<pre>import sys import shuti import numpy from numpy.r import matpl import sklea import tenso import keras from keras if from keras.u from keras.m from keras.l from keras.c from sklearn import pydot # Summarize print('Libra print('- Pyt print('- Num print('- Mat)</pre>	<pre>as np # Work with multi-dimensional arrays and matrices. andom import seed otlib as mpl # Create 2D charts. otlib.pyplot as plt In # Perform data mining and analysis. arflow # Train neural networks for deep learning. # Provide a frontend for TensorFlow. Import datasets tils import to_categorical, plot_model todels import Sequential, Model ayers import AveragePooling2D, BatchNormalization, Conv2D, MaxPooling2D, Dense, Flatten tallbacks import EarlyStopping model_selection import train_test_split software libraries used. tries used in this project:') hon {}'.format(sys.version)) try {}'.format(npversion_)) plotlib {}'.format(mplversion_)) plotlib {}'.format(mplversion_))</pre>	
# Load the d #shutil.rmtr #shutil.copy (X_train, y_ print('Loade print('Loade # Uncomment seed(1) tensorflow.r Libraries use - Python 3.8 NumPy 1.19 Matplotlib - scikit-lear - TensorFlow	Libraries used in this project: - Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] - NumPy 1.19.2 - Matplotlib 3.3.2 - scikit-learn 0.23.2 - TensorFlow 2.4.0 - Keras 2.4.3	
In [2]: # Show dimen X_train.shap Out[2]: ((60000, 28, In [3]: X_train[0].s Out[3]: (28, 28) Visualize	e the data examples	
<pre>W = 4 L = 5 fig, axes = axes = axes. n_training = for i in ran index = axes[i]. #axes[i] axes[i].</pre>	<pre>iden(X_train) idge(W*L): np.arange(0, n_training) imshow(X_train[i]) .set_title(y_train[index],fontsize=15) axis("off") ids_adjust(hspace=0.1)</pre>	
5		
3	 4 6 7 8 9 	
<pre>In [5]: # Reshape ar</pre>	28, 1)	
<pre>In [10]: X_test_scale In [11]: y_train_enco In [12]: y_test_encod In [13]: X_train_scal Out[13]: ((60000, 28, In [14]: X_train_scal Out[14]: array([[[0.</pre>	1, 1, 1, 1, 1, 1,	
[0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	<pre>],],],],],],],],],],],],],</pre>	
[0. [0] [0] [0] [0] [0] [0] [0] [0]	<pre> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,</pre>	
[0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	<pre> </pre>	
[0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	<pre> </pre>	
[0. [0. [0. [0. [0. [0. [0. [0.	<pre> </pre>	
[0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	<pre>],],],],],],],],],],],],],</pre>	
[0.07 [0.07 [0.07 [0.49 [0.53 [0.68],]],],],],],],],],],],],],	
[1. [0.96] [0.49] [0. [0. [0. [0. [0. [0. [0. [0.	[3098039], [3080392], [3080392], [3080392], [3080392], [3080392], [3080392], [3080392], [3080392], [3080392], [3080392], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [308039], [30803],	
[0.99 [0.99 [0.88 [0.67 [0.99 [0.94 [0.76 [0.25 [0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215688], 3215688], 3215688], 3215688], 3215688], 3215688], 3215688], 3215688], 3215688], 3215888], 3215888], 3215888], 3215888], 3215888], 3215888], 3215888], 3215888], 3215888], 3215888],	
[0.99 [0.99 [0.99 [0.99 [0.99 [0.99 [0.99 [0.98 [0.36 [0.32 [0.32 [0.32 [0.15 [0.15 [0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 2215686], 221588], 221588], 2315863]	
[0.85 [0.99 [0.99 [0.99 [0.99 [0.77 [0.71],],], [058824], [882353], [2215686], [2215686], [2215686], [2215686], [2315686], [2415686], [2547059], [372549], [3862745], [3862	
[0.61 [0.41 [0.99 [0.99 [0.80 [0.04 [0. [0.16 [0.60],]],],],],],],],],],],],],	
[0.00 [0.60 [0.99		
[0.	<pre> ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;</pre>	
[0. [0. [0.54 [0.99],],],],],],],],],],],],],]	
[0. [0. [0. [0. [0. [0. [0. [0.04],],],],],],], [1313725], [1509804], [1215686], [145098],],],],],],],],],],],],],]	
[0.94 [0.88 [0.62 [0.42],]],],],],],],],],],],],],	
[0. [0. [0. [0. [0. [0. [0. [0.],],],],],],],],],],],],],]	
[0.99 [0.99 [0.46	2215686], 2215686], 2215686], 3666667], 3803922], , , , , , , , , , , , ,	
[0. [0. [0. [0. [0.17 [0.72 [0.99		
[0.36 [0.98 [0.99 [0.73 [0. [0. [0.],],],],],],],],],], [,], [, 470588], 823529], 9215686], 3333333],],],],],],],],],],],],],]	
[0. [0. [0. [0. [0. [0. [0. [0.],]],],],],],],],],],],],],	
[0. [0. [0.] [0],],],],],],],],],],],],],]	
[0.50 [0.71 [0.99 [0.99 [0.81 [0.00 [0. [0. [0. [0. [0. [0. [0. [0. [9880392], 7.64706], 9215686], 9215686], 176471], 1784314],	
[0. [0.]		
[0. [0. [0. [0. [0. [0.99 [0.99 [0.99 [0.99 [0.99		
[0. [0. [0. [0. [0. [0. [0. [0.],],],],],],],],],], [,], [,], [,], [,], [,], [,], [,], [,], [,], [,], [,], [,], [,], [,], [,],], [,],],],],],],],],],],],],],	
[0. [0. [0. [0. [0. [0. [0. [0.],],],],],],],],],],],],], [058824], [058824], [0882353], [0215686], [215686], [215686], [215686], [215686], [21568],	
[0.		
[0.99 [0.95] [0.52] [0.04] [0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	2215686], 6886275], 2156863], 1313725], , , , , , , , , , , , ,	
[0.53 [0.99 [0.99 [0.83 [0.52 [0.51 [0.06 [0. [0. [0. [0. [0. [0. [0. [0. [0. [0.	3333333], 2215686], 2215686], 2215686], 3337255], 2941176], 764706], 527451],	
[0. [0. [0. [0. [0. [0. [0. [0.	<pre>11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,</pre>	
[0. [0. [0. [0. [0. [0. [0. [0.	<pre> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,</pre>	

[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 [15]: y_train_encoded[0] Out[15]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32) Split the datasets In [16]: # Split the training and validation datasets and their labels. from sklearn.model_selection import train_test_split X_train, X_val, y_train, y_val = train_test_split(X_train_scaled, y_train_encoded, random_state = 1) In [18]: X_train.shape, X_val.shape, y_train.shape, y_val.shape Out[18]: ((45000, 28, 28, 1), (15000, 28, 28, 1), (45000, 10), (15000, 10)) **Build the CNN structure** In [19]: # Import the required libraries. # Create the model. # Add model layers as specified. model = Sequential() model.add(Conv2D(filters=32, kernel_size = (3, 3), input_shape = (28, 28, 1), padding = 'same', activation=" model.add(MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid')) model.add(Flatten()) model.add(Dense(512, activation="relu")), model.add(Dense(10, activation="softmax")) In [21]: model.summary() Model: "sequential" Output Shape Layer (type) Param # conv2d (Conv2D) (None, 28, 28, 32) 320 max_pooling2d (MaxPooling2D) (None, 27, 27, 32) flatten (Flatten) (None, 23328) 0 dense (Dense) (None, 512) 11944448 dense_1 (Dense) (None, 10) 5130 Total params: 11,949,898 Trainable params: 11,949,898 Non-trainable params: 0 Compile the model and summarize the layers In [22]: from keras.optimizers import Adam In [23]: # Compile the model. # - Use the adam optimizer. # - Use categorical_crossentropy for the loss function. # - Use accuracy to measure model performance. optimizer = Adam(0.001) # Summarize the layers. model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy']) Plot a graph of the model In [24]: # Install the required library. #!conda install --yes graphviz==2.40.1 # Plot a graph of the model. In [26]: plot_model(model, show_shapes=True) Out[26]: [(None, 28, 28, 1)] input: conv2d_input: InputLayer [(None, 28, 28, 1)] output: (None, 28, 28, 1) input: conv2d: Conv2D (None, 28, 28, 32) output: (None, 28, 28, 32) input: max_pooling2d: MaxPooling2D (None, 27, 27, 32) output: (None, 27, 27, 32) input: flatten: Flatten (None, 23328) output: (None, 23328) input: dense: Dense (None, 512) output: input: (None, 512) dense_1: Dense output: (None, 10) Train the model In [27]: # Train the model over 1 epoch. history = model.fit(X_train,y_train, validation_data = (X_val, y_val), steps_per_epoch=100, batch_size=600, 75/100 [=========>.....] - ETA: 22s - loss: 0.5796 - accuracy: 0.8097WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least $`steps_per_epoch * epochs` batches (in this case, 100 batches). You may need to use the repeat() function wh$ en building your dataset. 30 - val_accuracy: 0.9674 Evaluate the model on the test data In [28]: # Evaluate the model on the test data, showing loss and accuracy. model.evaluate(X_test,y_test_encoded,batch_size=600) Out[28]: [23.127225875854492, 0.9470999836921692] Make predictions on the test data In [29]: # Make predictions on the test data. #model.predict_classes(X_test_scaled, batch_size=600) # Show the first 30 examples. prediction = np.argmax(model.predict(X_test_scaled, batch_size=600),axis=-1) In [33]: prediction[:30] Out[33]: array([7, 2, 1, 0, 4, 1, 4, 9, 6, 9, 0, 6, 9, 0, 1, 5, 9, 7, 8, 4, 9, 6, 6, 5, 4, 0, 7, 4, 0, 1], dtype=int64) In [34]: y_test[:30] Out[34]: array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, 9, 6, 6, 5, 4, 0, 7, 4, 0, 1], dtype=uint8) Visualize the predictions for 30 examples In [35]: import pandas as pd In [36]: pred = pd.DataFrame(prediction[:30],columns=["Prediction"]) actual = pd.DataFrame(y_test[:30],columns=["Actual"]) df = pd.concat([actual,pred],axis=1) In [39]: df Out[39]: **Actual Prediction** 0 7 7 1 2 1 1 0 4 4 1 5 4 6 7 9 9 8 5 6 9 0 10 0 11 6 9 12 9 13 1 14 1 15 9 9 16 17 18 3 8 19 9 20 9 21 6 22 6 23 4 24 25 7 26 27 28 0 Predict Index on left, Actual Index on right figure = plt.figure(figsize=(20, 8)) In [40]: for i, index in enumerate(np.random.choice(X_test.shape[0], size=15, replace=False)): ax = figure.add_subplot(3, 5, i + 1, xticks=[], yticks=[]) # Affichez chaque image ax.imshow(np.squeeze(X_test[index])) predict_index = prediction[index] true_index = y_test[index] # définir le titre de chaque image ax.set_title("{} ({{}})".format([predict_index], [true_index]), color=("green" if predict_index == true_index else "red") [6] ([6]) [9] ([9]) [2]([2]) [6] ([6]) [7] ([7]) [6] ([6]) [4] ([4]) [7] ([7]) [1]([1]) [1]([1]) [1]([1]) Loading [MathJax]/extensions/Safe.js