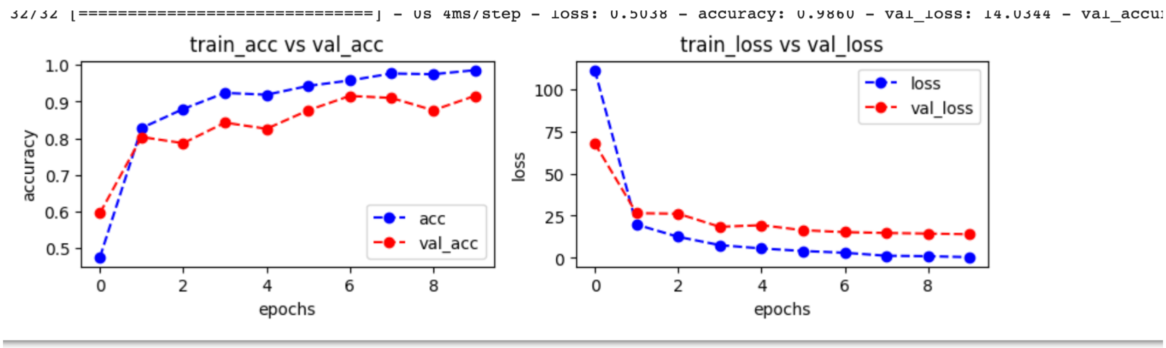
The model has a single hidden layer with 10 neurons and uses the sigmoid activation function. While training early stopping is implemented with a patience of 3 to prevent overfitting.

The model is trained using the 'adam' optimizer and 'sparse_categorical_crossentropy' loss function.

Output of model for 10 epochs :

```
Loading 0
Loading 1
Loading 2
Loading 3
Loading 4
Loading 5
Loading 6
Loading 7
Loading 8
Loading 9
Loading 0
Loading 1
Loading 2
Loading 3
Loading 4
Loading 5
Loading 6
Loading 7
Loading 8
Loading 9
Epoch 1/10
32/32 [==============================] - 1s 10ms/step - loss: 111.0027 - accuracy: 0.4750 - val_loss: 67.6944 - val_accuracy: 0.5955
Epoch 2/10
32/32 [==============================] - 0s 4ms/step - loss: 19.7782 - accuracy: 0.8280 - val_loss: 26.4992 - val_accuracy: 0.8034
Epoch 3/10
32/32 [==============================] - 0s 5ms/step - loss: 12.5170 - accuracy: 0.8800 - val_loss: 26.1839 - val_accuracy: 0.7865
Epoch 4/10
32/32 [==============================] - 0s 7ms/step - loss: 7.6201 - accuracy: 0.9240 - val_loss: 18.4447 - val_accuracy: 0.8427
Epoch 5/10
32/32 [==============================] - 0s 11ms/step - loss: 5.6340 - accuracy: 0.9190 - val_loss: 19.4063 - val_accuracy: 0.8258
Epoch 6/10
32/32 [==============================] - 0s 12ms/step - loss: 4.1458 - accuracy: 0.9430 - val_loss: 16.4220 - val_accuracy: 0.8764
Epoch 7/10
32/32 [==============================] - 0s 6ms/step - loss: 3.0977 - accuracy: 0.9580 - val_loss: 15.2940 - val_accuracy: 0.9157
Epoch 8/10
32/32 [==============================] - 0s 5ms/step - loss: 1.3288 - accuracy: 0.9770 - val_loss: 14.7724 - val_accuracy: 0.9101
Epoch 9/10
32/32 [==============================] - 0s 4ms/step - loss: 1.0534 - accuracy: 0.9750 - val_loss: 14.4335 - val_accuracy: 0.8764
Epoch 10/10
32/32 [==============================] - 0s 4ms/step - loss: 0.5038 - accuracy: 0.9860 - val_loss: 14.0344 - val_accuracy: 0.9157
```

As validation accuracy with 0.9157 . Training accuracy and validation accuracy both increase with each epoch, and they are relatively close to each other, indicating that the model is not overfitting the training data.



The training accuracy and validation accuracy are plotted in the upper-left subplot of the output figure. The blue line represents the training accuracy, and the red line represents the validation accuracy. Validation accuracy seems to plateau after around the 3rd epoch and 7th epoch, while the training accuracy continues to increase.This might indicate that the model is starting to overfit the training data after a certain number of epochs.

To improve accuracy and performance : we are adding more layer , apart from that we are scaling the dataset by 255.Increasing epochs value from 10 to 20 and Increased patience value to 5 for early stopping call backs.

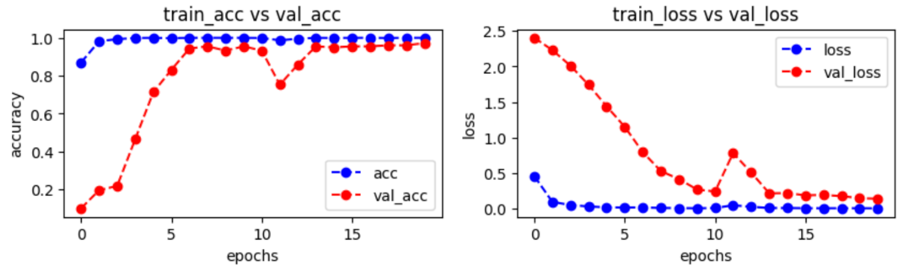
```
# Adding more layer in model
model = keras.Sequential([
    keras.layers.Flatten(),
    keras.layers.Dense(256, input_shape=(1024,), activation='sigmoid'),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(128, activation='sigmoid'),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(64, activation='sigmoid'),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation='sigmoid')
])

model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy']
 )

# scaling the data by dividing dataset by 255 ti improve accurancy
image_train_scaled = image_train/255
image_test_scaled = image_test/255
# Increased patiece value 5 and  epochs to 20 ffor eaarly stopping call backs
early_stopping_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
history2 = model.fit(image_train_scaled, label_train, epochs=20, validation_data=(image_test_scaled, label_test), callbacks=[early_stopping_callback])
```

By doing this there significant increase in validation accuracy 0.97 :

```
Epoch 1/20
32/32 [==============================] - 3s 19ms/step - loss: 0.4561 - accuracy: 0.8700 - val_loss: 2.4039 - val_accuracy: 0.1011
Epoch 2/20
32/32 [==============================] - 0s 11ms/step - loss: 0.0948 - accuracy: 0.9820 - val_loss: 2.2307 - val_accuracy: 0.1966
Epoch 3/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0496 - accuracy: 0.9930 - val_loss: 2.0113 - val_accuracy: 0.2191
Epoch 4/20
32/32 [==============================] - 0s 11ms/step - loss: 0.0302 - accuracy: 0.9990 - val_loss: 1.7430 - val_accuracy: 0.4663
Epoch 5/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0168 - accuracy: 1.0000 - val_loss: 1.4403 - val_accuracy: 0.7135
Epoch 6/20
32/32 [==============================] - 1s 17ms/step - loss: 0.0149 - accuracy: 0.9980 - val_loss: 1.1524 - val_accuracy: 0.8315
Epoch 7/20
32/32 [==============================] - 1s 33ms/step - loss: 0.0161 - accuracy: 1.0000 - val_loss: 0.7951 - val_accuracy: 0.9438
Epoch 8/20
32/32 [==============================] - 1s 19ms/step - loss: 0.0090 - accuracy: 1.0000 - val_loss: 0.5287 - val_accuracy: 0.9551
Epoch 9/20
32/32 [==============================] - 1s 17ms/step - loss: 0.0069 - accuracy: 1.0000 - val_loss: 0.4147 - val_accuracy: 0.9326
Epoch 10/20
32/32 [==============================] - 0s 13ms/step - loss: 0.0046 - accuracy: 1.0000 - val_loss: 0.2697 - val_accuracy: 0.9551
Epoch 11/20
32/32 [==============================] - 0s 11ms/step - loss: 0.0076 - accuracy: 0.9990 - val_loss: 0.2398 - val_accuracy: 0.9326
Epoch 12/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0468 - accuracy: 0.9870 - val_loss: 0.7804 - val_accuracy: 0.7528
Epoch 13/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0250 - accuracy: 0.9950 - val_loss: 0.5080 - val_accuracy: 0.8596
Epoch 14/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0082 - accuracy: 1.0000 - val_loss: 0.2150 - val_accuracy: 0.9551
Epoch 15/20
32/32 [==============================] - 0s 11ms/step - loss: 0.0081 - accuracy: 0.9990 - val_loss: 0.2181 - val_accuracy: 0.9494
Epoch 16/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 0.1871 - val_accuracy: 0.9551
Epoch 17/20
32/32 [==============================] - 0s 11ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 0.1923 - val_accuracy: 0.9551
Epoch 18/20
32/32 [==============================] - 0s 11ms/step - loss: 0.0048 - accuracy: 0.9990 - val_loss: 0.1756 - val_accuracy: 0.9607
Epoch 19/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0054 - accuracy: 1.0000 - val_loss: 0.1454 - val_accuracy: 0.9607
Epoch 20/20
32/32 [==============================] - 0s 10ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.1396 - val_accuracy: 0.9719
```

We can see training accuracy is 1 and validation accuracy has reached at 0.97, however after 10 epochs it started overfitting . Same there is drastic reduction on validation loss.

```
32/32 [==============================] - 0s 10ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.1396 - val_accuracy: 0.9719
```




```
6/6 [==============================] - 0s 4ms/step - loss: 0.1396 - accuracy: 0.9719
6/6 [==============================] - 0s 3ms/step
Predicted Value is  0
Predicted Value is  5
Predicted Value is  9
     0      5      10      15      20      25      30
```