Mid Term Project

Author: Jawahiir Nabhan

ID: 18-37550-1

Title: Implementing a CNN architecture to classify the MNIST handwritten data set.

Abstract: Recognizing a specific digit or alphabet when hand-written can be tough sometimes as the ways of writing can differ from person to person. The goal of this project is to create a convolutional neural network (CNN) that identifies the contents(Digits) in a hand-written MNSIT data set with an accuracy over 98\% using different optimizer.

1. Introduction:

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network, most commonly applied to analyze visual imagery. The MNIST data set is data set that contains images of hand written images which has 60000 training images and 10000 test images. In this project I have used 3 hidden layers in my model and 3 different optimizer to test which one works best for hand written data images. The optimizer are Adam, SGD and RMSProp.

```
model = keras.Sequential([
    keras.Input(shape= 784),
    layers.Dense(units=128, activation='relu'),
    layers.Dense(units=64, activation='relu'),
    layers.Dense(units=128, activation='relu'),
    layers.Dense(units=10, activation='softmax')
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,346
Trainable params: 118,346
Non-trainable params: 0
```

Figure 1: Sequential Model

The total number of images that have been train in this model is 118,346.

2. Results:

Different optimizer works differently in sync to given data sets. To acquire an accuracy over 98\% in the model I built I have verified which optimizer works best which would be the SGD optimizer.

The Adam optimizer gives an accuracy of 99.31% and a loss of 2.04%

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

```
h = model.fit(x=x_train, y=y_train, epochs= 10, batch_size=64, validation_split=0.3)
```

```
Epoch 1/10
657/657 [==============================] - 7s 11ms/step - loss: 0.3100 - accuracy: 0.9070 - val_loss: 0.1801 - val_accuracy: 0.9466
Epoch 2/10
657/657 [==============================] - 7s 11ms/step - loss: 0.1221 - accuracy: 0.9623 - val_loss: 0.1181 - val_accuracy: 0.9642
Epoch 3/10
657/657 [==============================] - 7s 10ms/step - loss: 0.0825 - accuracy: 0.9745 - val_loss: 0.1146 - val_accuracy: 0.9661
Epoch 4/10
657/657 [==============================] - 7s 11ms/step - loss: 0.0623 - accuracy: 0.9806 - val_loss: 0.1144 - val_accuracy: 0.9666
Epoch 5/10
657/657 [==============================] - 6s 10ms/step - loss: 0.0501 - accuracy: 0.9840 - val_loss: 0.1117 - val_accuracy: 0.9704
Epoch 6/10
657/657 [==============================] - 6s 10ms/step - loss: 0.0417 - accuracy: 0.9867 - val_loss: 0.1166 - val_accuracy: 0.9684
Epoch 7/10
657/657 [==============================] - 7s 10ms/step - loss: 0.0316 - accuracy: 0.9895 - val_loss: 0.1249 - val_accuracy: 0.9692
Epoch 8/10
657/657 [==============================] - 6s 10ms/step - loss: 0.0285 - accuracy: 0.9904 - val_loss: 0.1280 - val_accuracy: 0.9678
Epoch 9/10
657/657 [==============================] - 7s 11ms/step - loss: 0.0228 - accuracy: 0.9923 - val_loss: 0.1253 - val_accuracy: 0.9702
Epoch 10/10
657/657 [==============================] - 7s 11ms/step - loss: 0.0204 - accuracy: 0.9931 - val_loss: 0.1170 - val_accuracy: 0.9724
```


Figure 2 : ADAM Optimizer

The RMSProm optimizer gives an accuracy of 99.84% and loss of 0.47%

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

```
h = model.fit(x=x_train, y=y_train, epochs= 10, batch_size=64, validation_split=0.3)
```

```
Epoch 1/10
657/657 [==============================] - 7s 11ms/step - loss: 0.0147 - accuracy: 0.9952 - val_loss: 0.1403 - val_accuracy: 0.9739
Epoch 2/10
657/657 [==============================] - 6s 9ms/step - loss: 0.0110 - accuracy: 0.9961 - val_loss: 0.1462 - val_accuracy: 0.9749
Epoch 3/10
657/657 [==============================] - 7s 10ms/step - loss: 0.0088 - accuracy: 0.9969 - val_loss: 0.2181 - val_accuracy: 0.9664
Epoch 4/10
657/657 [==============================] - 7s 10ms/step - loss: 0.0086 - accuracy: 0.9971 - val_loss: 0.1811 - val_accuracy: 0.9738
Epoch 5/10
657/657 [==============================] - 8s 12ms/step - loss: 0.0068 - accuracy: 0.9975 - val_loss: 0.2040 - val_accuracy: 0.9732
Epoch 6/10
657/657 [==============================] - 7s 11ms/step - loss: 0.0062 - accuracy: 0.9980 - val_loss: 0.1855 - val_accuracy: 0.9757
Epoch 7/10
657/657 [==============================] - 8s 12ms/step - loss: 0.0062 - accuracy: 0.9981 - val_loss: 0.2062 - val_accuracy: 0.9747
Epoch 8/10
657/657 [==============================] - 7s 10ms/step - loss: 0.0060 - accuracy: 0.9981 - val_loss: 0.2160 - val_accuracy: 0.9746
Epoch 9/10
657/657 [==============================] - 7s 11ms/step - loss: 0.0056 - accuracy: 0.9982 - val_loss: 0.2416 - val_accuracy: 0.9716
Epoch 10/10
657/657 [==============================] - 8s 12ms/step - loss: 0.0047 - accuracy: 0.9984 - val_loss: 0.2403 - val_accuracy: 0.9737
```


Figure 3: RMSProp Optimizer

The SGD optimizer gives an accuracy of 100% and loss of 0.005%

```
model.compile(
    optimizer= tf.keras.optimizers.SGD(),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
h = model.fit(x=x_train, y=y_train, epochs= 10, batch_size=64, validation_split=0.3)
```

```
Epoch 1/10
657/657 [==============================] - 6s 10ms/step - loss: 0.0014 - accuracy: 0.9996 - val_loss: 0.2176 - val_accuracy: 0.9756
Epoch 2/10
657/657 [==============================] - 6s 10ms/step - loss: 2.1609e-04 - accuracy: 0.9999 - val_loss: 0.2163 - val_accuracy: 0.9762
Epoch 3/10
657/657 [==============================] - 6s 10ms/step - loss: 1.1466e-04 - accuracy: 1.0000 - val_loss: 0.2157 - val_accuracy: 0.9762
Epoch 4/10
657/657 [==============================] - 6s 10ms/step - loss: 9.4106e-05 - accuracy: 1.0000 - val_loss: 0.2153 - val_accuracy: 0.9762
Epoch 5/10
657/657 [==============================] - 6s 9ms/step - loss: 8.2922e-05 - accuracy: 1.0000 - val_loss: 0.2150 - val_accuracy: 0.9761
Epoch 6/10
657/657 [==============================] - 7s 11ms/step - loss: 7.5113e-05 - accuracy: 1.0000 - val_loss: 0.2148 - val_accuracy: 0.9759
Epoch 7/10
657/657 [==============================] - 7s 10ms/step - loss: 6.9229e-05 - accuracy: 1.0000 - val_loss: 0.2146 - val_accuracy: 0.9761
Epoch 8/10
657/657 [==============================] - 7s 11ms/step - loss: 6.4537e-05 - accuracy: 1.0000 - val_loss: 0.2144 - val_accuracy: 0.9762
Epoch 9/10
657/657 [==============================] - 7s 10ms/step - loss: 6.0626e-05 - accuracy: 1.0000 - val_loss: 0.2142 - val_accuracy: 0.9762
Epoch 10/10
657/657 [==============================] - 6s 10ms/step - loss: 5.7297e-05 - accuracy: 1.0000 - val_loss: 0.2141 - val_accuracy: 0.9763
```


Figure 4: SGD Optimizer

Among the 3 optimiser the SGD worked best for this model.

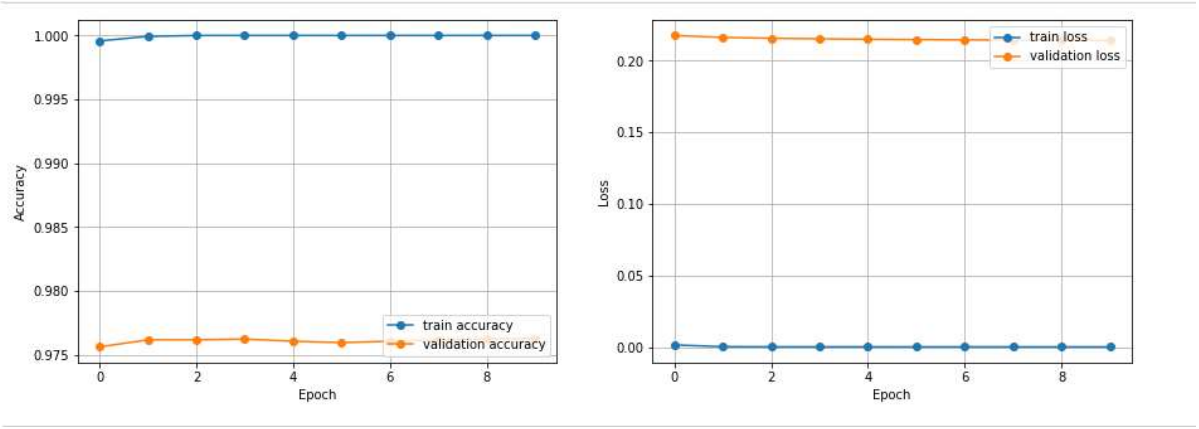

Figure 5: Plot for SGD Optimizer loss and accuracy

3. Discussion:

During my work to create the model at first using 2 hidden layers in the model did not seem to give me an accuracy in training over 94.00%. So I raised the number of hidden layer within the layer and also used a decent amount of units to train my model. Since this model was created to only detect digits in a MNIST data set, I hypothesize that 10 different output was easy for the model to make predictions. But if I were to take a data set of alphabets and create a model that detects 26 different labels that would decrease the accuracy of the model.