1) Program to implement CNN using mnist model

```
# Program to implement CNN using mnist model
import tensorflow_datasets as tfds
mnist_bldr=tfds.builder('mnist')
mnist_bldr.download_and_prepare()
datasets=mnist_bldr.as_dataset(shuffle_files=False)
mnist_train_orig=datasets['train']
mnist_test_orig=datasets['test']
print('Successfully executed')
mnist_test, info = tfds.load(name="mnist", with_info="true")
print(info)
import tensorflow as tf
BUFFER_SIZE=10000
BATCH_SIZE=64
NUM_EPOCHS=20
mnist_train=mnist_train_orig.map(lambda
item:(tf.cast(item['image'],tf.float32)/255.0,tf.cast(item['label'],tf.
int32)))

mnist_test=mnist_test_orig.map(lambda
item:(tf.cast(item['image'],tf.float32)/255.0,tf.cast(item['label'],tf.
int32)))

tf.random.set_seed(1)
mnist_train=mnist_train.shuffle(buffer_size=BUFFER_SIZE,reshuffle_each_
iteration=False)
mnist_valid=mnist_train.take(10000).batch(BATCH_SIZE)
mnist_train=mnist_train.skip(10000).batch(BATCH_SIZE)  # Dataset skips
10000 elements because 10000 elements are alloted to validation data
set
print('Successfully executed')

#Constructing a CNN

model=tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=(5,5),strides=(
2,2),padding='same',data_format='channels_last',name='conv_1',activatio
n='relu'))
model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2),name='pool_1'))

model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=(5,5),strides=(
1,1),padding='same',data_format='channels_last',name='conv_2',activatio
n='relu')) # Vary
model.add(tf.keras.layers.MaxPool2D(pool_size=(2,2),name='pool_2')) #
Vary
print('Successfully executed')

model.add(tf.keras.layers.Flatten())
```

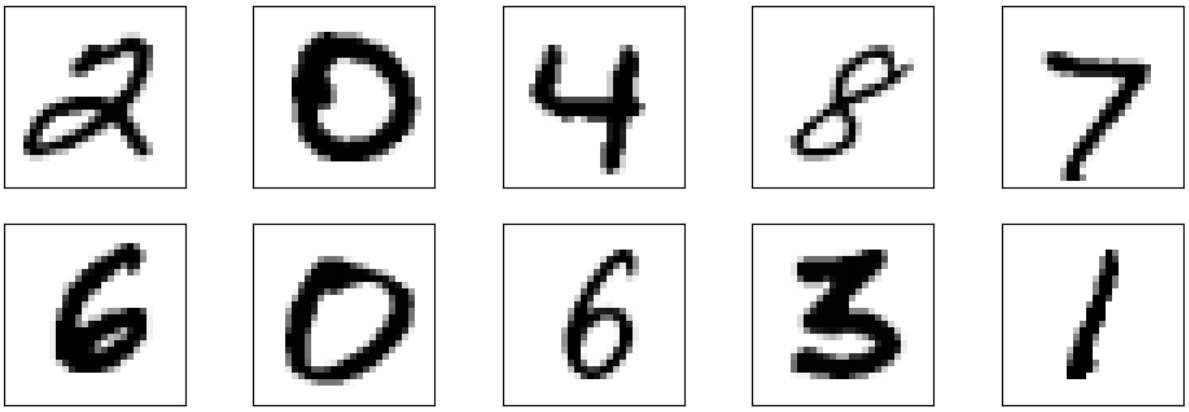
```
model.add(tf.keras.layers.Dense(units=1024,activation='relu')) # Vary


model.add(tf.keras.layers.Dropout(rate=0.5)) # Vary

model.add(tf.keras.layers.Dense(units=1024,activation='softmax'))

model.build(input_shape=(None,28,28,1))
#model.compile(
#    optimizer=tf.keras.optimizers.Adam(),loss=tf.keras.losses.Categori
calCrossentropy(),metrics=['Accuracy']
#)
model.compile(optimizer='adam',loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
#history=model.fit(mnist_train,epochs=1,batch_size=64,validation_data=m
nist_valid,shuffle=True) # Batch_size =64
history=model.fit(mnist_train,epochs=1,validation_data=mnist_valid,shuf
fle=True) # Batch_size =64
print('Successfully executed')

model.summary()

test_results=model.evaluate(mnist_test.batch(20))
print('Test Acc.{:.2f}\%'.format(test_results[1]*100))

import matplotlib.pyplot as plt
#import numpy as np
batch_test=next(iter(mnist_test.batch(10)))
preds=model(batch_test[0])
tf.print(preds.shape)
preds=tf.argmax(preds,axis=1)
print(preds)
#plt.show()
fig=plt.figure(figsize=(12,4))
for i in range(10):
  ax=fig.add_subplot(2,5,i+1)
  ax.set_xticks([]);ax.set_yticks([])
  img=batch_test[0][i,:,:,0]
  ax.imshow(img,cmap='gray_r')
  #ax.test(0.9,0.1,'{}'.format(preds[i]),size=15,color='blue',horizonta
lalignment='center',verticalalignment='center',transform=ax.transAxes)
plt.show()
```

OUTPUT

```
Successfully executed
It is the code segment to construct CNN
Successfully executed
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5729:
UserWarning: "`sparse_categorical_crossentropy` received
`from_logits=True`, but the `output` argument was produced by a Softmax
activation and thus does not represent logits. Was this intended?
  output, from_logits = _get_logits(
782/782 [==============================] - 78s 94ms/step - loss: 0.3437 -
accuracy: 0.8983 - val_loss: 0.0739 - val_accuracy: 0.9768
Successfully executed
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv_1 (Conv2D)	(None, 14, 14, 32)	832
pool_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv_2 (Conv2D)	(None, 7, 7, 64)	51264
pool_2 (MaxPooling2D)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 1024)	590848
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600

```
Total params: 1692544 (6.46 MB)
Trainable params: 1692544 (6.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
500/500 [==============================] - 4s 7ms/step - loss: 0.0597 -
accuracy: 0.9800
Test Acc.98.00\%
TensorShape([10, 1024])
tf.Tensor([2 0 4 8 7 6 0 6 3 1], shape=(10,), dtype=int64)
```



2) Program to implement RNN

```
print('Program to implement RNN')
import tensorflow as tf
tf.random.set_seed(1)
rnn_layer=tf.keras.layers.SimpleRNN(
    units=2,use_bias=True,
    return_sequences=True)
rnn_layer.build(input_shape=(None,None,5))
w_xh,w_oo,b_h=rnn_layer.weights
print('w_xh shape:',w_xh.shape)
print('w_oo shape:',w_oo.shape)
print('b_h shape:',b_h.shape)
print('The weights of input to hidden layer:w_xh')
print(w_xh.numpy())

print('The weights of output to output layer:w_oo')
print(w_oo.numpy())



print('The weights of bias of hidden layer:b_h')
print(b_h.numpy())

x_seq=tf.convert_to_tensor(
    [[1.0]*5,[2.0]*5,[3.0]*5],
    dtype=tf.float32)
print('Input ')
print(x_seq)
#output of SimpleRNN
output=rnn_layer(tf.reshape(x_seq,shape=(1,3,5)))

#Manually computing the output
out_man = []
for t in range(len(x_seq)):
    print('It is for t=',t)
    xt=tf.reshape(x_seq[t],(1,5))

    print('Time step{}=>'.format(t))
    print('Input  xt=     :',xt.numpy())
    print('It is  w_xh :',w_xh.numpy())
    print('It is   b_h  :',b_h.numpy())
    ht=tf.matmul(xt,w_xh)+ b_h
    print('  ht=xt*w_xh+b_h    :',ht.numpy())
    if t>0:
      prev_o=out_man[t-1]
      print('It is prev_o',prev_o.numpy())
```

```
     # print(prev_o)
    else:
      prev_o=tf.zeros(shape=(ht.shape))
      print('It is inside else ')
            #print(t)
      print('It is ht',ht.numpy())
      print(ht)
    print('It is prev_o',prev_o.numpy())
    print('It is w_oo',w_oo.numpy())
    print('It is ht',ht.numpy())
    ot=ht + tf.matmul(prev_o,w_oo)
    print('It is ot = ht+prev_o*w_oo  before tanh ',ot.numpy())
    ot=tf.math.tanh(ot)
    out_man.append(ot)
    print(' output (manual) :',ot.numpy())
    print(' SimpleRNN output:'.format(t),
        output[0][t].numpy())
    print()
```

Output

```
Program to implement RNN
w_xh shape:  (5, 2)
w_oo shape:  (2, 2)
b_h shape:  (2,)
The weights of input to hidden layer:w_xh
[[ 0.08978081 -0.04088193]
 [ 0.43208182 -0.05285555]
 [-0.7771946   0.4738449 ]
 [ 0.21115029 -0.88302094]
 [ 0.36534548 -0.3919193 ]]
The weights of output to output layer:w_oo
[[ 0.18909991  0.9819579 ]
 [-0.9819579   0.18909979]]
The weights of bias of hidden layer:b_h
[0. 0.]
Input
tf.Tensor(
[[1. 1. 1. 1. 1.]
 [2. 2. 2. 2. 2.]
 [3. 3. 3. 3. 3.]], shape=(3, 5), dtype=float32)
It is for t= 0
Time step0=>
Input  xt=     : [[1. 1. 1. 1. 1.]]
It is  w_xh : [[ 0.08978081 -0.04088193]
 [ 0.43208182 -0.05285555]
 [-0.7771946   0.4738449 ]
 [ 0.21115029 -0.88302094]
 [ 0.36534548 -0.3919193 ]]
It is   b_h  : [0. 0.]
  ht=xt*w_xh+b_h    : [[ 0.32116377 -0.89483285]]
It is inside else
It is ht [[ 0.32116377 -0.89483285]]
```

```
tf.Tensor([[ 0.32116377 -0.89483285]], shape=(1, 2), dtype=float32)
It is prev_o [[0. 0.]]
It is w_oo [[ 0.18909991  0.9819579 ]
 [-0.9819579   0.18909979]]
It is ht [[ 0.32116377 -0.89483285]]
It is ot = ht+prev_o*w_oo  before tanh  [[ 0.32116377 -0.89483285]]
 output (manual) : [[ 0.31055883 -0.7137726 ]]
 SimpleRNN output: [ 0.31055883 -0.7137726 ]

It is for t= 1
Time step1=>
Input  xt=     : [[2. 2. 2. 2. 2.]]
It is  w_xh : [[ 0.08978081 -0.04088193]
 [ 0.43208182 -0.05285555]
 [-0.7771946   0.4738449 ]
 [ 0.21115029 -0.88302094]
 [ 0.36534548 -0.3919193 ]]
It is   b_h  : [0. 0.]
  ht=xt*w_xh+b_h    : [[ 0.64232755 -1.7896657 ]]
It is prev_o [[ 0.31055883 -0.7137726 ]]
It is prev_o [[ 0.31055883 -0.7137726 ]]
It is w_oo [[ 0.18909991  0.9819579 ]
 [-0.9819579   0.18909979]]
It is ht [[ 0.64232755 -1.7896657 ]]
It is ot = ht+prev_o*w_oo  before tanh  [[ 1.4019489 -1.6196842]]
 output (manual) : [[ 0.8857722 -0.9245783]]
 SimpleRNN output: [ 0.8857722 -0.9245783]

It is for t= 2
Time step2=>
Input  xt=     : [[3. 3. 3. 3. 3.]]
It is  w_xh : [[ 0.08978081 -0.04088193]
 [ 0.43208182 -0.05285555]
 [-0.7771946   0.4738449 ]
 [ 0.21115029 -0.88302094]
 [ 0.36534548 -0.3919193 ]]
It is   b_h  : [0. 0.]
  ht=xt*w_xh+b_h    : [[ 0.9634912 -2.6844985]]
It is prev_o [[ 0.8857722 -0.9245783]]
It is prev_o [[ 0.8857722 -0.9245783]]
It is w_oo [[ 0.18909991  0.9819579 ]
 [-0.9819579   0.18909979]]
It is ht [[ 0.9634912 -2.6844985]]
It is ot = ht+prev_o*w_oo  before tanh  [[ 2.0388875 -1.9895451]]
 output (manual) : [[ 0.9666744  -0.96328145]]
 SimpleRNN output: [ 0.9666744  -0.96328145]
```