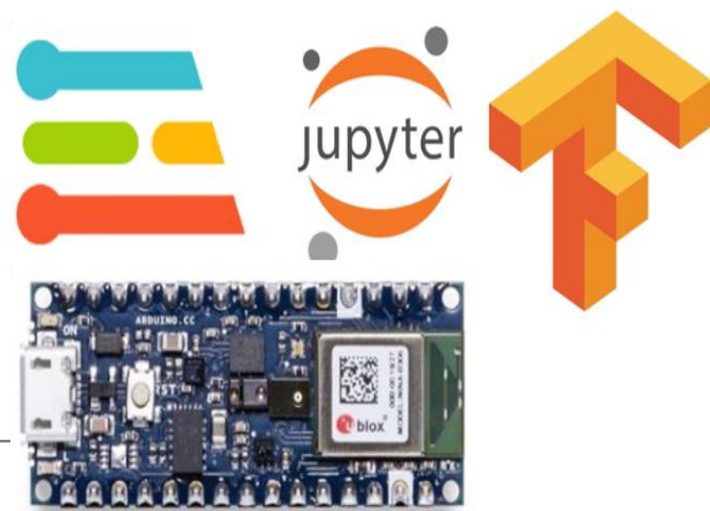
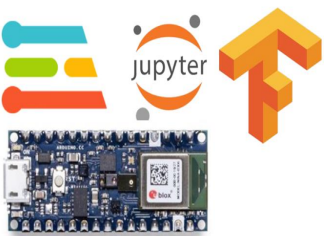


## Advanced Microprocessors

# MACHINE LEARNING ENVIRONMENT SETUP

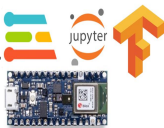
Dennis A. N. Gookyi

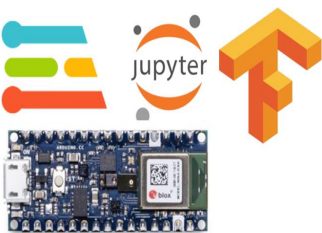




# CONTENTS

## ❖ Machine Learning Environment Setup





# TENSORFLOW ON A LOCAL MACHINE - TOOLS

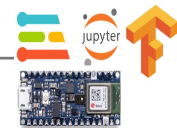
## ❖ TensorFlow on a local machine

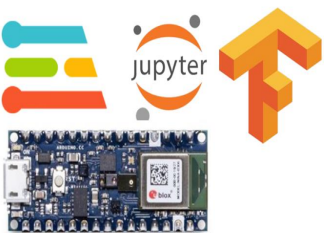


ANACONDA®

### Anaconda Jupyter

- Anaconda – Python Distribution.
- Jupyter Notebooks – Web-based program to code Python





# TENSORFLOW ON A LOCAL MACHINE - TOOLS

## ❖ TensorFlow on a local machine



**Numpy** – Multidimensional arrays and matrices

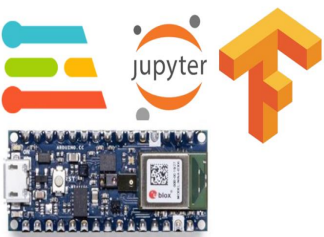
**Keras** – Python interface for ANN

**TensorFlow** – Training and Inference of DNN

**Pandas** – data manipulation and analysis

**OpenCV** – Real-time computer vision

**Matplotlib** – Plottig library



# INSTALLATION OF TOOLS

## ❖ Installation of tools


### Anaconda Distribution

## Free Download

Everything you need to get started in data science on your workstation.

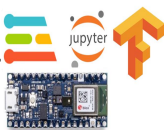
- ✓ Free distribution install
- ✓ Thousands of the most fundamental DS, AI, and ML packages
- ✓ Manage packages and environments from desktop application
- ✓ Deploy across hardware and software platforms

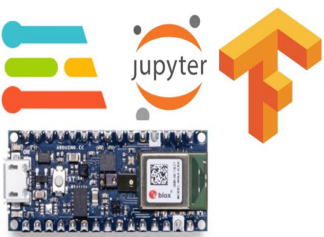
 Start Coding Now

 Download

Visit – The above link to download

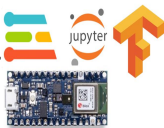
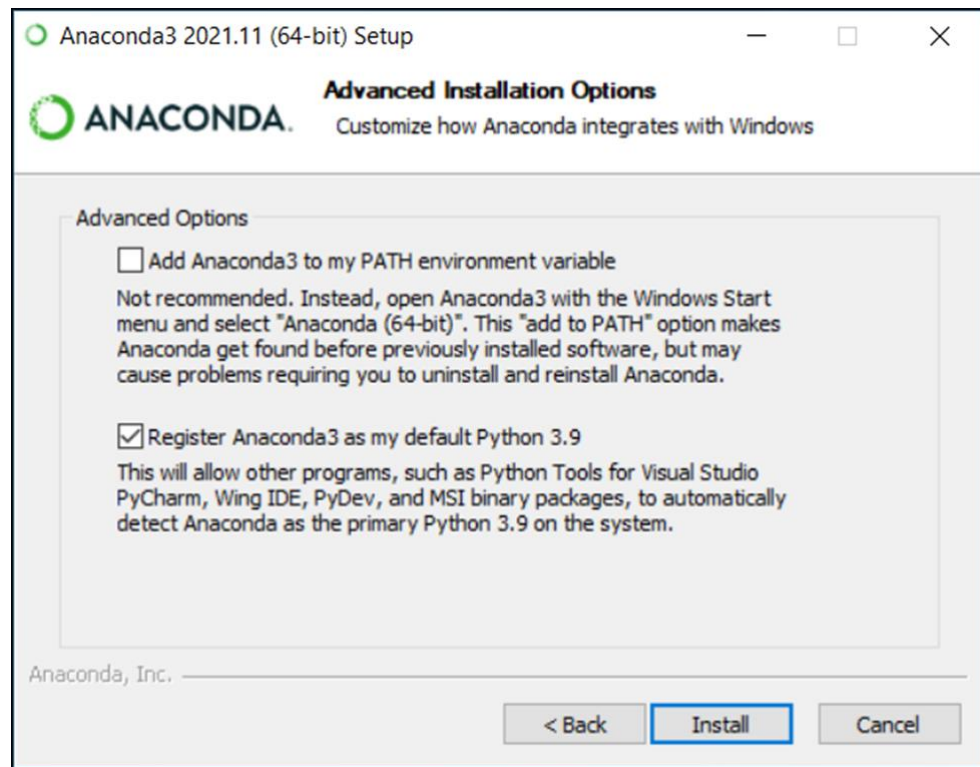
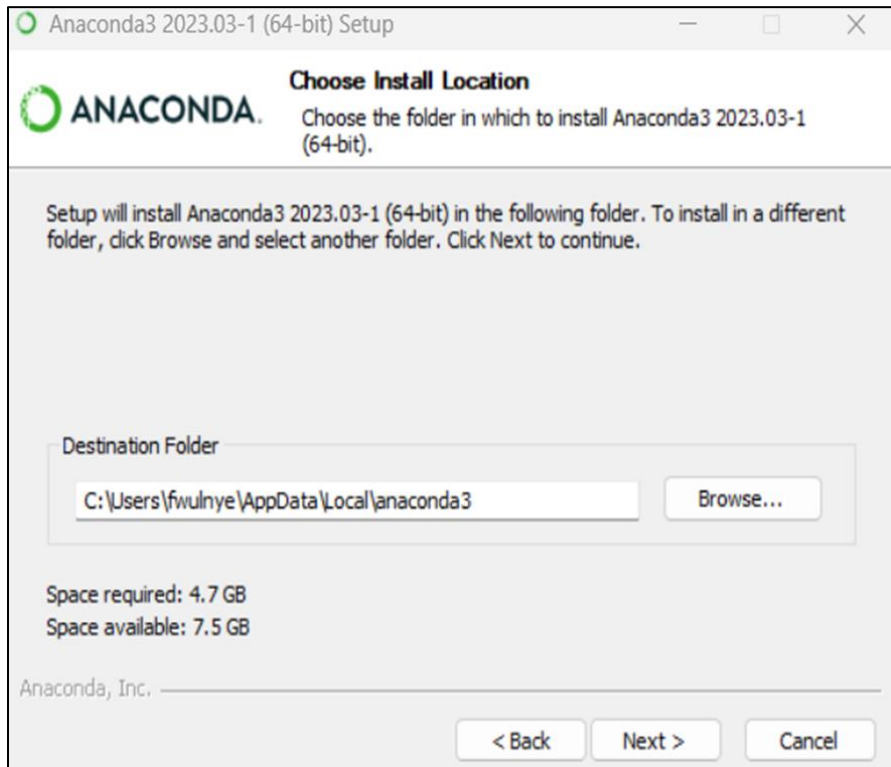
[Anaconda Download Link](#)



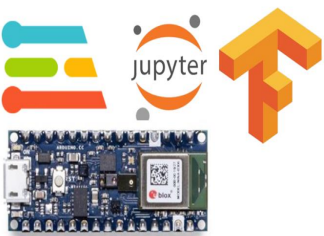


# INSTALLATION OF TOOLS

## ❖ Installation of tools

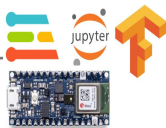
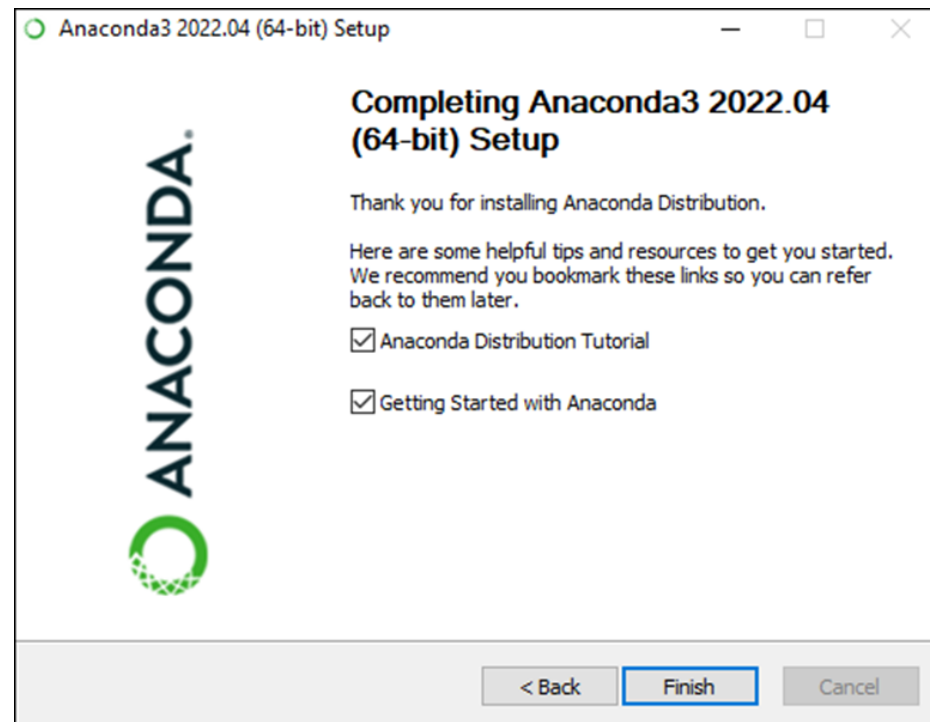


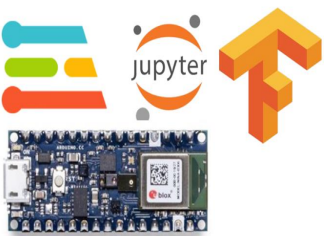




# INSTALLATION OF TOOLS

## ❖ Installation of tools





# INSTALLATION OF TOOLS

## ❖ Installation of tools

Anaconda Navigator

Connect

Home

Environments

Learning

Community

Anaconda Notebooks

Cloud notebooks with hundreds of packages ready to code.

Learn More

Documentation

Anaconda Blog

Create Clone Import Backup Remove

Search Environments

base (root)

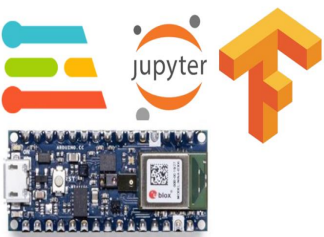
Installed Channels Update index...

Search Packages

Name	Description	Version
✓ _ipyw_lab_nb_ex...	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓ alabaster	Configurable, python 2+3 compatible sphinx theme.	0.7.12
✓ anaconda	Simplifies package management and deployment of anaconda	2022.10
✓ anaconda-client	Anaconda.org command line client library	1.11.0
✓ anaconda-project	Tool for encapsulating, running, and reproducing data science projects	0.11.1
✓ anyio	High level compatibility layer for multiple asynchronous event loop implementations on python	3.5.0
✓ appdirs	A small python module for determining appropriate platform-specific dirs.	1.4.4
✓ applaunchservices	Simple package for registering an app with apple launch services to handle uti and url	0.3.0
✓ appnope	Disable app nap on os x 10.9	0.1.2
✓ appscript	Control applescriptable applications from python.	1.1.2
✓ argon2-cffi	The secure argon2 password hashing algorithm.	21.3.0
✓ argon2-cffi-bindings	Low-level python cffi bindings for argon2	21.2.0
✓ arrow	Better dates & times for python	1.2.2
✓ astroid	A abstract syntax tree for python with inference support.	2.11.7
✓ astropy	Community-developed python library for astronomy	5.1

463 packages available

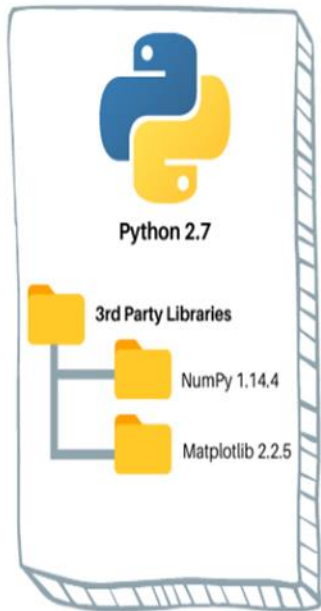




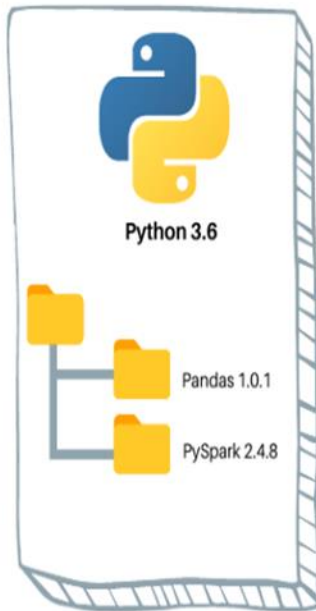
# INSTALLATION OF TOOLS

## ❖ Installation of tools

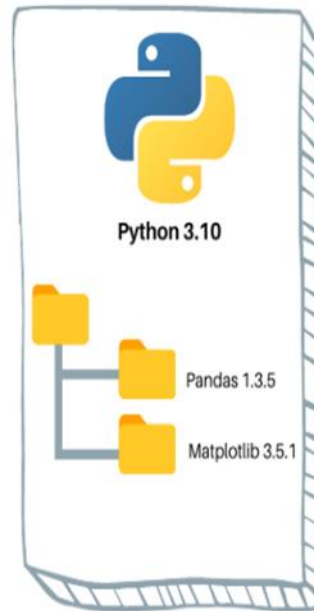
Virtual Environment 1



Virtual Environment 2



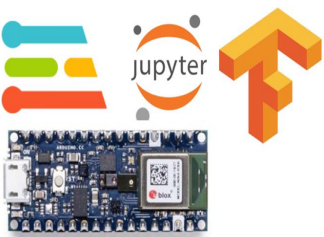
Virtual Environment 3



Virtual env name – tensorflow

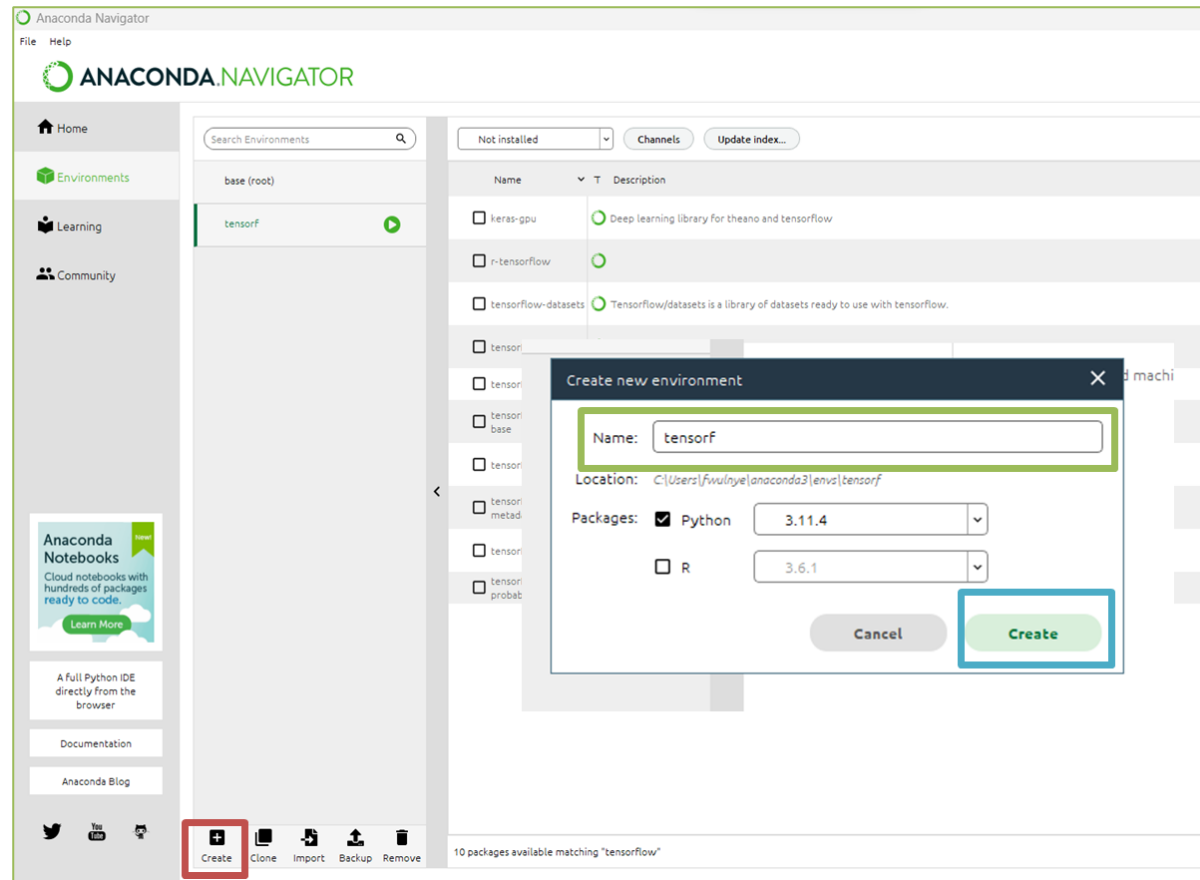
A tool to create isolated Python environments. virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

GUI – Anaconda Navigator



# INSTALLATION OF TOOLS

## ❖ Installation of tools

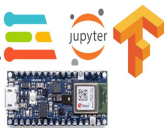


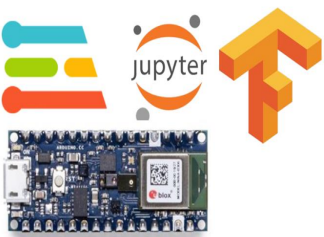
Click on Create

Name – Type in the name of your env

Select Python package and version – 3.11.4

Click Create





# INSTALLATION OF TOOLS

## ❖ Installation of tools

```

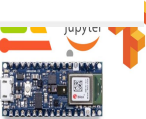
tensorfl_backup.yaml  libraries.txt x
libraries.txt
1 name: tensorf
2 channels:
3   - defaults
4 dependencies:
5   - numpy=1.23.5=py310h60c9a35_0
6   - numpy-base=1.23.5=py310h04254f7_0
7   - pandas=1.5.3=py310h4ed8f06_0
8   - scikit-image=0.19.3=py310hd77b12b_1
9   - scikit-learn=1.2.1=py310hd77b12b_0
10  - seaborn=0.12.2=py310haa95532_0
11  - tensorflow=2.10.0=mk1_py310hd99672f_0
12  - tensorflow-base=2.10.0=mk1_py310h6a7f48e_0
13  - pandas=1.5.3=py310h4ed8f06_0
14  - matplotlib=3.7.0=py310haa95532_0
15  - matplotlib-base=3.7.0=py310h4ed8f06_0
16  - opencv=4.6.0=py310h4ed8f06_3
17  - keras==2.12.0
  
```

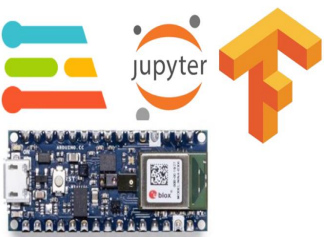
Navigate to the tensorf environment

Select not installed. And search and select for the library names in the library.txt .

Installed	Description	Version
✓ Installed	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓ Installed		1.0.1
✓ Installed	Configurable, python 2+3 compatible sphinx theme.	0.7.12
✓ anaconda-client	Anaconda.org command line client library	1.7.2
✓ appnope	Disable app nap on os x 10.9	0.1.0

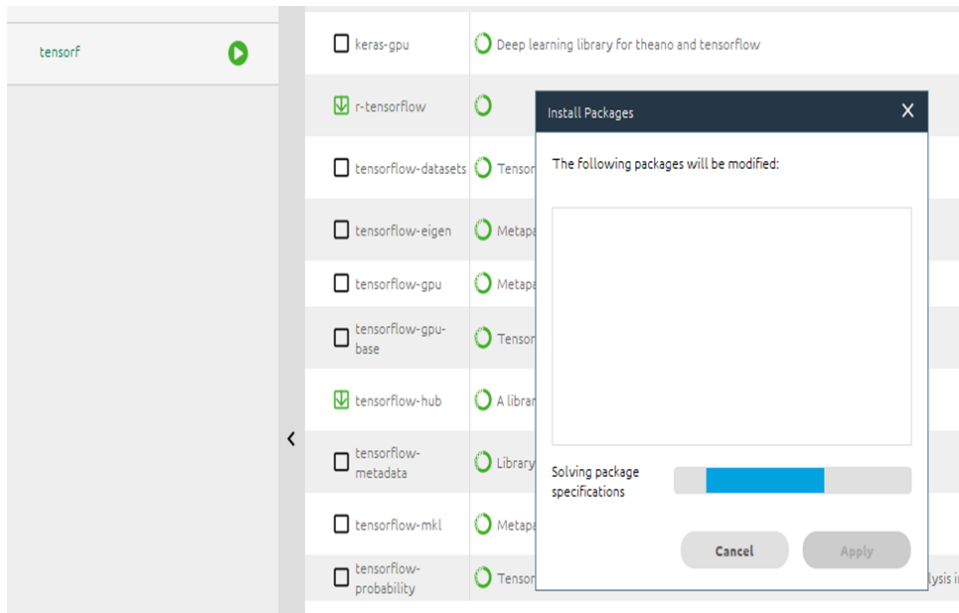
Name	Description
keras-gpu	Deep learning library for theano and tensorflow
r-tensorflow	
tensorflow-datasets	Tensorflow/datasets is a library of datasets ready to



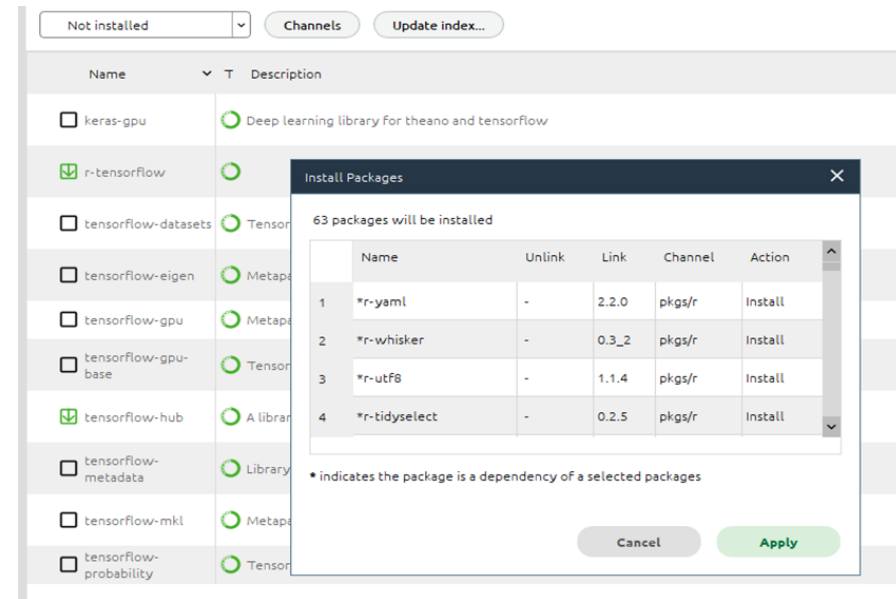


# INSTALLATION OF TOOLS

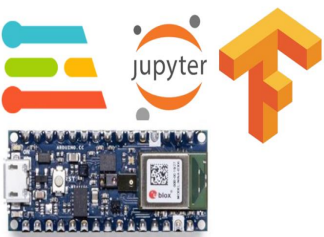
## ❖ Installation of tools



The package specification will be solved

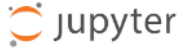


Click Apply to install



# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

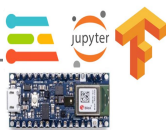
 Quit Logout

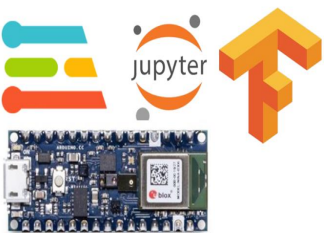
Files Running Clusters

Select items to perform actions on them. Upload New ↺

<input type="checkbox"/> 0		Name	Last Modified	File size
	📁	..	seconds ago	
<input type="checkbox"/>	📄	Exploring_Loss_Cost_Function.ipynb	5 days ago	120 kB
<input type="checkbox"/>	📄	IEST101_List_3_TF_Exploring_DNN_learning.ipynb	5 months ago	10 kB
<input type="checkbox"/>	📄	TF_First_Neural_Network.ipynb	4 days ago	69.2 kB
<input type="checkbox"/>	📄	TF_First_Neural_Network_v2_exploring_epochs.ipynb	5 months ago	103 kB
<input type="checkbox"/>	📄	TF_MNIST_Classification.ipynb	Running 31 minutes ago	51 kB
<input type="checkbox"/>	📄	my_first_model_no_training.h5	4 days ago	12.5 kB
<input type="checkbox"/>	📄	my_first_model_trained.h5	4 days ago	15 kB


Note book interface

















# NOTEBOOK EXAMPLE

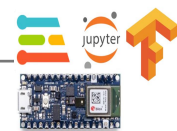
## ❖ Notebook Example: MNIST Classification

jupyter TF\_MNIST\_Classification Last Checkpoint: Last Monday at 9:52 AM (unsaved changes)  Logout

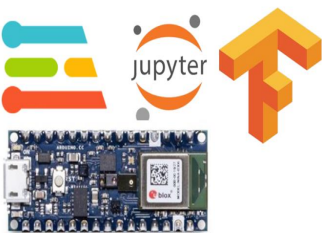
File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel) 

          Markdown 

### Digit Classification using Dense Neural Network (DNN)





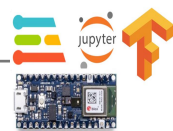


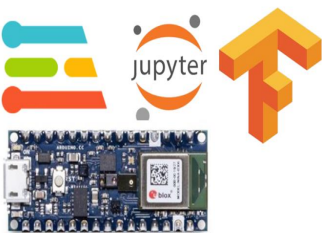
# NOTEBOOK EXAMPLE

❖ Notebook Example: MNIST Classification

## Import Libraries

```
In [1]: ▶ import numpy as np  
import matplotlib.pyplot as plt  
import tensorflow as tf
```





# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

### Upload and Explore Dataset

[MNIST](#) handwritten digits dataset

The MNIST database of handwritten digits, available from this [page](#), has a training set of 60,000 28x28 grayscale images of the 10 digits along a test set of 10,000 images. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

```
In [2]: ▶ data = tf.keras.datasets.mnist

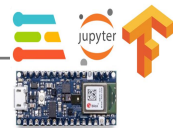
(training_images, training_labels), (val_images, val_labels) = data.load_data()
```

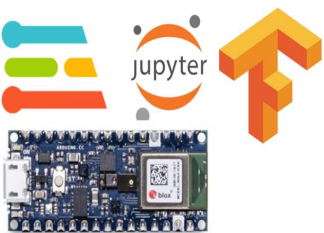
```
In [3]: ▶ print(training_images.shape)
print(training_labels.shape)

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

```
In [4]: ▶ print(val_images.shape)
print(val_labels.shape)

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





# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

### Exploring Labels

```
In [5]: training_labels
```

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

It is possible to keep training labels as "numbers", but in this case when compiling the model, you should use:  
`loss="sparse_categorical_crossentropy".`

And how about changing labels to categorical?

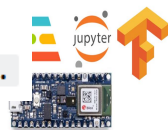
```
In [7]: training_labels[:10]
```

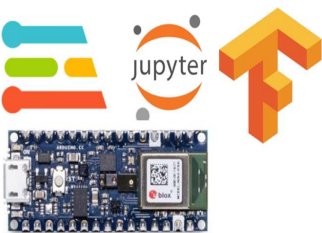
```
Out[7]: array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

```
In [8]: from tensorflow.keras.utils import to_categorical  
to_categorical(training_labels[:10])
```

```
Out[8]: array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],  
               [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],  
               [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],  
               [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],  
               [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],  
               [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)
```

When labels are defined as categories, when compiling the model you should use: `loss="categorical_crossentropy".`





# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

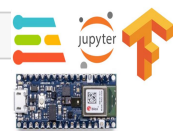
### Exploring images

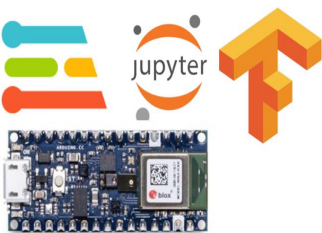
```
In [8]: ▶ np.set_printoptions(linewidth=200)  
print(training_images[2])
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  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  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  67 232 39  0  0  0  0]
 [ 0  0  0  0 62 81  0  0  0  0  0  0  0  0  0  0  0  0  0 120 180 39  0  0  0  0]
 [ 0  0  0  0 126 163  0  0  0  0  0  0  0  0  0  0  0  0  2 153 210 40  0  0  0  0]
 [ 0  0  0  0 220 163  0  0  0  0  0  0  0  0  0  0  0  0 27 254 162  0  0  0  0]
 [ 0  0  0  0 222 163  0  0  0  0  0  0  0  0  0  0  0  0 183 254 125  0  0  0  0]
 [ 0  0  0 46 245 163  0  0  0  0  0  0  0  0  0  0  0  0 198 254 56  0  0  0  0]
 [ 0  0  0 120 254 163  0  0  0  0  0  0  0  0  0  0  0 23 231 254 29  0  0  0  0]
 [ 0  0  0 159 254 120  0  0  0  0  0  0  0  0  0  0  0 163 254 216 16  0  0  0  0]
 [ 0  0  0 159 254 67  0  0  0  0  0  0  0  0  0 14 86 178 248 254 91  0  0  0  0]
 [ 0  0  0 159 254 85  0  0  0 47 49 116 144 150 241 243 234 179 241 252 40  0  0  0  0]
 [ 0  0  0 150 253 237 207 207 207 253 254 250 240 198 143 91 28 5 233 250  0  0  0  0]
 [ 0  0  0  119 177 177 177 177 177 98 56  0  0  0  0  0 102 254 220  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 254 137  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 254 57  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 254 57  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 255 94  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 254 96  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 254 153  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 169 255 153  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 96 254 153  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]]
```

```
In [9]: ▶ training_labels[2]
```

Out[9]: 4



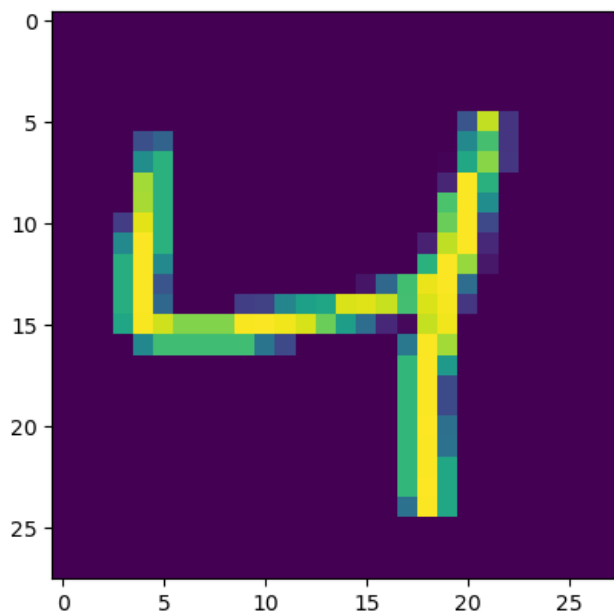


# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

```
In [10]: ▶ img = 2  
print("    Label of image {} is: {}".format(img, training_labels[img]))  
plt.imshow(training_images[img]);
```

Label of image 2 is: 4



```
In [11]: ▶ training_images.max()
```

Out[11]: 255

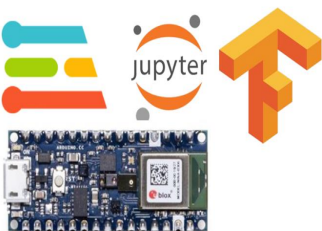
```
In [12]: ▶ training_images.min()
```

Out[12]: 0







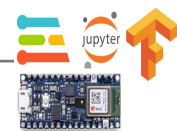
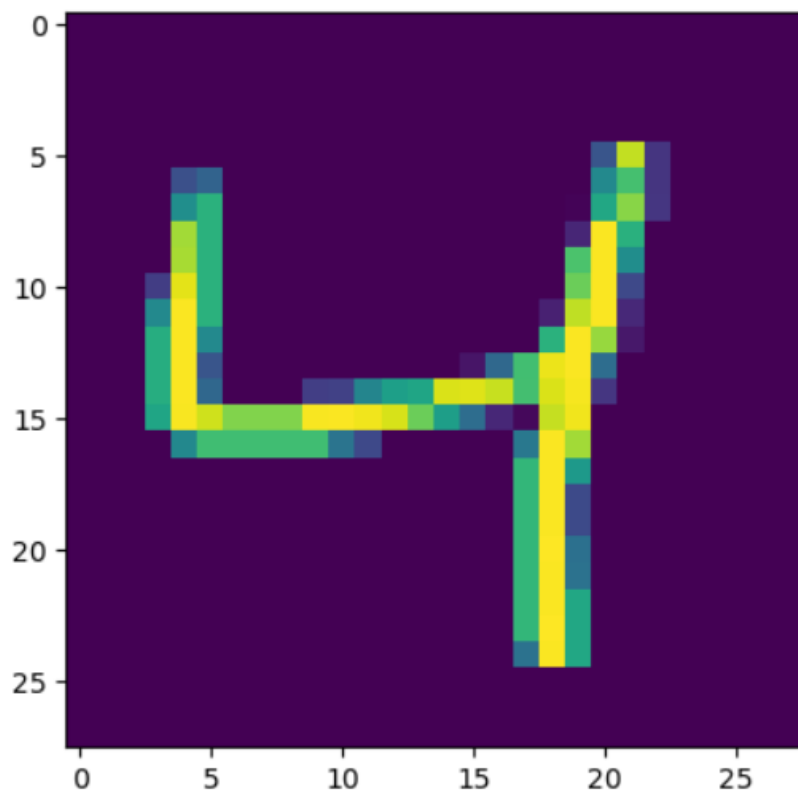


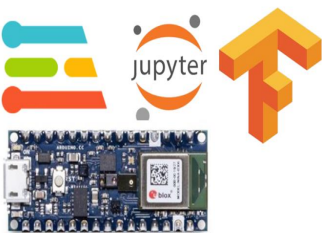
# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

```
In [12]: ▶ img = 2  
print("    Label of image {} is: {}".format(img, training_labels[img]))  
plt.imshow(training_images[img]);
```

Label of image 2 is: 4





# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

### Define and Compile Model

```
In [7]: ▶ model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,28)),  
                                              tf.keras.layers.Dense(20, activation=tf.nn.relu),  
                                              tf.keras.layers.Dense(10, activation=tf.nn.softmax)])  
  
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 20)	15700
dense_1 (Dense)	(None, 10)	210

=====

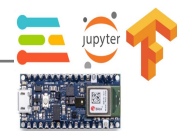
Total params: 15,910

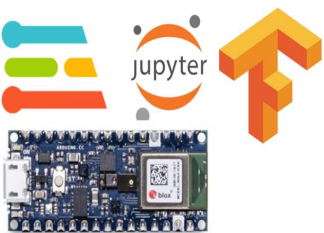
Trainable params: 15,910

Non-trainable params: 0

=====

```
In [8]: ▶ model.compile(  
    optimizer='adam',  
    loss='sparse_categorical_crossentropy', # Labels are not as an array  
    metrics=['accuracy'] # Calculates how often predictions equal labels  
)
```



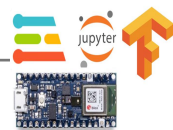


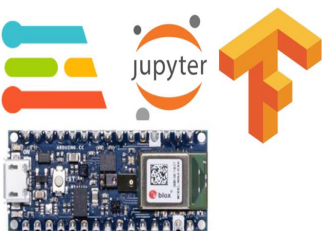
# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

### Train the model

```
In [9]: ▶ %%time
        history = model.fit(
            training_images,
            training_labels,
            epochs=20,
        )
```

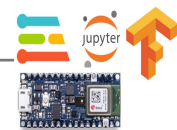


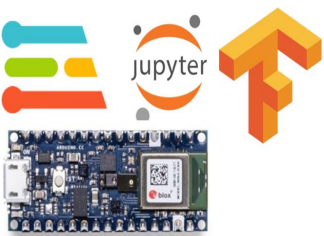


# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

```
Epoch 1/20
1875/1875 [=====] - 3s 1ms/step - loss: 0.4069 - accuracy: 0.8868
Epoch 2/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.2317 - accuracy: 0.9342
Epoch 3/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1939 - accuracy: 0.9438
Epoch 4/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1702 - accuracy: 0.9509
Epoch 5/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1543 - accuracy: 0.9548
Epoch 6/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1412 - accuracy: 0.9582
Epoch 7/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1308 - accuracy: 0.9614
Epoch 8/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1229 - accuracy: 0.9635
Epoch 9/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1175 - accuracy: 0.9648
Epoch 10/20
1875/1875 [=====] - 2s 1ms/step - loss: 0.1116 - accuracy: 0.9664
Epoch 11/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.1066 - accuracy: 0.9682
Epoch 12/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.1035 - accuracy: 0.9693
Epoch 13/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0997 - accuracy: 0.9702
Epoch 14/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0958 - accuracy: 0.9707
Epoch 15/20
1875/1875 [=====] - 3s 1ms/step - loss: 0.0926 - accuracy: 0.9723
Epoch 16/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0912 - accuracy: 0.9720
Epoch 17/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0879 - accuracy: 0.9736
Epoch 18/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0865 - accuracy: 0.9737
Epoch 19/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0842 - accuracy: 0.9745
Epoch 20/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0823 - accuracy: 0.9756
CPU times: total: 4min 23s
Wall time: 55.7 s
```





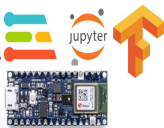
# NOTEBOOK EXAMPLE

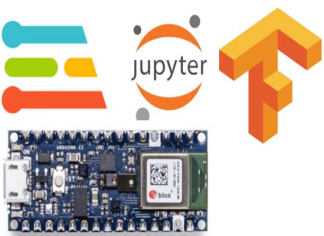
## ❖ Notebook Example: MNIST Classification

Inspecting the model

```
In [10]: ▶ train_eval = model.evaluate(training_images, training_labels)
          print ("Training data Accuracy: {:.1f}%".format(train_eval[1]*100))

1875/1875 [=====] - 2s 1ms/step - loss: 0.0732 - accuracy: 0.9783
Training data Accuracy: 97.8%
```

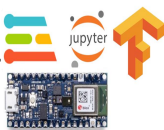
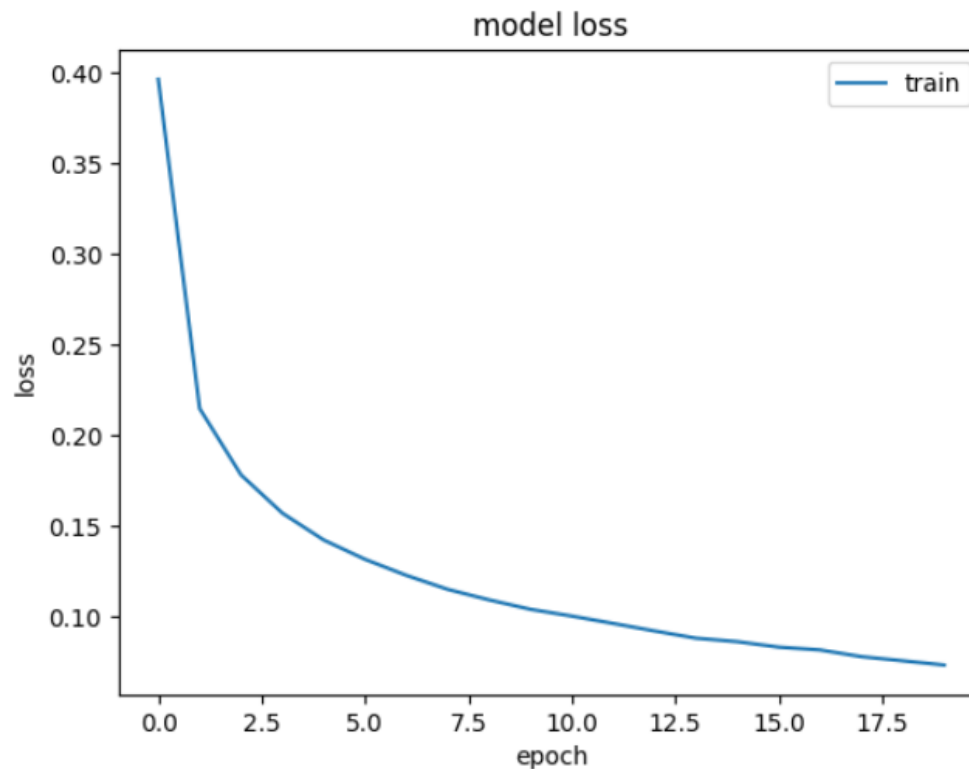




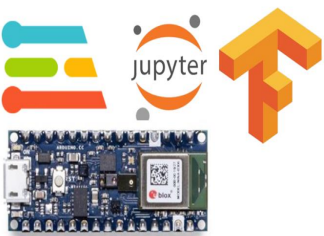
# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

```
✓ [2] plt.plot(history.history['loss'])  
0s plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train'], loc='upper right')  
plt.show()
```



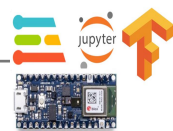
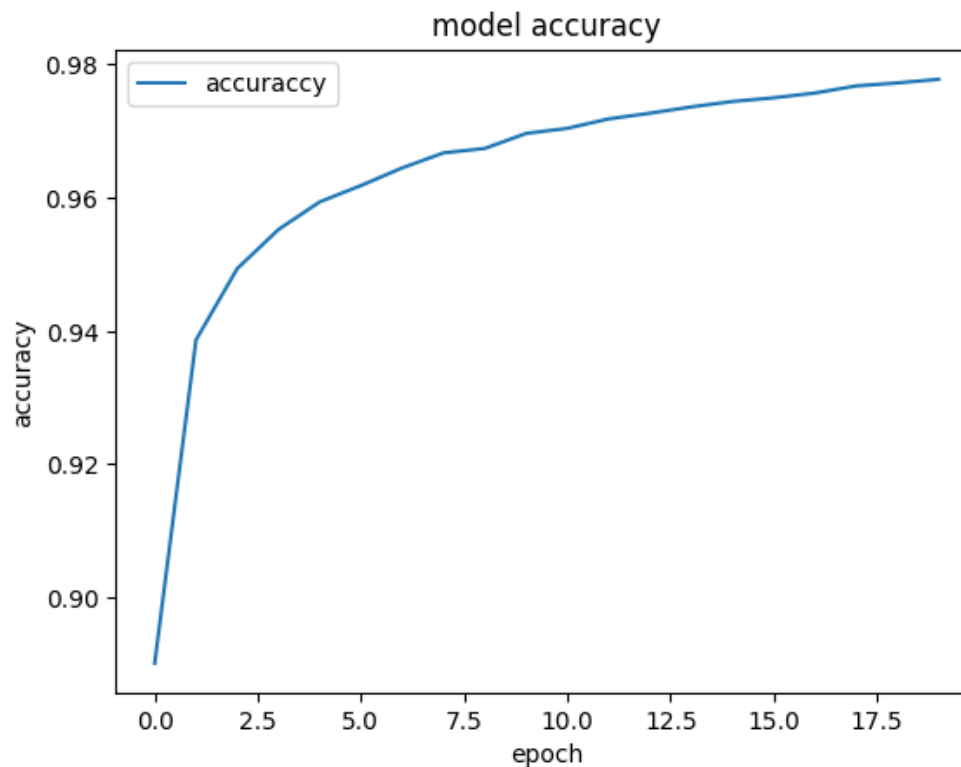


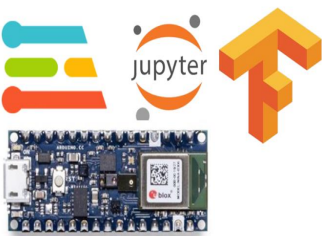


# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

```
✓ [23] plt.plot(history.history['accuracy'])  
      plt.title('model accuracy')  
      plt.ylabel('accuracy')  
      plt.xlabel('epoch')  
      plt.legend(['accuracy'], loc='upper left')  
      plt.show()
```





# NOTEBOOK EXAMPLE

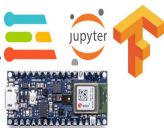
## ❖ Notebook Example: MNIST Classification

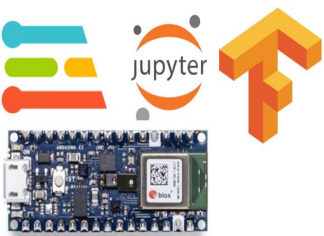
### ✓ Testing the trained model

Using `model.evaluate`, you can get metrics for a test set. In this case we only have a training set and a validation set, so we can try it out with the validation set. The accuracy will be slightly lower, at maybe 96%. This is because the model hasn't previously seen this data and may not be fully generalized for all data. Still it's a pretty good score. You can also predict images, and compare against their actual label. The [0] image in the set is a number 7, and here you can see that neuron 7 has a  $9.9e-1$  (99%+) probability, so it got it right!

```
✓ [24] test_eval = model.evaluate(val_images, val_labels)
1s      print ("Testing data Accuracy: {:.1f}%".format(test_eval[1]*100))
```

```
313/313 [=====] - 1s 2ms/step - loss: 0.1450 - accuracy: 0.9625
Testing data Accuracy: 96.2%
```

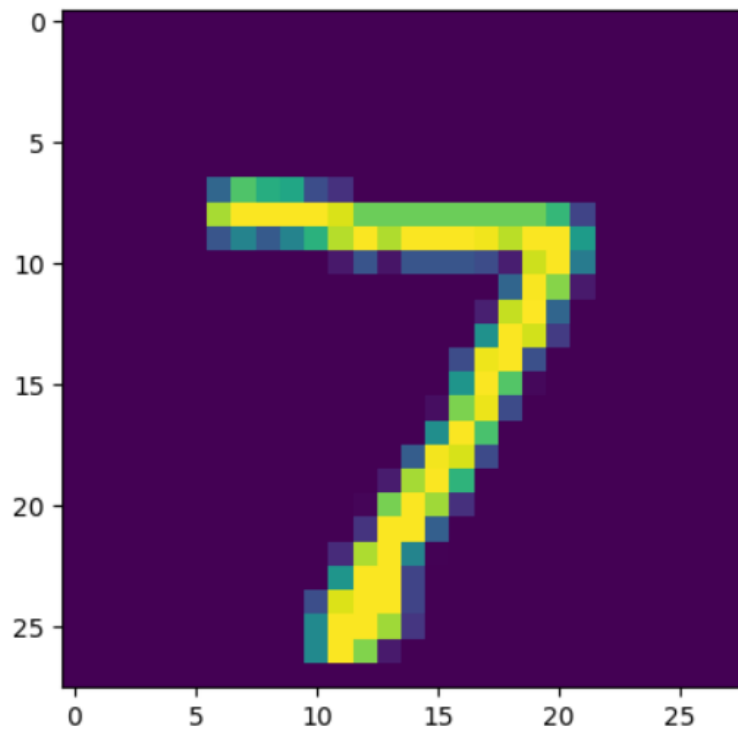




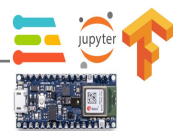
# NOTEBOOK EXAMPLE

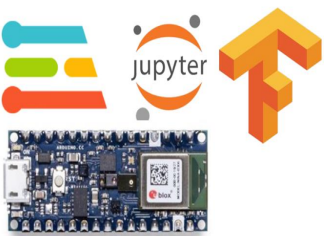
## ❖ Notebook Example: MNIST Classification

```
✓ [25] plt.imshow(val_images[0]);  
0s
```



```
✓ [26] print(val_labels[0])  
0s
```





# NOTEBOOK EXAMPLE

## ❖ Notebook Example: MNIST Classification

```
✓ [27] classifications = model.predict(val_images)
1s      print(classifications[0])

313/313 [=====] - 1s 2ms/step
[7.2781484e-05 3.8751616e-11 6.5620356e-05 1.0001573e-03 6.9909703e-12 1.5672607e-05 8.4830062e-15 9.9873394e-01 1.7328633e-05 9.4513402e-05]

✓ [28] # Returns the indices of the maximum values along an axis.
0s      np.argmax(classifications[0])

7
```

