NOISE REDUCTION

__/__/____

IMAGE ~~DENOISING~~ WITH AUTOENCODERS

USING TENSOR-FLOW

Objective → Train a neural network to identify ~~images~~ numbers displayed in handwritten images (MNIST) Dataset. Same we used in Basic Image classification with tensorflow.

→ Add noise and denoise the data. So that we can teach our network what important things to focus on while denoising the data which will be given in real time.

TASK-1 Import ~~Libraries~~ Libraries

TASK-2 Data Preprocessing.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```
numpy arrays.

Values are in range of 0-255 so we will normalize them next.

```
x_train = x_train.astype('float') / 255
x_test = x_test.astype('float') / 255
```

Normalizing the data i.e getting them in range 0-1. This is not a good

approach and only valid when the data starts from 0 and we know an upper limit.

```
x_train = np.reshape(x_train, (60000, 784))
x_test = np.reshape(x_test, (10000, 784))
```

Reshaping from (6000, (28,28)) to (60000, 784)

original shape 28×28 = 784. This is done so that we can have 784 features in first input layer.

TASK-3 ADDING NOISE

Random values b/w 0 & 1

```
x_train_noisy = x_train + np.random.random(60000,784)*0.9
```

shape of random values → Scaling them down

Each value of x_train will have something added to them randomly. This will be the noise.

We multiplied with 0.9 to ensure values do not get too large

```
x_test_noisy = x_test + np.random.rand(10000, 784)*0.9
```

```
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
```

Input array → min value → max value.

np.clip will return the input array but where values are less than minimum it will be replaced by min value and those greater than max will be replaced by max value.

__/__/____

>>> a = np.arrange (10) [0,1,2,3,4,5,6,7,8,9]

>>> np.clip (a,1,8) → [1,1,2,3,4,5,6,7,8,8]

But it won't change actual a if not assigned to a

Eg → a = np.clip(a,1,8) → will change
np.clip(a,1,8) → won't change

In short it RETURNS the clipped array

>>> np.clip (a,3,6, out=a) [3,3,3,3,4,5,6,6,6,6]

This changes the original array without assignment

>>> a = [0,1,2,3,4,5,6,7,8,9]
>>> np.clip(a, [3,4,1,1,1,4,4,4,4,4], 8)

min value for each — max value

Output → [3,4,2,3,4,5,6,7,8,8]

def plot (x, p, labels = false): — x: array to plot; labels: default false, labels will be used later; p: prediction for each will be used later

plt.figure (figsize = (20,2)) → The total dimension of display panel not individual images.

show() displays the figure. You shouldn't call it until you have plotted things and want them to be displayed.

plt.imshow() draws an image on current figure.

plt.imshow just finishes drawing a picture instead of printing it. plt.show will print it.

→ display only first 10.

```
for i in range(10):
    plt.subplot(1,10,i+1)
```

1 row, 10 columns, we will start index from 1 as i from 0 is not allowed.

getting back to original

```
plt.imshow(x[i].reshape(28,28), shape
cmap = binary)
cmap
```

x ticks & y ticks are kind of ~~x label~~ values on x-axis like 0, 10, 20 etc

```
plt.xticks([])
plt.yticks([])
```
} Removing x ticks & y ticks

If predictions are there then display on x-axis

```
if labels:
    plt.xlabel(np.argmax(p[i]))
```

plt.show() → finally display plot.

plot(x_train, None)

TAS12-4 Building & Training a Classifier

We will train a classifier on non-noisy data and we will see how it does on noisy data.

Model of Keras in which we only need to input the layers as rows

```
classifier = Sequential([
```

First layer → `Dense(258, activation = 'relu', input_shape(784,)),`

- Dense layer
- no. of nodes
- activation function
- rectified linear unit
- Input nodes

Second layer → `Dense(256, activation = 'relu'),`

Output layer → `Dense(10, activation = 'softmax')`

- 10 because 10 classes in end i.e 1 to 10
- to get probability score

```
classifier.compile(
    optimizer = 'adam'
```

- loss function
- adaptive moment estimate

```
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)
```

no. of training examples utilized in one iteration

```
classifier.fit(X_train, Y_train, batch_size = 512, epochs = 3)
```

__/__/____

loss, accuracy = classifier.evaluate (x_test, y_test)

TASK-5 Building Autoencoder

Our noisy data gave accuracy of 20% (approx) which is not good.

So, we will built Auto encoder to denoise this data.

```
input_image = Input (shape = (784,))
encoded = Dense (64, activation = 'relu') (input_image)
```

Squeeze to less nodes to make it learn only the important ones

```
decoded = Dense (784, activation = 'sigmoid') (encoded)
```

Bring the dimensionality back. i.e denoised data.

creating the model

```
autoencoder = Model (input_image, decoded)
```

(input_image → input, decoded → output)

```
autoencoder.compile (loss = 'binary_crossentropy',
                     optimizer = 'adam')
```

__/__/____

TASK-6 Training the Auto Encoder.

```
autoencoder.fit( X_train_noisy, X_train, epochs=100,
                 batch_size=512, validation_split=0.2,
                 verbose=False,

                 callbacks=[
                    EarlyStopping(monitor='val_loss', patience=5),
                    LambdaCallback(----)
                 ]
)
```

TASK -> Denoised Images.

→ predictions = autoencoder.predict(X_test_noisy)

Denoised Images.

plot(X_test_noisy, None) → Noisy data

plot(predictions, None) → De noised images.

loss, accuracy = classifier.evaluate(predictions, y_test)

print(accuracy) → from 20% to 90% approx.

TASK - 8 Composite Model.

Combine classifier and encoder to first denoise then classify images. This will be a Composite model which completes the prediction pipeline.

So we send whether noisy or non noisy data the Model works.

input_image = Input (shape = (784,))

↳ x = autoencoder (input_image)

Denoised Image.

y = classifier (x)

denoise_and_classify = Model (input_image, y)

model name.

Here it is 1 to 10

predictions = denoise_and_classify.predict (x_test_noisy)

plot (x_test_noisy, predictions, True)

↳ These a help in comparison between expected and true output.

plot (x_test, to_categorical (y_test), True)

original test set.

one hot encoded values

Quiz

Point 1 → x = np.reshape (x, (10000, 784)) and x = x.reshape ((10000, 784)) are same things.

Point 2 → v = [0, 1, 1, 0, 5, 2, 1, 0, 2]

np.argmax (v) → It returns the index of max value in v

Point 3 → When classification problem eg → 10 classes, we can use these loss functions

RTO →

Loss function 1 → Sparse Categorical cross entropy when labels are not encoded

or

Loss function 2 → Categorical cross entropy after one-hot encoding labels.

Points →

→ Auto encoding can be used for lossy data compression where the compression is dependent on data itself.

→ Auto encoding is algo to help reduce dimensionality of data.

→ This algo to reduce dimensionality of data, as learned from data itself, can also be used for reducing noise in data.