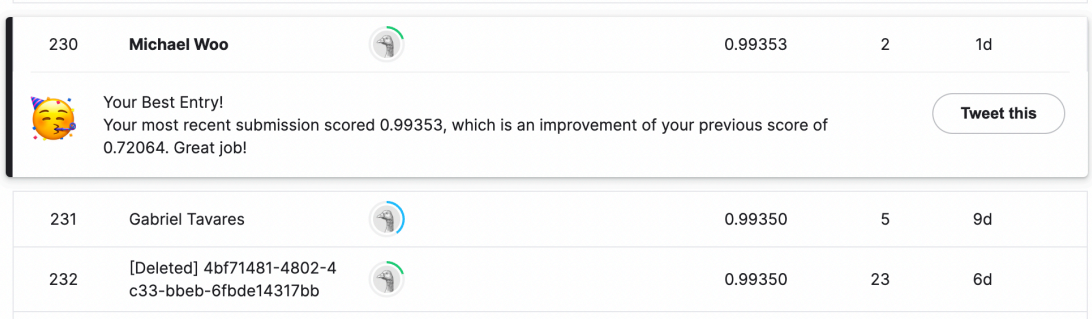
Digit_Recog_DNN

February 27, 2022

1 Final Scoring in the Competition and Ranking!

Submission

Leaderboard



Summary of Improvements * Since we are dealing with image data, I would think it would be best to use a few *convolutional* and *pooling* layers for this case * Added a *Dropout layer*: To prevent overfitting * Adjust *Neurons* per layer * Added *Learning Rate Scheduling*: Piecewise Constant Scheduling * Added *Batch Normalization*: To standardizes the inputs and stabilize the learning process * Used activation function (*Elu*) and initialization of weights (*lecun_normal*) * Added more layers and neurons * Used a different type of optimizer (*Adam*) * Added a *gradient clipping*: To prevent exploding gradients

Imports

[5]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import Normalizer
```

Dataset

[6]:
```
digit = pd.read_csv("data/train.csv")
digit_test = pd.read_csv("data/test.csv")
```

Training Dataset View

[7]:
```
digit.head()
```

[7]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	\
0	1	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	

	pixel8	…	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	\
0	0	…	0	0	0	0	0	0	
1	0	…	0	0	0	0	0	0	
2	0	…	0	0	0	0	0	0	
3	0	…	0	0	0	0	0	0	
4	0	…	0	0	0	0	0	0	

	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

[5 rows x 785 columns]

Test Dataset View

[8]:
```
digit_test.head()
```

[8]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	\
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	

	pixel9	…	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	\
0	0	…	0	0	0	0	0	0	
1	0	…	0	0	0	0	0	0	

```
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 784 columns]
```

Get only values

[9]:
```
x = digit.drop('label', axis = 1).values
y = digit.label.values
```

[10]:
```
x.shape
```

[10]:
```
(42000, 784)
```

[11]:
```
y.shape
```

[11]:
```
(42000,)
```

Re-scaling

[12]:
```
x = x/255.0
test_df = digit_test.values/255.0
```

Reshaping

[14]:
```
x_train = x.reshape(x.shape[0], 28, 28,1)
test_df = test_df.reshape(digit_test.shape[0], 28, 28,1)
y_train = y.reshape(-1,1)
```

Splitting Data

[15]:
```
x_train, x_valid, y_train, y_valid = train_test_split(x_train,y_train,test_size␣
↪= 0.20,
                                                      random_state = 141)
```

Clear the Backend

[18]:
```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

Piecewise Constant Scheduling * Added an additional learning constant * A dynamic learning process

[19]:
```
def piecewise_constant_fn(epoch):
    if epoch < 5:
        return 0.01
    elif epoch < 10:
        return 0.005
    elif epoch < 15:
        return 0.003
    elif epoch < 20:
        return 0.001
    else:
        return 0.0001
```

[20]:
```
def piecewise_constant(boundaries, values):
    boundaries = np.array([0] + boundaries)
    values = np.array(values)
    def piecewise_constant_fn(epoch):
        return values[np.argmax(boundaries > epoch) - 1]
    return piecewise_constant_fn
piecewise_constant_fn = piecewise_constant([5,10,15,20,25], [0.01, 0.005,0.
 ↪003,0.001,0.0001])
```

[21]:
```
lr_scheduler = keras.callbacks.LearningRateScheduler(piecewise_constant_fn)
```

Model Creation

[36]:
```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(filters=64, kernel_size=(5, 5),padding="valid" ,
                              ␣
 ↪activation='elu',kernel_initializer="lecun_normal", input_shape=(28,28,1)))
model.add(keras.layers.AveragePooling2D())
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(filters=32, kernel_size=(5, 5),padding="valid",
                              ␣
 ↪activation='elu',kernel_initializer="lecun_normal"))
model.add(keras.layers.AveragePooling2D())
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.GaussianDropout(0.25))
model.add(keras.layers.Flatten())
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dense(units=128,␣
 ↪activation='selu',kernel_initializer="lecun_normal"))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dense(units=64,␣
 ↪activation='selu',kernel_initializer="lecun_normal"))
model.add(keras.layers.Dense(units=10, activation = 'softmax'))
```

[37]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 24, 24, 64)        1664

 average_pooling2d_4 (Averag  (None, 12, 12, 64)       0
 ePooling2D)

 batch_normalization_14 (Bat  (None, 12, 12, 64)       256
 chNormalization)

 conv2d_7 (Conv2D)           (None, 8, 8, 32)          51232

 average_pooling2d_5 (Averag  (None, 4, 4, 32)         0
 ePooling2D)

 batch_normalization_15 (Bat  (None, 4, 4, 32)         128
 chNormalization)

 gaussian_dropout_3 (Gaussia  (None, 4, 4, 32)         0
 nDropout)

 flatten_3 (Flatten)         (None, 512)               0

 batch_normalization_16 (Bat  (None, 512)              2048
 chNormalization)

 dense_13 (Dense)            (None, 128)               65664

 batch_normalization_17 (Bat  (None, 128)              512
 chNormalization)

 dense_14 (Dense)            (None, 64)                8256

 dense_15 (Dense)            (None, 10)                650

=================================================================
Total params: 130,410
Trainable params: 128,938
Non-trainable params: 1,472
_________________________________________________________________
```

Model Compile

[38]:
```
model.compile(loss="sparse_categorical_crossentropy",optimizer=keras.optimizers.
↪Adam(clipvalue=1.0,clipnorm=1.0),metrics=["accuracy"])
```

Model Fitting

[39]:
```
history = model.
↪fit(x_train,y_train,epochs=25,validation_data=(x_valid,y_valid),callbacks=[lr_scheduler])
```

```
Epoch 1/25

2022-02-26 15:25:47.913170: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

1049/1050 [============================>.] - ETA: 0s - loss: 0.2338 - accuracy:
0.9304

2022-02-26 15:26:21.398391: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

1050/1050 [==============================] - 37s 34ms/step - loss: 0.2337 -
accuracy: 0.9304 - val_loss: 0.0989 - val_accuracy: 0.9731 - lr: 0.0100
Epoch 2/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.1190 -
accuracy: 0.9651 - val_loss: 0.0793 - val_accuracy: 0.9771 - lr: 0.0100
Epoch 3/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.1025 -
accuracy: 0.9709 - val_loss: 0.2088 - val_accuracy: 0.9464 - lr: 0.0100
Epoch 4/25
1050/1050 [==============================] - 34s 33ms/step - loss: 0.0933 -
accuracy: 0.9731 - val_loss: 0.0791 - val_accuracy: 0.9789 - lr: 0.0100
Epoch 5/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0853 -
accuracy: 0.9770 - val_loss: 0.0539 - val_accuracy: 0.9880 - lr: 0.0100
Epoch 6/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0467 -
accuracy: 0.9856 - val_loss: 0.0412 - val_accuracy: 0.9890 - lr: 0.0050
Epoch 7/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0426 -
accuracy: 0.9866 - val_loss: 0.0419 - val_accuracy: 0.9880 - lr: 0.0050
Epoch 8/25
1050/1050 [==============================] - 33s 31ms/step - loss: 0.0438 -
accuracy: 0.9861 - val_loss: 0.0466 - val_accuracy: 0.9864 - lr: 0.0050
Epoch 9/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0375 -
accuracy: 0.9884 - val_loss: 0.0426 - val_accuracy: 0.9888 - lr: 0.0050
Epoch 10/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0390 -
accuracy: 0.9884 - val_loss: 0.0393 - val_accuracy: 0.9905 - lr: 0.0050
```

```
Epoch 11/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0270 -
accuracy: 0.9917 - val_loss: 0.0301 - val_accuracy: 0.9917 - lr: 0.0030
Epoch 12/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0230 -
accuracy: 0.9930 - val_loss: 0.0342 - val_accuracy: 0.9918 - lr: 0.0030
Epoch 13/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0235 -
accuracy: 0.9927 - val_loss: 0.0342 - val_accuracy: 0.9921 - lr: 0.0030
Epoch 14/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0226 -
accuracy: 0.9927 - val_loss: 0.0363 - val_accuracy: 0.9911 - lr: 0.0030
Epoch 15/25
1050/1050 [==============================] - 419s 399ms/step - loss: 0.0199 -
accuracy: 0.9932 - val_loss: 0.0356 - val_accuracy: 0.9901 - lr: 0.0030
Epoch 16/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0147 -
accuracy: 0.9953 - val_loss: 0.0307 - val_accuracy: 0.9918 - lr: 0.0010
Epoch 17/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0123 -
accuracy: 0.9958 - val_loss: 0.0342 - val_accuracy: 0.9906 - lr: 0.0010
Epoch 18/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0107 -
accuracy: 0.9963 - val_loss: 0.0331 - val_accuracy: 0.9915 - lr: 0.0010
Epoch 19/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0103 -
accuracy: 0.9965 - val_loss: 0.0335 - val_accuracy: 0.9921 - lr: 0.0010
Epoch 20/25
1050/1050 [==============================] - 55s 52ms/step - loss: 0.0103 -
accuracy: 0.9968 - val_loss: 0.0330 - val_accuracy: 0.9929 - lr: 0.0010
Epoch 21/25
1050/1050 [==============================] - 120s 114ms/step - loss: 0.0080 -
accuracy: 0.9975 - val_loss: 0.0306 - val_accuracy: 0.9925 - lr: 1.0000e-04
Epoch 22/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0087 -
accuracy: 0.9970 - val_loss: 0.0303 - val_accuracy: 0.9924 - lr: 1.0000e-04
Epoch 23/25
1050/1050 [==============================] - 34s 32ms/step - loss: 0.0078 -
accuracy: 0.9973 - val_loss: 0.0302 - val_accuracy: 0.9927 - lr: 1.0000e-04
Epoch 24/25
1050/1050 [==============================] - 33s 32ms/step - loss: 0.0078 -
accuracy: 0.9971 - val_loss: 0.0295 - val_accuracy: 0.9926 - lr: 1.0000e-04
Epoch 25/25
1050/1050 [==============================] - 123s 117ms/step - loss: 0.0078 -
accuracy: 0.9974 - val_loss: 0.0285 - val_accuracy: 0.9924 - lr: 1.0000e-04
```

Save the Model * I already saved this model to local and cloud

[112]:
```
#model.save("Digit_Recog_Mod_2.h5")
```

[113]:
```
ml = keras.models.load_model("Digit_Recog_Mod_2.h5")
```

Testing Model Against Test Data

[114]:
```
pred = ml.predict(test_df)
```

```
2022-02-26 17:12:07.516663: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

Get the index of the max value in the array

[117]:
```
prediction = [np.argmax(i) for i in pred]
```

[117]:
```
[2,
 0,
 9,
 0,
 3,
 7,
 0,
 3,
 0,
 3,
 5,
 7,
 4,
 0,
 4,
 3,
 3,
 1,
 9,
 0,
 9,
 1,
 1,
 5,
 7,
 4,
 2,
 7,
 4,
 7,
 7,
 5,
```

4,
2,
6,
2,
5,
5,
1,
6,
7,
7,
4,
9,
8,
7,
8,
2,
6,
7,
6,
8,
8,
3,
8,
2,
1,
2,
2,
0,
4,
1,
7,
0,
0,
0,
1,
9,
0,
1,
6,
5,
8,
8,
2,
8,
9,
9,
2,

3,
5,
4,
1,
0,
9,
2,
4,
3,
6,
7,
2,
0,
6,
6,
1,
4,
3,
9,
7,
4,
0,
9,
2,
0,
7,
3,
0,
5,
0,
8,
0,
0,
4,
7,
1,
7,
1,
1,
3,
3,
3,
7,
2,
8,
6,
3,

```
8,
7,
7,
4,
3,
5,
6,
0,
0,
0,
3,
1,
3,
6,
4,
3,
4,
5,
5,
8,
7,
7,
2,
8,
4,
3,
5,
6,
5,
3,
7,
5,
7,
8,
3,
0,
4,
5,
1,
2,
7,
6,
3,
0,
2,
7,
8,
```

```
6,
1,
3,
7,
4,
1,
2,
4,
8,
5,
2,
4,
9,
2,
1,
6,
0,
6,
1,
4,
9,
6,
0,
9,
7,
6,
9,
1,
9,
0,
9,
9,
0,
8,
4,
6,
2,
0,
9,
3,
6,
3,
2,
1,
6,
3,
4,
```

2,
3,
1,
2,
2,
0,
4,
6,
1,
0,
0,
4,
9,
1,
7,
3,
2,
3,
8,
6,
8,
6,
2,
8,
5,
5,
4,
8,
3,
5,
9,
7,
1,
3,
8,
4,
5,
1,
4,
5,
6,
3,
3,
5,
7,
0,
6,

8,
3,
1,
6,
0,
6,
3,
9,
5,
1,
5,
8,
4,
0,
9,
2,
0,
5,
3,
7,
1,
9,
9,
5,
7,
7,
9,
9,
6,
3,
0,
3,
3,
6,
9,
8,
2,
6,
3,
7,
1,
4,
5,
8,
5,
9,
0,

```
0,
3,
8,
4,
1,
8,
4,
1,
1,
9,
8,
4,
5,
1,
5,
3,
6,
3,
1,
3,
0,
9,
0,
0,
6,
0,
6,
3,
1,
8,
6,
0,
6,
5,
2,
2,
6,
7,
7,
2,
5,
8,
3,
9,
2,
7,
8,
```

```
6,
3,
8,
4,
2,
3,
8,
1,
6,
4,
8,
7,
9,
7,
6,
9,
5,
3,
7,
6,
5,
5,
4,
2,
6,
2,
1,
3,
7,
1,
7,
9,
9,
6,
1,
1,
1,
7,
3,
9,
7,
6,
1,
1,
1,
9,
3,
```

8,
5,
5,
0,
4,
1,
2,
3,
1,
1,
3,
5,
9,
6,
6,
5,
3,
1,
4,
7,
4,
7,
4,
8,
5,
2,
6,
1,
3,
9,
5,
0,
8,
4,
7,
4,
4,
4,
1,
5,
3,
9,
5,
7,
6,
9,
5,

```
9,
2,
3,
5,
6,
6,
7,
5,
0,
5,
1,
7,
4,
4,
1,
1,
4,
9,
5,
6,
0,
1,
3,
1,
0,
4,
8,
1,
2,
7,
9,
4,
8,
3,
7,
7,
4,
2,
4,
6,
7,
6,
3,
2,
0,
6,
5,
```

```
9,
4,
1,
8,
3,
3,
0,
2,
7,
6,
8,
7,
5,
3,
5,
7,
4,
3,
6,
9,
0,
7,
7,
1,
0,
1,
1,
7,
0,
5,
3,
8,
3,
5,
6,
5,
7,
3,
0,
2,
8,
2,
0,
3,
0,
9,
2,
```

```
1,
1,
3,
0,
5,
0,
0,
7,
5,
6,
2,
0,
3,
8,
1,
6,
5,
4,
1,
1,
4,
6,
5,
3,
6,
0,
4,
8,
2,
4,
2,
5,
1,
7,
6,
9,
1,
7,
3,
8,
0,
8,
8,
4,
5,
3,
6,
```

6,
6,
0,
3,
5,
1,
7,
1,
6,
2,
8,
5,
6,
4,
7,
4,
3,
3,
2,
4,
7,
0,
0,
9,
8,
5,
9,
4,
0,
8,
8,
3,
6,
2,
6,
1,
8,
6,
1,
4,
7,
7,
8,
3,
0,
9,
9,

```
6,
7,
7,
4,
8,
1,
8,
4,
8,
0,
2,
8,
2,
4,
3,
3,
7,
2,
3,
4,
0,
4,
8,
1,
3,
3,
6,
3,
9,
4,
3,
8,
7,
7,
2,
6,
0,
6,
9,
8,
8,
1,
3,
4,
6,
9,
9,
```

2,
6,
0,
1,
8,
4,
3,
9,
8,
8,
4,
0,
5,
0,
6,
0,
4,
4,
6,
5,
1,
8,
1,
5,
3,
6,
2,
3,
7,
8,
9,
3,
1,
0,
1,
0,
6,
4,
7,
5,
7,
1,
3,
2,
7,
7,
1,

```
5,
1,
5,
4,
4,
3,
4,
3,
9,
0,
7,
8,
6,
4,
9,
4,
4,
1,
4,
7,
1,
1,
8,
3,
0,
4,
0,
4,
0,
0,
5,
1,
8,
6,
5,
0,
1,
5,
3,
4,
6,
3,
1,
1,
6,
9,
8,
```

```
3,
5,
5,
4,
8,
8,
5,
0,
4,
0,
4,
3,
1,
6,
9,
9,
1,
1,
3,
3,
1,
4,
9,
6,
9,
1,
5,
4,
2,
3,
2,
4,
0,
9,
7,
4,
3,
0,
5,
0,
1,
9,
0,
4,
5,
2,
8,
```

```
0,
5,
9,
3,
9,
6,
1,
5,
5,
1,
9,
0,
8,
4,
6,
7,
2,
8,
5,
8,
9,
7,
7,
2,
8,
1,
3,
4,
5,
0,
4,
1,
4,
2,
3,
6,
9,
2,
3,
4,
5,
4,
2,
3,
3,
1,
1,
```

```
0,
1,
4,
9,
1,
1,
2,
7,
1,
5,
4,
9,
1,
7,
6,
0,
4,
2,
9,
4,
1,
1,
5,
3,
5,
7,
9,
7,
8,
3,
2,
7,
2,
0,
4,
7,
1,
6,
4,
6,
1,
5,
7,
3,
5,
9,
4,
```

```
7,
9,
6,
6,
3,
3,
2,
1,
4,
5,
3,
7,
7,
9,
5,
6,
2,
6,
1,
0,
9,
3,
2,
9,
2,
6,
7,
5,
2,
3,
2,
8,
3,
0,
2,
7,
9,
4,
0,
9,
5,
1,
8,
8,
5,
3,
2,
```

```
 9,
 6,
 7,
 0,
 8,
 0,
 7,
 4,
 5,
 8,
 7,
 9,
 7,
 7,
 0,
 5,
 3,
 2,
 1,
 9,
 0,
 6,
 8,
 3,
 6,
 2,
 2,
 9,
 …]
```

[102]:
```
img_id = list(range(1,28001))
img_id = np.array(img_id)
```

[104]:
```
img_id.shape
```

[104]: (28000,)

Final Dataframe of Predictions * Has to be in this format for the competition

[105]:
```
df = pd.DataFrame({"ImageId":img_id,"Label":prediction})
df.head()
```

[106]:
```
df.head()
```

[106]:

	ImageId	Label
0	1	2
1	2	0

```
2        3      9
3        4      0
4        5      3
```

[107]:
```
df.tail()
```

[107]:
```
       ImageId  Label
27995    27996      9
27996    27997      7
27997    27998      3
27998    27999      9
27999    28000      2
```

Download to csv format

[108]:
```
df.to_csv("Michael_Woo_Predictions_Digit_Recog_2.csv",index=False)
```