```
########################################
# Shell package installer and checker #
########################################

%%shell

function echo_sep ( ) {
  echo "--------------------------------------------------------------------------------"
}

function install_pip( ) {
  install_res=1;
  echo "pip - installing package $1"
  pip3 -q install --upgrade --force-reinstall $1$2

  if pip list -v | grep $1; then
    echo "pip package $1 install success"
    install_res=0
  else
    echo "pip package $1 install failed"
  fi

  echo_sep

  return $install_res;
}

function install_all() {
  # Use && to consequently check package installation
  # pip install section
  install_pip 'tensorflow-gpu' &&
  echo "All packages installed" && echo_sep && echo
}


install_all
```

```
pip - installing package tensorflow-gpu
ERROR: tensorflow-metadata 0.27.0 has requirement absl-py<0.11,>=0.9, but you'll have
ERROR: nbclient 0.5.1 has requirement jupyter-client>=6.1.5, but you'll have jupyter-
ERROR: google-colab 1.0.0 has requirement google-auth~=1.17.2, but you'll have google
ERROR: google-colab 1.0.0 has requirement requests~=2.23.0, but you'll have requests
ERROR: datascience 0.10.6 has requirement folium==0.2.1, but you'll have folium 0.8.3
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have im
tensorflow-gpu                2.4.1           /usr/local/lib/python3.6/dist-packages
pip package tensorflow-gpu install success
--------------------------------------------------------------------------------
All packages installed
--------------------------------------------------------------------------------
```

```
# Modules
import tensorflow.compat.v1 as tf
import numpy                as np
```

```
import numpy                as np
import pandas               as pd
import os

from collections import Counter
from sklearn.datasets import fetch_20newsgroups

# config
os.environ[ 'TF_CPP_MIN_LOG_LEVEL' ] = '2'
tf.disable_v2_behavior() # req for compat.v1 and placeholder
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/comp
Instructions for updating:
non-resource variables are not supported in the long term
```

```
# Text process and classifier class
class TextClassifier:
  total_words = ""
  categories = []
  newsgroups_train = []
  newsgroups_test = []
  wti = {}


  def __init__( self ):
    return


  # Test
  def test_tf_graph( self ):
    tf_graph = tf.Graph()

    with tf.compat.v1.Session( graph = tf_graph ) as session:
      tf_x = tf.constant( [120, 380, 600 ] )
      tf_y = tf.constant( [80 , 120, 100 ] )
      tf_a = tf.add( tf_x, tf_y )
      tf_r = session.run( fetches = tf_a )

      print( tf_r )

      return


  def word_to_index(self, enum):
    wti = {}

    for i, word in enumerate(enum):
      wti[ word ] = i;

    return wti


  def category_to_vector( self, category ):
    y = np.zeros( ( 3 ), dtype = float )
```

```
  if category == 0:
    y[0] = 1.
  elif category == 1:
    y[1] = 1.
  else:
    y[2] = 1.

  return y


def batch( self, df, i, batch_size ):
  batches    = []
  results    = []
  texts      = df.data[   i * batch_size:i * batch_size + batch_size ]
  categories = df.target[ i * batch_size:i * batch_size + batch_size ]

  for text in texts:
      layer = self.text_to_vector( text )
      batches.append( layer )

  for category in categories:
      y = self.category_to_vector(category)
      results.append(y)

  return np.array( batches ), np.array( results )


# Text vectorisation
def text_to_vector( self, text ):
  layer = np.zeros( self.total_words, dtype = float )

  for word in text.split(' '):
    layer[ self.wti[ word.lower() ] ] += 1

  return layer


def download_data( self ):
  self.categories       = [ "comp.graphics", "sci.space", "rec.sport.baseball" ]
  self.newsgroups_train = fetch_20newsgroups( subset = 'train', categories = self.catego
  self.newsgroups_test  = fetch_20newsgroups( subset = 'test' , categories = self.catego

  return self


def process_data( self ):
  vocab = Counter() # Vocabulary counter

  # Input data word splitting
  for train_data in self.newsgroups_train.data:
    for word in train_data.split(' '): vocab[ word.lower() ] += 1

  for test_data in self.newsgroups_test.data:
    for word in test_data.split(' '): vocab[ word.lower() ] += 1
```

```
    self.wti = self.word_to_index( vocab ) # Converted result
    self.total_words = len( vocab )

    print( "Total words:", len( vocab )                                , '\n',
           'total texts in train:', len( self.newsgroups_train.data ), '\n',
           'total texts in test:' , len( self.newsgroups_test.data  ), '\n',
           'text'     , self.newsgroups_train.data[0]                  , '\n',
           'category:', self.newsgroups_train.target[0]                , '\n',
           "Each batch has 100 texts and each matrix has 119930 elements (words):"   , sel
           "Each batch has 100 labels and each matrix has 3 elements (3 categories):", sel

    return self


# Neural network class
class NeuralNetwork:

  def __init__( self ):
    self.classifier = TextClassifier()
    self.classifier.download_data().process_data() # Download and process data

    hid_layer_1  = 10         # Hiiden layer 1 - num of attribute
    hid_layer_2  = 5          # Hiiden layer 2 - num of attribute
    input_num    = self.classifier.total_words # Dictionary words
    class_num    = 3          # Category classes

    self.learn_rate   = 0.001 # Model learning rate
    self.train_epochs = 10    # Epochs iterations number
    self.batch_size   = 150   # Batch size
    self.display_step = 1     # Display step

    self.weights = {
      'h1'  : tf.Variable( tf.random.normal( [ input_num  , hid_layer_1 ] ) ),
      'h2'  : tf.Variable( tf.random.normal( [ hid_layer_1, hid_layer_2 ] ) ),
      'out' : tf.Variable( tf.random.normal( [ hid_layer_2, class_num   ] ) )
    }
    self.biases = {
      'b1'  : tf.Variable( tf.random.normal( [ hid_layer_1 ] ) ),
      'b2'  : tf.Variable( tf.random.normal( [ hid_layer_2 ] ) ),
      'out' : tf.Variable( tf.random.normal( [ class_num   ] ) )
    }


    # Tensor setup
    self.tf_input  = tf.placeholder( tf.float32, [ None, input_num ], name = "input"  )
    self.tf_output = tf.placeholder( tf.float32, [ None, class_num ], name = "output" )

    return



  def init_nn_model( self ):
    self.init_perceptrons()

    # Entropy && loss
```

```
    self.loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits( logits = self.per

    # Optimize
    self.optimizer = tf.train.AdamOptimizer( learning_rate = self.learn_rate ).minimize( s

    # Init tf
    self.init = tf.global_variables_initializer()

    return self

  def init_perceptrons( self ):
    layer_1_mult   = tf.matmul ( self.tf_input, self.weights[ 'h1' ]  )
    layer_1_add    = tf.add    ( layer_1_mult , self.biases[ 'b1' ]   )
    layer_1_act    = tf.nn.relu( layer_1_add                          )
    layer_2_mult   = tf.matmul ( layer_1_add  , self.weights[ 'h2' ]  )
    layer_2_add    = tf.add    ( layer_2_mult , self.biases[ 'b2' ]   )
    layer_2_act    = tf.nn.relu( layer_2_add                          )
    layer_out_mult = tf.matmul ( layer_2_act  , self.weights[ 'out' ] )

    self.perceptrons_result = layer_out_mult + self.biases[ 'out' ]

    return self


  # Prediction
  def init_prediction( self ):
    # Define category
    data_categories = self.classifier.newsgroups_test.data[5]

    print( 'text', data_categories, '\n',
           "Text correct category:", self.classifier.newsgroups_test.target[5] )

    # Vectorization
    data_vector = self.classifier.text_to_vector( data_categories )

    # Transfrom to NP array
    input_arr = np.array( [ data_vector ] )
    input_arr.shape

    # Init saver
    self.model_save()

    # Restore model
    with tf.Session() as session:
      self.saver.restore( session, self.save_file)
      print( "Model model.ckpt restored." )

      classification = session.run( tf.argmax( self.perceptrons_result, 1 ), feed_dict = {
      print( "Predicted category: ", classification )

    # Define N categories
    n = 10
    x_n_texts, y_n_correct_labels = self.classifier.batch( self.classifier.newsgroups_test

    self.model save()
```

```
  # Restore model
  with tf.Session() as session:
    self.saver.restore(session, self.save_file)
    print( "Model restored." )

    classification = session.run( tf.argmax( self.perceptrons_result, 1 ), feed_dict = {
    print( "Predicted categories:", classification, '\n'
           "Correct categories:", np.argmax( y_n_correct_labels, 1 ) )

  return self


# Initial model train
def model_graph_calc( self ):
  with tf.Session() as session:
    session.run( self.init )

    # Cycle training
    for epoch in range( self.train_epochs ):
      avg_cost = 0.
      batch    = int( len( self.classifier.newsgroups_train.data ) / self.batch_size )
      # Elements iteration
      for i in range( batch ):
        batch_x, batch_y = self.classifier.batch( self.classifier.newsgroups_train, i, s

        # Optimisation run
        c, _ = session.run( [ self.loss, self.optimizer ], feed_dict = { self.tf_input:
                                                                         self.tf_output:

        # Error calculation
        avg_cost += c / batch

      # Log ouput
      if epoch % self.display_step == 0:
        print( "Epoch:", '%04d' % ( epoch + 1 ), "loss=", "{:.9f}".format( avg_cost ) )
        print( "Optimization Finished!" )

    # Check
    correction = tf.equal( tf.argmax( self.perceptrons_result, 1 ), tf.argmax( self.tf_o

    # Error accuracy calculation
    accuracy = tf.reduce_mean( tf.cast( correction, "float" ) )
    test_result = len( self.classifier.newsgroups_test.target )

    batch_x_test, batch_y_test = self.classifier.batch( self.classifier.newsgroups_test,

    print("Accuracy:", accuracy.eval( { self.tf_input:  batch_x_test,
                                        self.tf_output: batch_y_test } ) )

    # Save trained model
    self.save_file = "model.ckpt"
    self.save_path = self.saver.save( session, self.save_file )
    print("Model saved in path: %s" % self.save_path)
```

```
      return self

  # Save trained model
  def model_save( self ):
    self.saver = tf.train.Saver();

    return


# Test tf graph
TextClassifier().test_tf_graph()

     [200 500 700]


# Init neural network model
NN = NeuralNetwork()
NN.init_nn_model().model_save();

     Total words: 119930
      total texts in train: 1774
      total texts in test: 1180
      text From: jk87377@lehtori.cc.tut.fi (Kouhia Juhana)
     Subject: Re: More gray levels out of the screen
     Organization: Tampere University of Technology
     Lines: 21
     Distribution: inet
     NNTP-Posting-Host: cc.tut.fi

     In article <1993Apr6.011605.909@cis.uab.edu> sloan@cis.uab.edu
     (Kenneth Sloan) writes:
     >
     >Why didn't you create 8 grey-level images, and display them for
     >1,2,4,8,16,32,64,128... time slices?

     By '8 grey level images' you mean 8 items of 1bit images?
     It does work(!), but it doesn't work if you have more than 1bit
     in your screen and if the screen intensity is non-linear.

     With 2 bit per pixel; there could be 1*c_1 + 4*c_2 timing,
     this gives 16 levels, but they are linear if screen intensity is
     linear.
     With 1*c_1 + 2*c_2 it works, but we have to find the best
     compinations -- there's 10 levels, but 16 choises; best 10 must be
     chosen. Different compinations for the same level, varies a bit, but
     the levels keeps their order.

     Readers should verify what I wrote... :-)

     Juhana Kouhia

      category: 0
      Each batch has 100 texts and each matrix has 119930 elements (words): (100, 119930)
      Each batch has 100 labels and each matrix has 3 elements (3 categories): (100, 3)
     WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/util
     Instructions for updating:

     Future major versions of TensorFlow will allow gradients to flow
     into the labels input on backprop by default.
```

```
See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

```
# Init graph and prediction
NN.model_graph_calc().init_prediction()
```

```
Epoch: 0002 loss= 57.797368136
Optimization Finished!
Epoch: 0003 loss= 50.210497076
Optimization Finished!
Epoch: 0004 loss= 44.041845148
Optimization Finished!
Epoch: 0005 loss= 39.211533460
Optimization Finished!
Epoch: 0006 loss= 35.128347917
Optimization Finished!
Epoch: 0007 loss= 31.560690793
Optimization Finished!
Epoch: 0008 loss= 28.516226335
Optimization Finished!
Epoch: 0009 loss= 25.895324360
Optimization Finished!
Epoch: 0010 loss= 23.511588877
Optimization Finished!
Accuracy: 0.3779661
Model saved in path: model.ckpt
text From: neff@iaiowa.physics.uiowa.edu (John S. Neff)
Subject: Re: Space spinn offs
Nntp-Posting-Host: pluto.physics.uiowa.edu
Organization: The University of Iowa
Lines: 23

In article <1rruis$9do@bigboote.WPI.EDU> wfbrown@wpi.WPI.EDU (William F Brown) wri
>From: wfbrown@wpi.WPI.EDU (William F Brown)
>Subject: Re: Space spinn offs
>Date: 30 Apr 1993 19:27:24 GMT
>I just wanted to point out, that Teflon wasn't from the space program.
>It was from the WWII nuclear weapons development program.  Pipes in the
>system for fractioning and enriching uranium had to be lined with it.
>
>Uranium Hexafloride was the chemical they turned the pitchblend into for
>enrichment.  It is massively corrosive.  Even to Stainless steels. Hence
>the need for a very inert substaance to line the pipes with.  Teflon has
>all its molecular sockets bound up already, so it is very unreactive.
>
>My 2 sense worth.
>
>Bill
>

The artifical pacemaker was invented in 1958 by Wilson Greatbatch an
American biomedical engineer. The bill authorizing NASA was signed
in October of 1958 so it is clear that NASA had nothing to do with
the invention of the pacemaker.


 Text correct category: 2
INFO:tensorflow:Restoring parameters from model.ckpt
Model model.ckpt restored.
```

```
Predicted category:  [2]
INFO:tensorflow:Restoring parameters from model.ckpt
Model restored.
Predicted categories: [1 1 2 1 1 2 2 2 2 2]
Correct categories: [1 0 1 0 1 2 2 0 1 2]
```