encoder

October 12, 2021

[ ]:
```
# Github: https:github.com/stevemats

import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import random
import cv2
import numpy as np

(x_train, y_train),(x_test, y_test) = tf.keras.datasets.mnist.load_data()  #␣
↪loading "mnist" training dataset

plt.imshow(x_train[0], cmap="gray")
```

[ ]:
```
plt.imshow(x_train[1], cmap="gray")
```

[ ]:
```
#compression so that data is 28*28px
x_train[0]
```

[ ]:
```
x_train[0].shape
```

[ ]:
```
28*28
# Below value is a result of the 28*28 px values equivalent to a total no. of␣
↪unique features
```

[ ]:
```
encoder_input = keras.Input(shape=(28, 28, 1), name='img') # Starts encoder
x = keras.layers.Flatten()(encoder_input) #flatten img so it can be used with␣
↪dense layers
encoder_output = keras.layers.Dense(64, activation="relu")(x) # compression␣
↪after flatten

encoder = keras.Model(encoder_input, encoder_output, name='encoder')


decoder_input = keras.layers.Dense(64, activation="relu")(encoder_output) #␣
↪starts decoder
decoder_output = keras.layers.Reshape((28, 28, 1))(x)
```

```
opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6) # setting an optimizer
#now combining encoder with decoder into a singular "autoencoder" model
autoencoder = keras.Model(encoder_input, decoder_output, name='autoencoder')
autoencoder.summary() #making sure theirs no errors
```

[ ]:
```
autoencoder.compile(opt, loss='mse') #compiling our model with the optimizer␣
↪and a loss metric
```

[34]:
```
# Training & saving the model each time
epochs=3

for epoch in range(epochs):

    history = autoencoder.fit(
      x_train,
      x_train,
      epochs=1,
      batch_size=32, validation_split=0.10
        )
    autoencoder.save(f"models/AE-{epoch+1}.model")
```

```
1688/1688 [==============================] - 1s 796us/step - loss: 0.0000e+00 -
val_loss: 0.0000e+00
INFO:tensorflow:Assets written to: models/AE-1.model\assets
1688/1688 [==============================] - 1s 821us/step - loss: 0.0000e+00 -
val_loss: 0.0000e+00
INFO:tensorflow:Assets written to: models/AE-2.model\assets
1688/1688 [==============================] - 2s 1ms/step - loss: 0.0000e+00 -
val_loss: 0.0000e+00
INFO:tensorflow:Assets written to: models/AE-3.model\assets
```

[ ]:
```
example = encoder.predict([ x_test[0].reshape(-1, 28, 28, 1) ])

print(example[0].shape)
```

[ ]:
```
print(example[0])
```

[ ]:
```
plt.imshow(example[0].reshape((8,8)), cmap="gray") # visualizing an 8*8 vector␣
↪of 64 values
```

[ ]:
```
plt.imshow(x_test[0], cmap="gray")
```

[ ]:
```
plt.imshow(example[0].reshape((8,8)), cmap="gray") # How the above looks after␣
↪going through our autoencoder
```

[ ]:
```
ae_out = autoencoder.predict([ x_test[0].reshape(-1, 28, 28, 1) ])
img = ae_out[0]  # predict is done on a vector, and returns a vector, even if␣
↪its just 1 element, so we still need to grab the 0th
```

```
plt.imshow(ae_out[0], cmap="gray")
```

[ ]:
```
plt.imshow(example[0].reshape((8,8)), cmap="gray")
```

[ ]:
```
# The idea  behind autoencoders is in data simplification
for d in x_test[:5]:

    ae_out = autoencoder.predict([ d.reshape(-1, 28, 28, 1) ])
    img = ae_out[0]

    cv2.imshow("decoded",img)
    cv2.imshow("original",np.array(d))
    cv2.waitKey(1000)  # wait 1000ms = 1 sec, and then show the next
```

[ ]:
```
smaller = cv2.resize(x_test[0], (8,8))
back_to_original = cv2.resize(smaller, (28,28))
plt.imshow(smaller, cmap="gray")
```

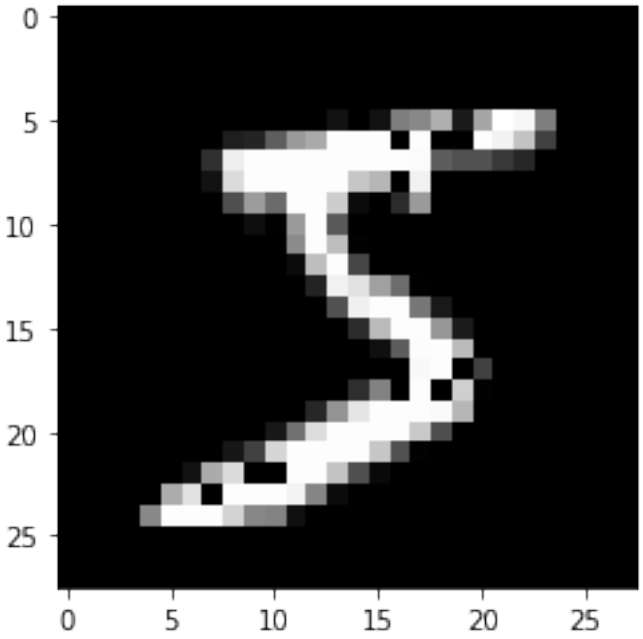
[ ]:
```
plt.imshow(back_to_original, cmap="gray")
```

[ ]:
```
# function to add noise
def add_noise(img, random_chance=5):
    noisy = []
    for row in img:
        new_row = []
        for pix in row:
            if random.choice(range(100)) <= random_chance:
                new_val = random.uniform(0, 1)
                new_row.append(new_val)
            else:
                new_row.append(pix)
        noisy.append(new_row)
    return np.array(noisy)
```

[26]:
```
noisy = add_noise(x_train[0])
```

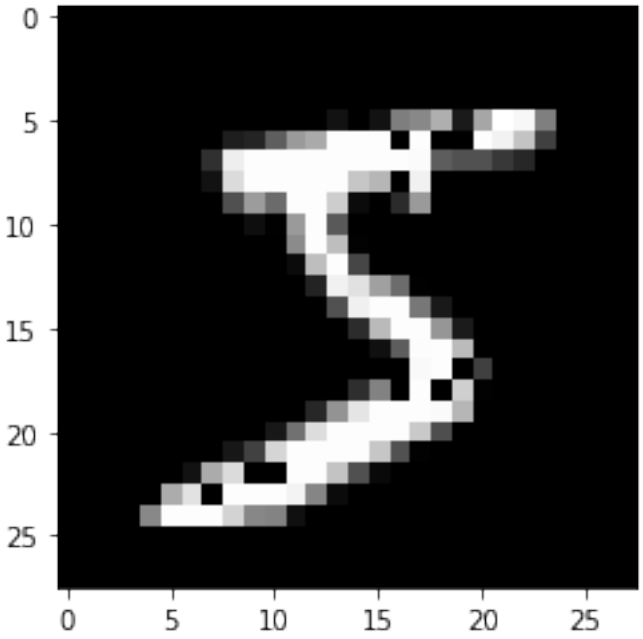
[27]:
```
plt.imshow(noisy, cmap="gray")
```

[27]:
```
<matplotlib.image.AxesImage at 0x17bddb90ca0>
```



[28]:
```
# The result above shows a very noisy 5
# let's try feed it to our autoencoder
ae_out = autoencoder.predict([ noisy.reshape(-1, 28, 28, 1) ])
img = ae_out[0]  # predict is done on a vector, and returns a vector, even if␣
 ↪its just 1 element, so we still need to grab the 0th
plt.imshow(ae_out[0], cmap="gray")
```

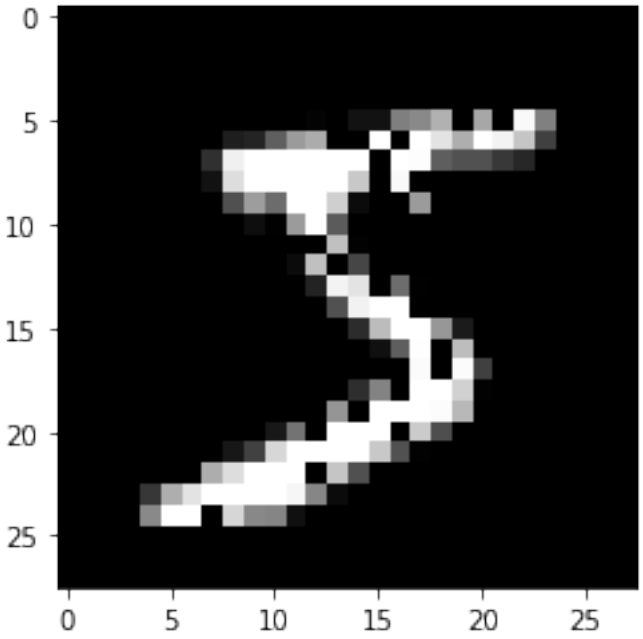
[28]:
```
<matplotlib.image.AxesImage at 0x17bddbf5b80>
```



[29]:
```
# results above show that noise has removed
# let's now try fill in the gaps with blw function
def remove_values(img, random_chance=5):
    noisy = []
    for row in img:
        new_row = []
        for pix in row:
            if random.choice(range(100)) <= random_chance:
                new_val = 0  # changing this to be 0
                new_row.append(new_val)
            else:
                new_row.append(pix)
        noisy.append(new_row)
    return np.array(noisy)
```

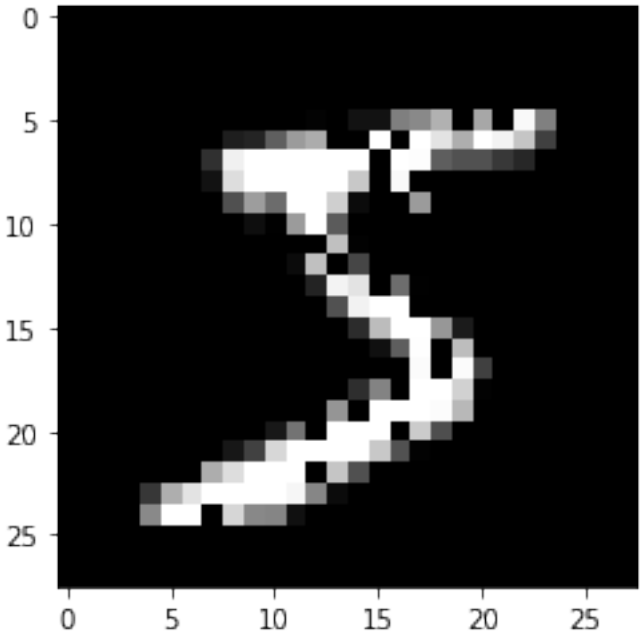
[30]:
```
some_hidden = remove_values(x_train[0], random_chance=15)  # slightly higher␣
↪chance so we see more impact
plt.imshow(some_hidden, cmap="gray")
```

[30]: <matplotlib.image.AxesImage at 0x17bddc24ac0>



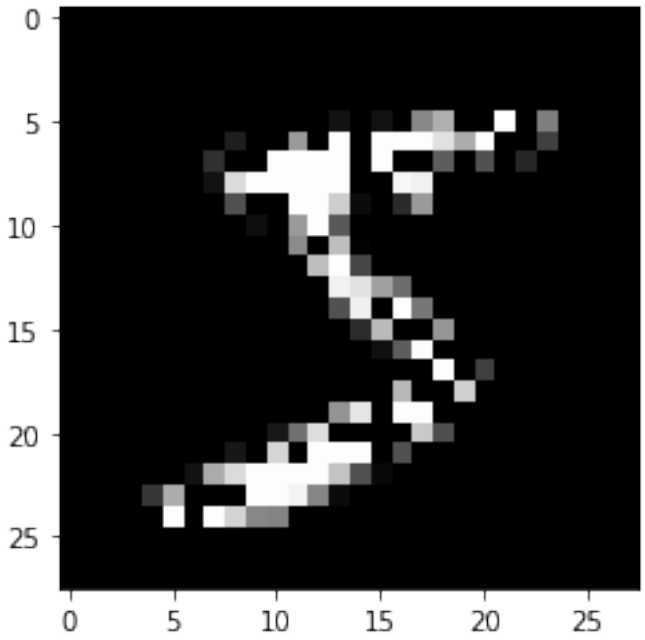
[31]:
```
ae_out = autoencoder.predict([ some_hidden.reshape(-1, 28, 28, 1) ])
img = ae_out[0]  # predict is done on a vector, and returns a vector, even if␣
↪its just 1 element, so we still need to grab the 0th
plt.imshow(ae_out[0], cmap="gray")
```

[31]:
```
<matplotlib.image.AxesImage at 0x17bddc9e880>
```



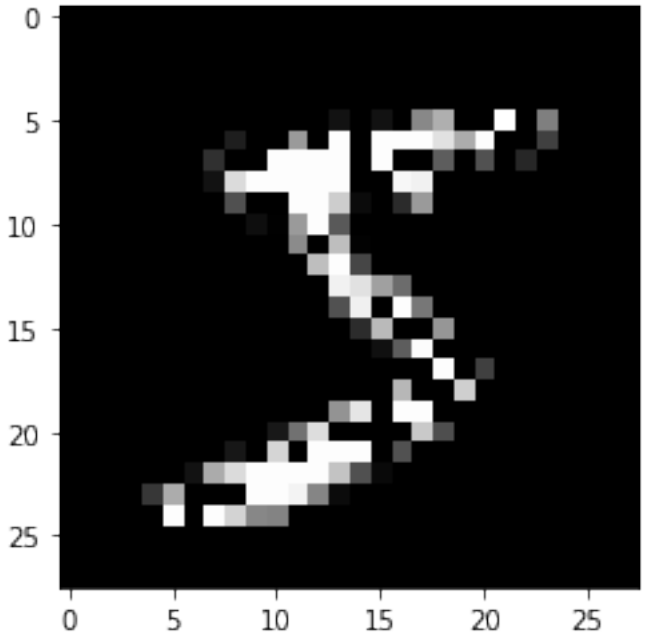
[32]:
```
some_hidden = remove_values(x_train[0], random_chance=35)  # slightly higher␣
↪chance so we see more impact
plt.imshow(some_hidden, cmap="gray")
```

[32]:
```
<matplotlib.image.AxesImage at 0x17bddce04c0>
```



[33]:
```
ae_out = autoencoder.predict([ some_hidden.reshape(-1, 28, 28, 1) ])
img = ae_out[0]  # predict is done on a vector, and returns a vector, even if␣
↪its just 1 element, so we still need to grab the 0th
plt.imshow(ae_out[0], cmap="gray")
```

[33]:
```
<matplotlib.image.AxesImage at 0x17bdc87ff10>
```


0
5
10
15
20
25
0
5
10
15
20
25

[ ]: