v00 - Forked from: Brutally simple CNN with Keras, 99% Courtesy: https://www.kaggle.com/eiffelwong1 (https://www.kaggle.com/eiffelwong1)

v03-04 trying some things from https://keras.io/examples/generative/vae/ (https://keras.io/examples/generative/vae/) I don't like the binary (no greyscale) image conversion, but interested in the convolutional front and back end...

In [ ]:

```
import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import tensorflow.keras as keras
import keras

#for data augmentation (to make more data from existing data by shifting, r
otating, scaling, etc.)
# Data augmentation is NOT used in this version of the compressor
from keras.preprocessing.image import ImageDataGenerator

#simple CNN model with Keras
from keras.models import Model, Sequential, load_model
from keras.layers import Convolution2D, MaxPooling2D, BatchNormalization,
Conv2DTranspose
from keras.layers import Dense, Flatten, Activation, Reshape

# For visualization
from matplotlib import pyplot
```

```
/kaggle/input/digit-recognizer/test.csv
/kaggle/input/digit-recognizer/sample_submission.csv
/kaggle/input/digit-recognizer/train.csv
```

```
Using TensorFlow backend.
```

In [2]:
```
#reading both files
data = pd.read_csv('/kaggle/input/digit-recognizer/train.csv')
val_data = pd.read_csv('/kaggle/input/digit-recognizer/test.csv')
```

In [3]:
```
NUM_CLASS = 10

#making one hot encoding for the label
label = data['label']
label_one_hot = np.zeros((label.size, NUM_CLASS))
for i in range(label.size):
    label_one_hot[i,label[i]] = 1
#remove the label column, so the remaining 784 columns can form a 28*28 pho
to
del data['label']

#changing data from DataFrame object to a numpy array, cause I know numpy b
etter :p
# MADE THE DATA -127 to +128 instead of 0 to 255, to better match the resid
uals
# data = data.to_numpy()
data = data.to_numpy() - 127
print(data.shape)

#making data to 28*28 photo
data = data.reshape(-1,28,28,1)
```

```
(42000, 784)
```

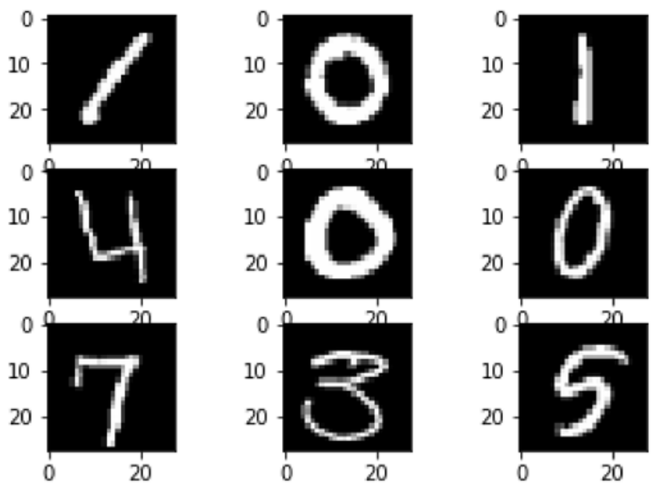
In [4]:

```
#checking out data shape
print(' data shape: {} \n one hot lable shape: {}'.format(
    data.shape, label_one_hot.shape))
print(' data minimum: {} \n data maximim: {}'.format(
    data.min(), data.max()))
```

```
 data shape: (42000, 28, 28, 1)
 one hot lable shape: (42000, 10)
 data minimum: -127
 data maximim: 128
```

In [5]:

```
# generate samples and plot
for i in range(9):
        # define subplot
        pyplot.subplot(330 + 1 + i)
        # convert for viewing
        image = data[i][:,:,0]
        # plot raw pixel data
        pyplot.imshow(image, cmap="gray")
# show the figure
pyplot.show()
```



The Nth Autoencoder

In [6]:

```
# The Nth Autoencoder network model compresses the entire 28x28 pixel image
down to a few floating point numbers
num_autoencoders = 3
# The number of floating point numbers images are compressed down to
compression = [1, 10, 100]
# number of training epochs per netowrk
num_epochs = 5


# the autoencoder models
model = []
# The front half (coder) models
c_model = []
# The decompressed approximations (decoded)
data_decomp = []
# The residual (difference between input and compressed appriximation)
data_res = []

for i in range (num_autoencoders) :
    this_compression = compression[i]
    model.append (Sequential([
        Convolution2D(filters = compression[i] * 4, kernel_size = (3,3),
activation = 'relu', strides=2, padding="same", input_shape=(28,28,1), na
me='A'),
        Convolution2D(filters = compression[i] * 4, kernel_size = (3,3),
activation = 'relu', strides=2, padding="same", name='B'),
        #Convolution2D(filters = compression[i] * 4, kernel_size = (3,3), a
ctivation = 'relu', strides=2, padding="same", name='C'),
        Flatten(name="flat"),
#        Dense(compression[i] * 2, activation='relu', name='D'),
        Dense(compression[i], activation='linear', name='code'),
        # decoding portion
        Dense(7 * 7 *  compression[i] * 4, activation='relu', name='D2'),
        Reshape((7, 7, compression[i] * 4), name='reshape'),
        #Conv2DTranspose(filters = compression[i] * 4, kernel_size = (3,3),
activation = 'relu', strides=2, padding="same", name='E'),
        Conv2DTranspose(filters = compression[i] * 4, kernel_size = (3,3
), activation = 'relu', strides=2,  padding="same", name='F'),
        Conv2DTranspose(compression[i] * 4, (3,3), activation="relu", str
ides=2, padding="same", name='G'),
        Conv2DTranspose(1, (3,3), activation="linear", padding="same", na
me='H')
        #  ---> End of Convolution

        #  ---> START Dense Only Compression (no knowledge that this is an
 image)
#        Flatten(input_shape=(28,28,1), name="flat"),
#        Dense(compression[i] * middle, activation='relu', name='middle1
b'),
#        Dense(compression[i] * middle, activation='relu', name='middle1
c'),
#        Dense(compression[i] * middle, activation='relu', name='middle1
d'),
#        # Code Generation Layer
#        Dense(compression[i], activation='linear', name='code'),
#        Dense(compression[i] * middle, activation='relu', name='middle2
b'),
#        Dense(compression[i] * middle, activation='relu', name='middle2
c'),
#        Dense(compression[i] * middle, activation='relu', name='middle2
d'),
#        Dense(784, activation='linear', name='decode'),
#        Reshape((28,28,1)),
        #  ---> END Dense Only
        ]))
    model[i].compile('adam',
              loss='mse',
              metrics=['mse']
             )

    # Diplay the model summary
    print("model",i,"summary")
    model[i].summary()
    print ("\n")

    # Train the to recreate original image or residual
    if (i == 0) :
        model[i].fit(data, data, epochs = num_epochs, validation_split =
0.1)
    else :
        model[i].fit(data_res[i-1], data_res[i-1], epochs = num_epochs, v
alidation_split = 0.1)
```

```
            .     'model_' + str






           .
                                     * 4               =  3 3
= 'relu'         =2         ="same"             = 28 28 1       ='A'
                              =                * 4              =  3 3
           = 'relu'          =2         ="same"     ='B'


                    ="flat"

                                        ='linear'     ='code'

























              .              'model_' + str           =True



    if    == 0
                   .               .
                .            _
                  =           .
    else
```

```
                              .                      .                        -1
                          .                        -1   -
                            =     .                                       .                          -1
                =  1
```

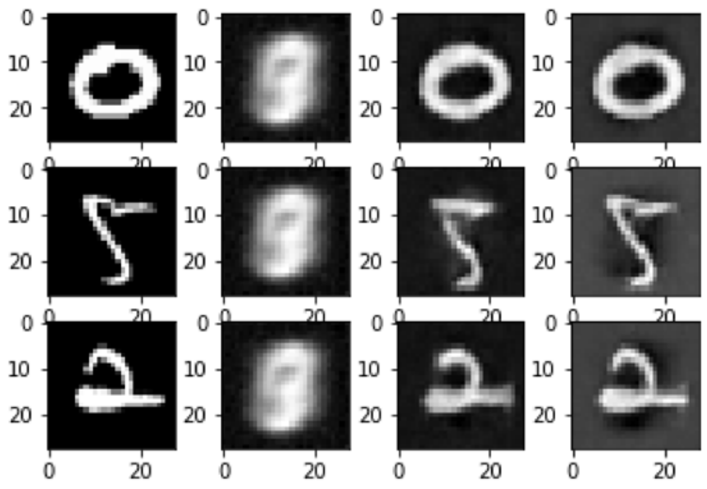
```
model 0 summary
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
A (Conv2D)                   (None, 14, 14, 4)         40        
_________________________________________________________________
B (Conv2D)                   (None, 7, 7, 4)           148       
_________________________________________________________________
flat (Flatten)               (None, 196)               0         
_________________________________________________________________
code (Dense)                 (None, 1)                 197       
_________________________________________________________________
D2 (Dense)                   (None, 196)               392       
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 4)           0         
_________________________________________________________________
F (Conv2DTranspose)          (None, 14, 14, 4)         148       
_________________________________________________________________
G (Conv2DTranspose)          (None, 28, 28, 4)         148       
_________________________________________________________________
H (Conv2DTranspose)          (None, 28, 28, 1)         37        
=================================================================
Total params: 1,110
Trainable params: 1,110
Non-trainable params: 0
_________________________________________________________________


Train on 37800 samples, validate on 4200 samples
Epoch 1/5
37800/37800 [==============================] - 11s 289us/step - loss: 
5168.2914 - mse: 5168.2910 - val_loss: 4414.3195 - val_mse: 4414.3184
Epoch 2/5
37800/37800 [==============================] - 7s 184us/step - loss: 4
366.8971 - mse: 4366.8950 - val_loss: 4382.0331 - val_mse: 4382.0332
Epoch 3/5
37800/37800 [==============================] - 7s 194us/step - loss: 4
349.6146 - mse: 4349.6138 - val_loss: 4371.0554 - val_mse: 4371.0547
Epoch 4/5
37800/37800 [==============================] - 7s 189us/step - loss: 4
```

```
343.5770 - mse: 4343.5757 - val_loss: 4367.2937 - val_mse: 4367.2944
Epoch 5/5
37800/37800 [==============================] - 7s 189us/step - loss: 4
340.2600 - mse: 4340.2598 - val_loss: 4363.6443 - val_mse: 4363.6440
model 1 summary
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
A (Conv2D)                   (None, 14, 14, 40)        400
_________________________________________________________________
B (Conv2D)                   (None, 7, 7, 40)          14440
_________________________________________________________________
flat (Flatten)               (None, 1960)              0
_________________________________________________________________
code (Dense)                 (None, 10)                19610
_________________________________________________________________
D2 (Dense)                   (None, 1960)              21560
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 40)          0
_________________________________________________________________
F (Conv2DTranspose)          (None, 14, 14, 40)        14440
_________________________________________________________________
G (Conv2DTranspose)          (None, 28, 28, 40)        14440
_________________________________________________________________
H (Conv2DTranspose)          (None, 28, 28, 1)         361
=================================================================
Total params: 85,251
Trainable params: 85,251
Non-trainable params: 0
_________________________________________________________________


Train on 37800 samples, validate on 4200 samples
Epoch 1/5
37800/37800 [==============================] - 7s 187us/step - loss: 1
512.3520 - mse: 1512.3525 - val_loss: 1155.6665 - val_mse: 1155.6663
Epoch 2/5
37800/37800 [==============================] - 7s 177us/step - loss: 1
091.2988 - mse: 1091.2992 - val_loss: 1057.1596 - val_mse: 1057.1595
Epoch 3/5
37800/37800 [==============================] - 6s 172us/step - loss: 1
```

```
011.4916 - mse: 1011.4913 - val_loss: 989.5101 - val_mse: 989.5099
Epoch 4/5
37800/37800 [==============================] - 7s 184us/step - loss: 9
70.7628 - mse: 970.7628 - val_loss: 960.0925 - val_mse: 960.0925
Epoch 5/5
37800/37800 [==============================] - 7s 175us/step - loss: 9
41.9848 - mse: 941.9842 - val_loss: 945.7160 - val_mse: 945.7158
model 2 summary
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
A (Conv2D)                   (None, 14, 14, 400)       4000      
_________________________________________________________________
B (Conv2D)                   (None, 7, 7, 400)         1440400   
_________________________________________________________________
flat (Flatten)               (None, 19600)             0         
_________________________________________________________________
code (Dense)                 (None, 100)               1960100   
_________________________________________________________________
D2 (Dense)                   (None, 19600)             1979600   
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 400)         0         
_________________________________________________________________
F (Conv2DTranspose)          (None, 14, 14, 400)       1440400   
_________________________________________________________________
G (Conv2DTranspose)          (None, 28, 28, 400)       1440400   
_________________________________________________________________
H (Conv2DTranspose)          (None, 28, 28, 1)         3601      
=================================================================
Total params: 8,268,501
Trainable params: 8,268,501
Non-trainable params: 0
_________________________________________________________________


Train on 37800 samples, validate on 4200 samples
Epoch 1/5
37800/37800 [==============================] - 34s 904us/step - loss:
550.1638 - mse: 550.1639 - val_loss: 321.5673 - val_mse: 321.5673
Epoch 2/5
37800/37800 [==============================] - 34s 888us/step - loss:
```

```
264.9019 - mse: 264.9019 - val_loss: 244.3658 - val_mse: 244.3658
Epoch 3/5
37800/37800 [==============================] - 33s 883us/step - loss:
210.8641 - mse: 210.8642 - val_loss: 218.7230 - val_mse: 218.7231
Epoch 4/5
37800/37800 [==============================] - 34s 893us/step - loss:
185.3393 - mse: 185.3395 - val_loss: 201.9874 - val_mse: 201.9873
Epoch 5/5
37800/37800 [==============================] - 34s 895us/step - loss:
168.3077 - mse: 168.3077 - val_loss: 193.5392 - val_mse: 193.5392
```

In [7]:
```
from random import random
# Let's see a hadful of data samples, showing the original image followed b
y the compressed image
# generate samples and plot
num_draw = 3
for j in range (num_draw) :
    pick = int(random() * data.shape[0])
    for i in range(num_autoencoders + 1):
        # convert to np array for viewing
        if (i == 0) :
            image = data[pick][:,:,0]
        else :
            for k in range(i) :
                if k==0 :
                    image = data_decomp[k][pick][:,:,0]
                else :
                    image += data_decomp[k][pick][:,:,0]
        # define subplot
        pyplot.subplot(num_draw, num_autoencoders + 1, i + j * (num_autoe
ncoders + 1) + 1)
        # plot raw pixel data
        pyplot.imshow(image, cmap='gray')
# show the figure
pyplot.show()
```



In [8]:
```
# Now, thanks to multiple layers of compression, the data is represented by a sequence of codes
data_code.shape
```

Out[8]:
```
(42000, 111)
```

How to learn the digits using only the coded data

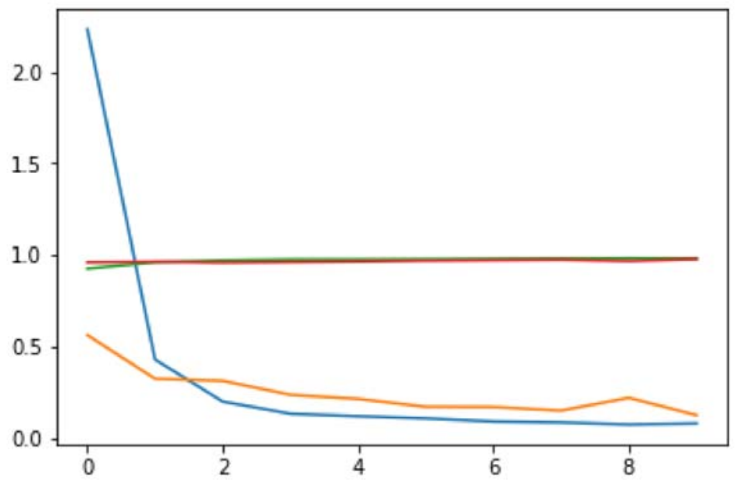
In [9]:
```
num_feat = int(data_code[0].shape[0])

Fmodel = Sequential([
    Dense(num_feat * 2, activation='relu', input_shape=(num_feat,)),
    Dense(num_feat * 2, activation='relu'),
    Dense(10),
    Activation('softmax')
          ])

Fmodel.compile('adam',
              loss='categorical_crossentropy',
              metrics=['accuracy']
             )

# Diplay the model summary
print("Final model summary")
Fmodel.summary()
print ("\n")

history = Fmodel.fit(data_code, label_one_hot, epochs = 10, validation_sp
lit = 0.1)
```

```
Final model summary
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 222)               24864     
_________________________________________________________________
dense_2 (Dense)              (None, 222)               49506     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2230      
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0         
=================================================================
Total params: 76,600
Trainable params: 76,600
Non-trainable params: 0
_________________________________________________________________


Train on 37800 samples, validate on 4200 samples
Epoch 1/10
37800/37800 [==============================] - 4s 103us/step - loss:
2.2322 - accuracy: 0.9257 - val_loss: 0.5619 - val_accuracy: 0.9598
Epoch 2/10
37800/37800 [==============================] - 4s 97us/step - loss: 0.
4282 - accuracy: 0.9602 - val_loss: 0.3230 - val_accuracy: 0.9631
Epoch 3/10
37800/37800 [==============================] - 4s 95us/step - loss: 0.
1979 - accuracy: 0.9711 - val_loss: 0.3113 - val_accuracy: 0.9583
Epoch 4/10
37800/37800 [==============================] - 4s 105us/step - loss:
0.1316 - accuracy: 0.9758 - val_loss: 0.2353 - val_accuracy: 0.9612
Epoch 5/10
37800/37800 [==============================] - 4s 98us/step - loss: 0.
1180 - accuracy: 0.9760 - val_loss: 0.2127 - val_accuracy: 0.9645
Epoch 6/10
37800/37800 [==============================] - 4s 93us/step - loss: 0.
1061 - accuracy: 0.9771 - val_loss: 0.1694 - val_accuracy: 0.9695
Epoch 7/10
37800/37800 [==============================] - 4s 98us/step - loss: 0.
0891 - accuracy: 0.9787 - val_loss: 0.1682 - val_accuracy: 0.9719
```

```
Epoch 8/10
37800/37800 [==============================] - 4s 93us/step - loss: 0.
0841 - accuracy: 0.9800 - val_loss: 0.1488 - val_accuracy: 0.9745
Epoch 9/10
37800/37800 [==============================] - 4s 93us/step - loss: 0.
0729 - accuracy: 0.9819 - val_loss: 0.2186 - val_accuracy: 0.9671
Epoch 10/10
37800/37800 [==============================] - 4s 105us/step - loss:
0.0787 - accuracy: 0.9805 - val_loss: 0.1237 - val_accuracy: 0.9779
```

In [10]:

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.show()
```



In [11]:

```
#we read the csv before, but just read it again here.
val_data = pd.read_csv('/kaggle/input/digit-recognizer/test.csv')

#the same way to process the training data after seperating the label
val_data = val_data.to_numpy() - 127 # shifted the data to more resemble t
he residuals, which are both positive and negative values
val_data = val_data.reshape(-1,28,28,1)

# Encode the data
# The decompressed approximations (decoded)
val_decomp = []
# The residual (difference between input and compressed appriximation)
val_res = []
for i in range (num_autoencoders) :
    this_compression = compression[i]
    # Let's calculate the decompressed estimation and the residual (first r
esidual minus decompressed approximation)
    if (i == 0) :
        val_decomp.append(model[i].predict(val_data))
        val_res.append(val_data - val_decomp[i])
        val_code = c_model[i].predict(val_data)
    else :
        val_decomp.append(model[i].predict(val_res[i-1]))
        val_res.append(val_res[i-1] - val_decomp[i])
        val_code = np.append(val_code, c_model[i].predict(val_res[i-1]),
axis = 1)



    #here we ask the model to predict what the class is
raw_result = Fmodel.predict(val_code)

#note: model.predict will return the confidence level for all 10 class,
#      therefore we want to pick the most confident one and return it as th
e final prediction
result = np.argmax(raw_result, axis = 1)

#generating the output, remember to submit the result to the competition af
terward for your final score.
submission = pd.DataFrame({'ImageId':range(1,len(val_data) + 1), 'Label':
np.argmax(raw_result, axis = 1)})
submission.to_csv('SimpleCnnSubmission.csv', index=False)
```