TRAINING NEURAL NETWORK

ABOUT THE MNIST DATASET

MNIST dataset, a classic in the machine learning community, which has been around almost as long as the field itself and has been intensively studied. It's a set of 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s. The problem we're trying to solve here is to classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9).

TOPICS IN THIS ASSIGNMENT

1. Importing and understanding dataset
2. EDA
3. Preparing data
4. Building the model
5. Compiling and fitting the model
6. Prediction on test data
7. Evaluating the model
8. Rebuilding the model(repeating 3-7 steps)

How To Load Dataset?

- Documentation Link - https://www.tensorflow.org/api_docs/python/tf/keras/datasets
- Video link below

```
from IPython.display import YouTubeVideo
YouTubeVideo('Vm_wFo4j3So', width=600, height=300)
```

1. Importing & Understanding Data

```
# Import mnist from tensorflow.keras.datasets
```

```
# load the data using mnist.load_data and define train_images, train_labels, test_images,
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
# check the shape of train_images dataset
```

```
(60000, 28, 28)
```

```
# check the shape of train_images single image (train_images[0])
```

```
(28, 28)
```

```
# let's look at the first image which will show in the array form
```

```
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
 205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
  90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
 190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
 253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
 241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
 195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
  11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```

Observation from the array output

- You can see value in the array ranging from 0-255 depicting RGB color.

```
# check shape of train_labels
```

```
60000
```

```
# check the labels of train data
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
# check shape of test data
```

```
(10000, 28, 28)
```

```
# check the labels of test data
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

How to plot an image?

- You can find a simple way to plot an image of MNIST dataset by watching the video below.

```
YouTubeVideo('2JhLogJAUGc', width=600, height=300)
```

2. **EDA**

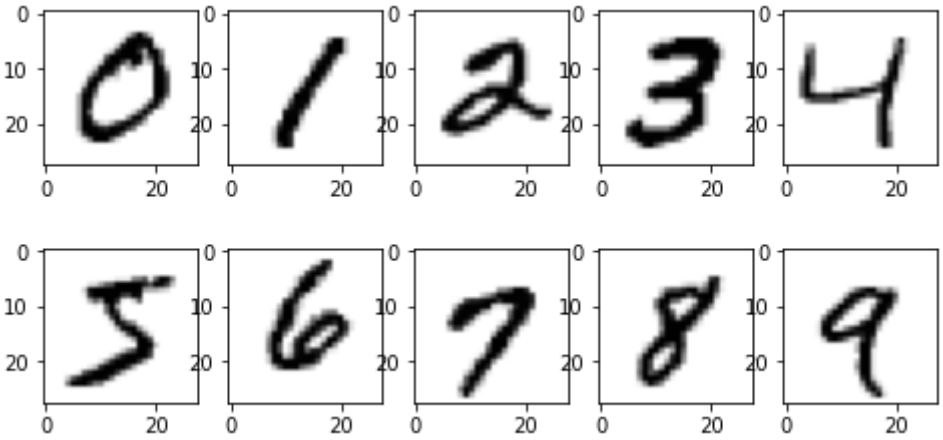
How to plot multiple images of the output(0-9 digits)?

- Follow the comments below in the code to plot.

```
# importing pyplot and numpy for plotting images of 0-9


# defining subplots (2,5)
fig, ax = plt.subplots(2,5, figsize=(8,4))

# looping over ax.flatten(), and plotting each digit
for i, ax in None: # code instead of None
    # choosing each digit occuring at its first instance
    im_idx = None # code instead of None
    # reshaping the selected digit to (28, 28) from (1, 28, 28)
    plottable_image = # code here
    # now pass this plottable_image to ax.imshow
    ax.imshow(plottable_image, cmap='gray_r')
```



3. **Preparing the data**

How to prepare the image for model building?

- Follow the comments below to understand the process.

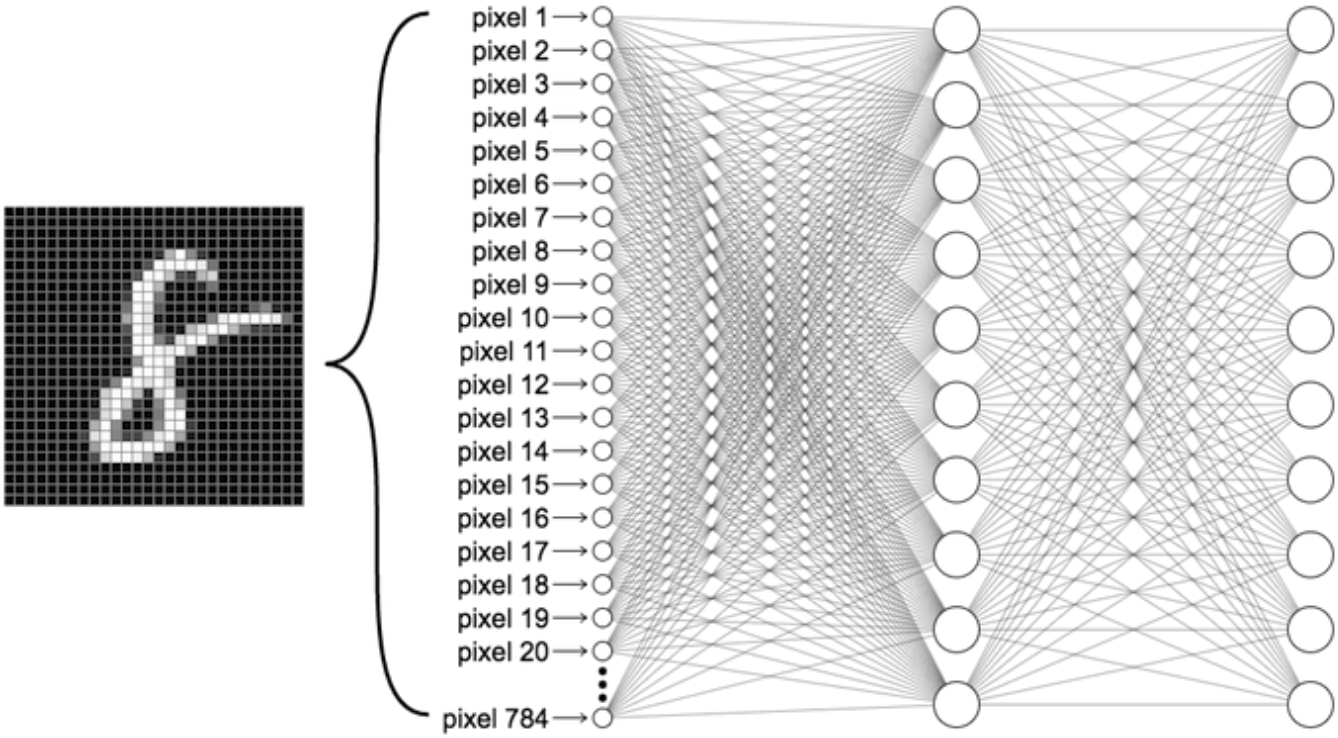
```
# reshape train_images from (60000, 28, 28) to (60000, 28*28)

# convert dtype of train_images from uint8 to float32

# reshape train_images from (10000, 28, 28) to (10000, 28*28)
```

```
# convert dtype of test_images from uint8 to float32
```

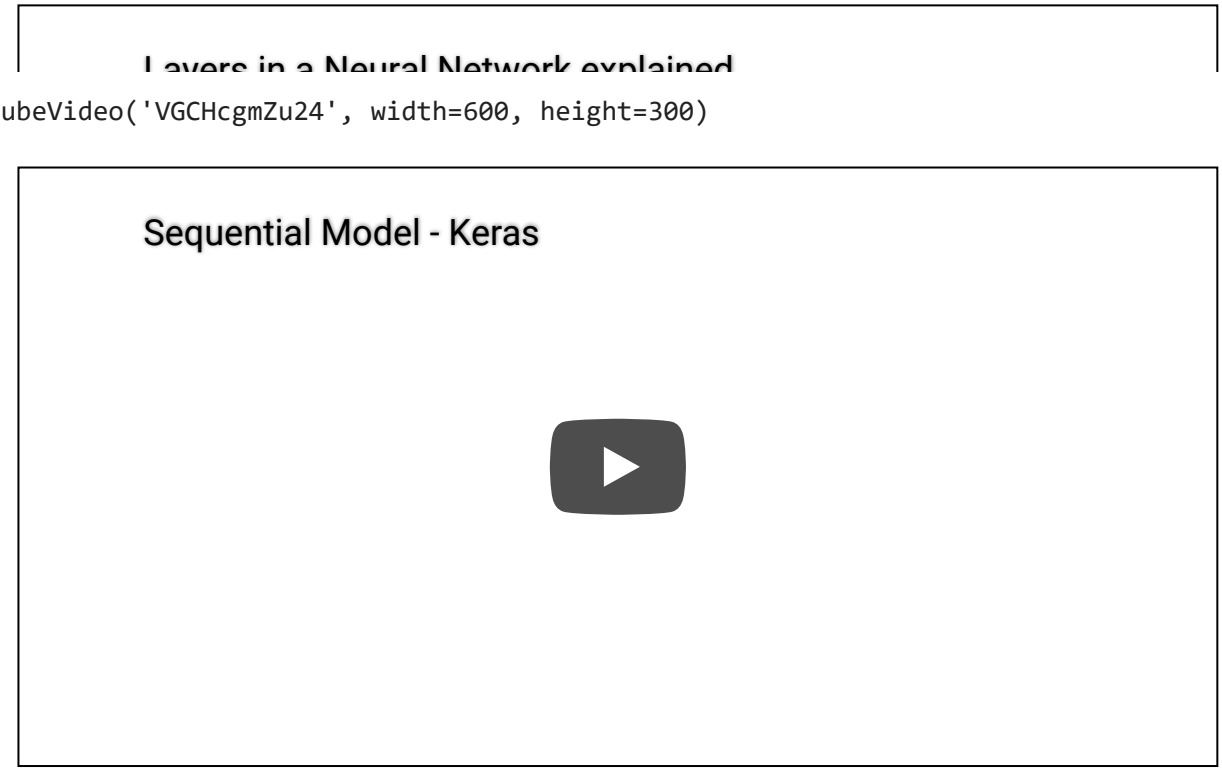
4.Model 1: Building the model



How To Build Model?

- Documentation Link - https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
- Watch the video below & follow the steps in code cells later.

```
YouTubeVideo('FK77zZxaBoI', width=600, height=300)
```




```
YouTubeVideo('VGCHcgmZu24', width=600, height=300)
```


Let's import necessary libraries and define our model by following the comments

```
# importing keras and layers from tensorflow


# define the model and its network architecture
# define two dense layers having first layer with 512 neurons & activation='relu'
# second layer with 10 neurons & activation = 'softmax'
model = #code here
```

5. **Compiling & fitting the model**

In the next cell, we will compile our model.

```
# compile the model with optimizer='rmsprop', loss='sparse_categorical_crossentropy', & me
```

Then we will fit our model

```
# fit the model with epochs=10, batch_size=128
```

```
Epoch 1/10
469/469 [==============================] - 6s 11ms/step - loss: 0.2545 - accuracy: 0
Epoch 2/10
469/469 [==============================] - 5s 10ms/step - loss: 0.1044 - accuracy: 0
```

```
Epoch 3/10
469/469 [==============================] - 5s 10ms/step - loss: 0.0689 - accuracy: 0
Epoch 4/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0503 - accuracy: 0
Epoch 5/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0374 - accuracy: 0
Epoch 6/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0282 - accuracy: 0
Epoch 7/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0218 - accuracy: 0
Epoch 8/10
469/469 [==============================] - 5s 10ms/step - loss: 0.0163 - accuracy: 0
Epoch 9/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0130 - accuracy: 0
Epoch 10/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0102 - accuracy: 0
<keras.callbacks.History at 0x7f9f4d3422d0>
```

6. **Prediction on test data**

In the next cell, we will take first 10 images of test data

```
# define a variable test_digits and store the first 10 images of test data
```

Then we will predict on those 10 images

```
# predict the test_digits using our model
```

We will check prediction on first image in next cell

```
# check the first image prediction from predictions
```

```
array([1.83151341e-10, 8.44416462e-15, 7.57867014e-09, 8.81661836e-05,
       3.46131464e-16, 1.52920732e-10, 4.08821713e-19, 9.99911666e-01,
       1.89882643e-09, 1.10814504e-07], dtype=float32)
```

Previous output shows probability of first image being either one of 0-9 digits. For example, probability of first image being 0 is 1.83151341e-10 which is very very low.

```
# checking the index having maximum prediction
```

```
7
```

In next cell, we can see the maximum prediction for first image is 0.99 at index 7 showing that first image is digit 7.

```
# checking the index value having maximum prediction
predictions[0][7]
```

```
0.99991167
```

In next cell, we are confirming whether our prediction is right or wrong by checking label.

```
# checking the label for that index having maximum prediction
test_labels[0]
```

```
7
```

7. **Evaluating the model**

Now we evaluate our model on unseen data, which is test set.

```
# check the loss and accuracy for test data using model.evaluate
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0681 - accuracy: 0.9
```

```
# print test accuracy
```

```
test_acc: 0.9825000166893005
```

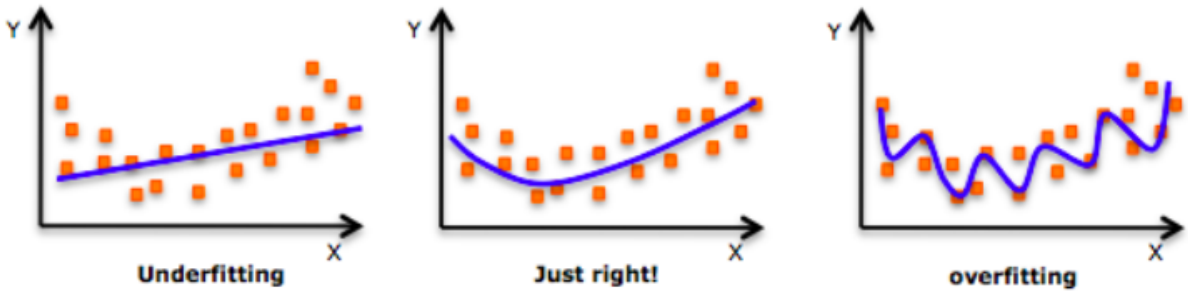
Observation from evaluation

- We can see that our train accuracy is 99.7% and our test accuracy is 98.2% which clearly shows the case of overfitting.

REGULARIZATION

Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. It is a way to overcome overfitting problem in machine learning/deep learning problem. This in turn improves the model's performance on the unseen data as well.

Overfittng: It is a situation where your model performed exceptionally well on train data but was not able to predict test data.



How do we deal with reglarization?

There are multiple ways to deal with regularization

- L2 & L1 regularization
- Dropouts
- Early stopping
- Data augmentation

Besides these ways, tweaking the architecture is a natural way to avoid overfitting. Here we just gave you an overview of regularization. We will cover this detail in another assignment.

Here we will just tweak our architecture to show you how you can avoid overfitting. This is an excellent way to learn how you can train the network with different architectures.

8. Model 2: Rebuilding another model

```
# build 2nd model
# architecture: total 3 dense layers,
# first 2 layers with 128, 128 neurons with activation='relu'
# and last layer with 10 neurons and activation='softmax'
model2 = # code here
```

```
# compiling model2 with optimizer='rmsprop', loss='sparse_categorical_crossentropy' and me
```

```
# fitting model 2 with epochs=6, batch_size=128
```

```
Epoch 1/6
469/469 [==============================] - 4s 7ms/step - loss: 0.2952 - accuracy: 0.9
Epoch 2/6
469/469 [==============================] - 3s 7ms/step - loss: 0.1240 - accuracy: 0.9
Epoch 3/6
469/469 [==============================] - 3s 6ms/step - loss: 0.0872 - accuracy: 0.9
Epoch 4/6
469/469 [==============================] - 3s 6ms/step - loss: 0.0655 - accuracy: 0.9
Epoch 5/6
469/469 [==============================] - 3s 6ms/step - loss: 0.0524 - accuracy: 0.9
Epoch 6/6
469/469 [==============================] - 3s 7ms/step - loss: 0.0417 - accuracy: 0.9
<keras.callbacks.History at 0x7f9f43adab10>
```

```
# check the loss and accuracy for test data using model2.evaluate
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0809 - accuracy: 0.9
```

Observation from evaluation

- We can see the gap between train accuracy & test accuracy falling down from 1.5% in our first model to 0.9% this time. Overfitting is still a problem so we will do some more changes in our architecture.

9. Model 3: Rebuilding another model

```
# Build 3rd model
# add 3 dense layers in which
# first 2 layers with 64 neurons and activation = 'relu'
# last layer with 10 neurons and activation='softmax'
model3 = # code here
```

```
# compiling model3 with optimizer='rmsprop', loss='sparse_categorical_crossentropy' and me
```

```
# fitting the model 3 with epochs = 5 and batch_size=128
```

```
Epoch 1/5
469/469 [==============================] - 4s 7ms/step - loss: 0.3770 - accuracy: 0.8
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.1767 - accuracy: 0.9
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1283 - accuracy: 0.9
```

```
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1028 - accuracy: 0.9
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0857 - accuracy: 0.9
<keras.callbacks.History at 0x7f9f43879e10>
```

```
# check the loss and accuracy for test data using model3.evaluate
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0981 - accuracy: 0.9
```

Observation from evaluation

- We can see the gap between train accuracy & test accuracy falling down from 0.9% in our 2nd model to 0.5% this time. Let's try to reduce this gap also.

10. **Model 4: Rebuilding another model**

```
# Build 4th model
# add 2 dense layers in which
# first layer with 128 neurons and activation = 'relu'
# last layer with 10 neurons and activation='softmax'
model4 = # code here
```

```
# compiling model4 with optimizer='rmsprop', loss='sparse_categorical_crossentropy' and me
```

```
# fitting the model 4th with epochs = 5 and batch_size=128
```

```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.3362 - accuracy: 0.9
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1581 - accuracy: 0.9
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1117 - accuracy: 0.9
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0862 - accuracy: 0.9
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.0692 - accuracy: 0.9
<keras.callbacks.History at 0x7f9f3da72dd0>
```

```
# check the loss and accuracy for test data using model4.evaluate
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0861 - accuracy: 0.9
```

Observation from evaluation

- We can see the gap between train accuracy & test accuracy falling down from 0.5% in our fourth model to 0.4% this time. You can also try different architecture and play with it to see how it works. Learn from such insights.

CONCLUSION FROM THIS ASSIGNMENT

- We saw that how model complexity i.e. no. of layers and neurons affect our model.
- Increasing no. of neurons/layers can make our model overfit the data.

IT'S TIME FOR FEEDBACK GUYS.....PLEASE FILL THE FORM AND HELP US TO IMPROVE

https://forms.zohopublic.in/cloudyml/form/CloudyMLDeepLearningFeedbackForm/formperma/VCFbldnXAnbcgAIl0lWv2blgHdSldheO4RfktMdgK7s