1. Checking the TF version and availability of physical devices

a) Get the version of TensorFlow running on your machine?

In [1]:

```
import tensorflow as tf
print(tf.__version__)
```

```
2.7.0
```

b) Get the type & number of physical devices available on your machine, print what are they, and test whether the GPU is available?

1. https://medium.com/ibm-data-ai/memory-hygiene-with-tensorflow-during-model-training-and-deployment-for-inference-45cf49a15688
2. GPU testing: https://beverly-wang0005.medium.com/initial-setup-install-packages-offline-create-virtual-environment-set-up-gpu-for-tensorflow-1d7f802db356

In [ ]:

```
# Creates 2 virtual devices cpu:0 and cpu:1 for using distribution strategy
cpus = tf.config.experimental.list_physical_devices("CPU")
print('CPUs available are:')
for cpu in cpus:
    print(cpu)
```

In [ ]:

```
gpus = tf.config.experimental.list_physical_devices("GPU")
print('GPUs available are:')
if gpus:
    print(gpu)
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu,True)
```

In [6]:

```
# tf.test.is_gpu_available()
print(tf.test.is_gpu_available(),tf.test.gpu_device_name)
```

```
True <function gpu_device_name at 0x000001F97FB651F0>
```

In [ ]:

```
# tf.config.experimental.set_virtual_device_configuration(
#     physical_devices[0], [
#         tf.config.experimental.VirtualDeviceConfiguration(),
#         tf.config.experimental.VirtualDeviceConfiguration()
#     ])
```

2. Random number generator

a) Create a random number generator using TensorFlow with a seed of 42 \ b) Generate a Gaussian random matrix of shape 2x3

a) What is the need for setting a 'seed' value in any random number generation?

b) Create two random number generators using TensorFlow with the same seed of 42, create two random gaussian tensors of shape 2x3, and verify that the both tensors are identical.

In [9]:

```
# Create two random (but the same) tensors
random_1 = tf.random.Generator.from_seed(42) # set the seed for reproducibility
random_1 = random_1.normal(shape=(3, 2)) # create tensor from a normal distributio
random_2 = tf.random.Generator.from_seed(42)
random_2 = random_2.normal(shape=(3, 2))

# Are they equal?
random_1, random_2, random_1 == random_2
```

Out[9]:

```
(<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
 array([[-0.7565803 , -0.06854702],
        [ 0.07595026, -1.2573844 ],
        [-0.23193765, -1.8107855 ]], dtype=float32)>,
 <tf.Tensor: shape=(3, 2), dtype=float32, numpy=
 array([[-0.7565803 , -0.06854702],
        [ 0.07595026, -1.2573844 ],
        [-0.23193765, -1.8107855 ]], dtype=float32)>,
 <tf.Tensor: shape=(3, 2), dtype=bool, numpy=
 array([[ True,  True],
        [ True,  True],
        [ True,  True]])>)
```

c) Create two random number generators using TensorFlow with two different seed values say 42 & 11, create two random gaussian tensors of shape 2x3, and verify that the both tensors are not identical.

In [10]:

```
# Create two random (and different) tensors
random_3 = tf.random.Generator.from_seed(42)
random_3 = random_3.normal(shape=(3, 2))
random_4 = tf.random.Generator.from_seed(11)
random_4 = random_4.normal(shape=(3, 2))

# Check the tensors and see if they are equal
random_3, random_4, random_1 == random_3, random_3 == random_4
```

Out[10]:

```
(<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
 array([[-0.7565803 , -0.06854702],
        [ 0.07595026, -1.2573844 ],
        [-0.23193765, -1.8107855 ]], dtype=float32)>,
 <tf.Tensor: shape=(3, 2), dtype=float32, numpy=
 array([[ 0.2730574 , -0.29925638],
        [-0.3652325 ,  0.61883307],
        [-1.0130816 ,  0.2829171 ]], dtype=float32)>,
 <tf.Tensor: shape=(3, 2), dtype=bool, numpy=
 array([[ True,  True],
        [ True,  True],
        [ True,  True]])>,
 <tf.Tensor: shape=(3, 2), dtype=bool, numpy=
 array([[False, False],
        [False, False],
        [False, False]])>)
```

3. Shuffling of Tensors

a) Shuffle the given Tensor with and without an operation seed value. Write down your observations.

In [11]:
```
# Shuffle a tensor (valuable for when you want to shuffle your data)
not_shuffled = tf.constant([[10, 7],
                            [3, 4],
                            [2, 5]])
```

In [19]:
```
# Gets different results each time
tf.random.shuffle(not_shuffled)
```

Out[19]:
```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[10,  7],
       [ 3,  4],
       [ 2,  5]])>
```

In [29]:
```
# Shuffle in the same order every time using the seed parameter (won't acutally be
tf.random.shuffle(not_shuffled, seed=42)
```

Out[29]:
```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[ 2,  5],
       [ 3,  4],
       [10,  7]])>
```

b) Show that 'operation seed' in 'tf.random.shuffle' and the 'global seed' in 'tf.random.set_seed' are different? Illustrate that having both gives the tensor in same order every time after shuffling?

In [34]:
```
# Shuffle in the same order every time

# Set the global random seed
tf.random.set_seed(42)

# Set the operation random seed
tf.random.shuffle(not_shuffled, seed=42)
```

Out[34]:
```
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[10,  7],
       [ 3,  4],
       [ 2,  5]])>
```

4. Reshaping the tensors

a)

(i) Construct a vector consisting of first 24 integers using 'numpy'.\ (ii) Convert that numpy vector into a Tensor of rank 3. \ (iii) Write your observations on how the elements of the vector got rearranged in the rank 3 tensor.

In [35]:
```
import numpy as np
numpy_A = np.arange(1, 25, dtype=np.int32) # create a NumPy array between 1 and 25
A = tf.constant(numpy_A,
                shape=[2, 4, 3]) # note: the shape total (2*4*3) has to match the
numpy_A, A
```

Out[35]:

```
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24]),
 <tf.Tensor: shape=(2, 4, 3), dtype=int32, numpy=
 array([[[ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9],
         [10, 11, 12]],

        [[13, 14, 15],
         [16, 17, 18],
         [19, 20, 21],
         [22, 23, 24]]])>)
```

b)

(i) Create a tensor of rank 2. \ (ii) Convert that tensor into another tensor of shape 2x2x1 using 'tf.newaxis'.

In [36]:

```
# Create a rank 2 tensor (2 dimensions)
rank_2_tensor = tf.constant([[10, 7],
                             [3, 4]])
```

In [42]:

```
# Get the last item of each row
rank_2_tensor[:, -1]
# print(rank_2_tensor[-1,:])
```

Out[42]:

```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([7, 4])>
```

In [43]:

```
# Add an extra dimension (to the end)
rank_3_tensor = rank_2_tensor[..., tf.newaxis] # in Python "..." means "all dimens
rank_2_tensor, rank_3_tensor # shape (2, 2), shape (2, 2, 1)
```

Out[43]:

```
(<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
 array([[10,  7],
        [ 3,  4]])>,
 <tf.Tensor: shape=(2, 2, 1), dtype=int32, numpy=
 array([[[10],
         [ 7]],

        [[ 3],
         [ 4]]])>)
```

c)

(i) Create a tensor of rank 2. \ (ii) Convert that tensor into another tensor of shape 2x2x1 using 'tf.expand_dims'.

In [44]:

```
# Create a rank 2 tensor (2 dimensions)
rank_2_tensor = tf.constant([[10, 7],
                             [3, 4]])
```

In [45]:

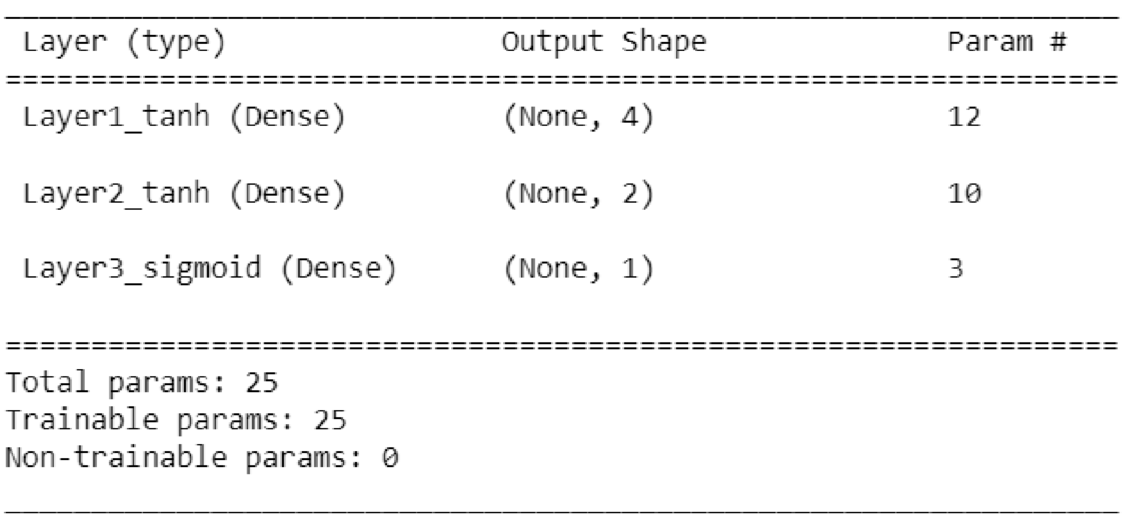
```
tf.expand_dims(rank_2_tensor, axis=-1) # "-1" means last axis
```

Out[45]:

```
<tf.Tensor: shape=(2, 2, 1), dtype=int32, numpy=
array([[[10],
        [ 7]],

       [[ 3],
        [ 4]]])>
```

5. ANN



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Layer1_tanh (Dense)         (None, 4)                 12

 Layer2_tanh (Dense)         (None, 2)                 10

 Layer3_sigmoid (Dense)      (None, 1)                 3

=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

a) Import all the necessary libraries required for creating the above neural network model

In [55]:

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
# from tensorflow.keras.utils import to_categorical, plot_model
```

b) Construct the model using Sequential API. Consider the conrresponding activation functions as specified in the Layer Type, and assume that the input feature vector has two features in it.

In [ ]:

```
model = Sequential()
model.add(Dense(4, input_shape=(2,), activation='tanh',name="Layer1_tanh"))
model.add(Dense(2, activation='tanh',name="Layer2_tanh"))
model.add(Dense(1, activation='sigmoid', name="Layer3_sigmoid"))
```

c) Compile the model with the following details and print the model as given in the figure

(i) Optimizer = Adam \ (ii) Loss function = Choose appropriately from the layer types \ (iii) Metrics = Choose appropriately considering the binary class classification task

In [66]:

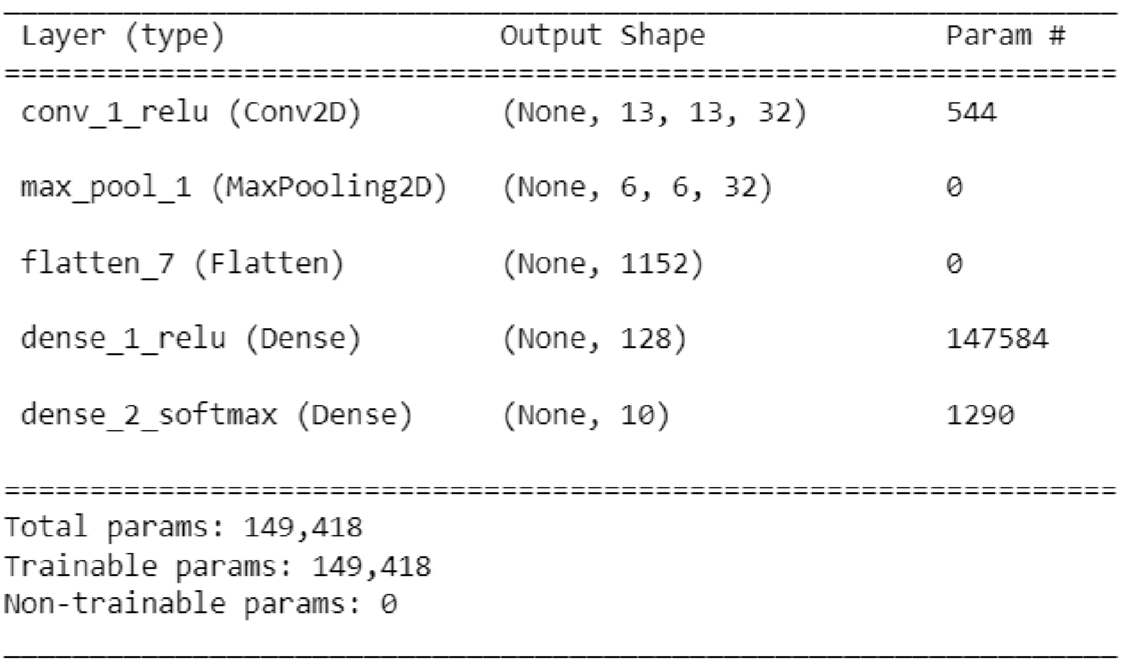
```
model.compile(Adam(lr=0.01), 'binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 Layer1_tanh (Dense)         (None, 4)                 12        
                                                                 
 Layer2_tanh (Dense)         (None, 2)                 10        
                                                                 
 Layer3_sigmoid (Dense)      (None, 1)                 3         
                                                                 
=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

d) Construct the model using Functional API. Consider the conrresponding activation functions as specified in the Layer Type, and assume that the input feature vector has two features in it.

e) Compile the model with the following details and print the model as given in the figure

6. CNNs



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv_1_relu (Conv2D)        (None, 13, 13, 32)        544       
                                                                 
 max_pool_1 (MaxPooling2D)   (None, 6, 6, 32)          0         
                                                                 
 flatten_7 (Flatten)         (None, 1152)              0         
                                                                 
 dense_1_relu (Dense)        (None, 128)               147584    
                                                                 
 dense_2_softmax (Dense)     (None, 10)                1290      
                                                                 
=================================================================
Total params: 149,418
Trainable params: 149,418
Non-trainable params: 0
_________________________________________________________________
```

a) Import all the necessary things

```
In [77]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten
         from tensorflow.keras.optimizers import Adam
```

b) Construct the model using Sequential API. Consider the conrresponding activation functions as specified in the Layer Type, and assume that the input is of shape 28x28x1.

In [87]:

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(4,4),
                 input_shape=(28,28,1),activation='relu',strides=(2, 2),
                 padding="valid",name="conv_1_relu"))
model.add(MaxPool2D(pool_size=(2,2),name="max_pool_1"))
model.add(Flatten())
model.add(Dense(128,activation='relu',name='dense_1_relu'))
model.add(Dense(10,activation='softmax',name='dense_2_softmax'))
```

c) Compile the model with the following details and print the model as given in the figure

(i) Optimizer = Adam \ (ii) Loss function = Choose appropriately from the layer types \ (iii) Metrics = Choose appropriately considering the multiclass classification task

In [89]:

```
model.compile(loss='categorical_crossentropy',
             optimizer='adam',
             metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_1_relu (Conv2D)        (None, 13, 13, 32)        544

 max_pool_1 (MaxPooling2D)   (None, 6, 6, 32)          0

 flatten_7 (Flatten)         (None, 1152)              0

 dense_1_relu (Dense)        (None, 128)               147584

 dense_2_softmax (Dense)     (None, 10)                1290

=================================================================
Total params: 149,418
Trainable params: 149,418
Non-trainable params: 0
_________________________________________________________________
```

d) Construct the model using Functional API. Consider the conrresponding activation functions as specified in the Layer Type, and assume that the input is of shape 28x28x1.

e) Compile the model with the following details and print the model as given in the figure

(i) Optimizer = Adam \ (ii) Loss function = Choose appropriately from the layer types \ (iii) Metrics = Choose appropriately considering the multiclass classification task

7. Loading and preprocessing the the input image data