Quick, Draw! Doodle Recognition Challenge

By: Dake Shi

Overview

- "Quick, Draw!" was released as an experimental game to educate the public in a playful way about how AI works. The game prompts users to draw an image depicting a certain category, such as ”banana,” “table,” etc. The game generated more than 1B drawings, of which a subset was publicly released as the basis for this competition’s training set. That subset contains 50M drawings encompassing 340 label categories.
- Sounds fun, right? Here's the challenge: since the training data comes from the game itself, drawings can be incomplete or may not match the label. You’ll need to build a recognizer that can effectively learn from this noisy data and perform well on a manually-labeled test set from a different distribution.
- Your task is to build a better classifier for the existing Quick, Draw! dataset. By advancing models on this dataset, Kagglers can improve pattern recognition solutions more broadly. This will have an immediate impact on handwriting recognition and its robust applications in areas including OCR (Optical Character Recognition), ASR (Automatic Speech Recognition) & NLP (Natural Language Processing).

- Reference: https://www.kaggle.com/c/quickdraw-doodle-recognition

Data

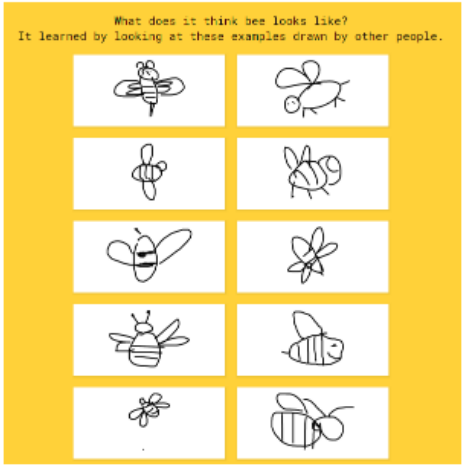
- The Quick Draw Dataset is a collection of millions of drawings across 300+ categories, contributed by players of Quick, Draw! The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located.

- test_raw.csv - the test data in the raw vector format
- test_simplified.csv - the test data in the simplified vector format
- train_raw.zip - the training data in the raw vector format; one csv file per word
- train_simplified.zip - the training data in the simplified vector format; one csv file per word

Data Continue

This is one data example. What I used in this project are column of drawing and column of word, which representative Picture dot matrix and label.

	A	B	C	D	E	F
1	countrycode	drawing	key_id	recognized	timestamp	word
2	US	[[[111, 148, 161, 175, 19	5.15991E+15	TRUE	02:16.2	alarm clock
3	US	[[[154, 144, 129, 86, 66,	4.60809E+15	TRUE	42:04.7	alarm clock
4	US	[[[26, 15, 1, 2, 12, 27,	5.50211E+15	TRUE	52:35.4	alarm clock
5	US	[[[181, 174, 133, 110, 5	6.60068E+15	TRUE	40:58.0	alarm clock
6	CZ	[[[120, 86, 46, 29, 24, 2	6.34494E+15	TRUE	40:59.2	alarm clock
7	GB	[[[183, 144, 113, 98, 69,	4.66517E+15	TRUE	20:50.8	alarm clock
8	US	[[[104, 76, 62, 48, 10, 6	6.6337E+15	TRUE	31:57.8	alarm clock
9	US	[[[152, 142, 129, 81, 67,	5.53396E+15	TRUE	43:08.2	alarm clock
10	IT	[[[83, 67, 53, 36, 27, 14	6.29793E+15	TRUE	45:19.3	alarm clock
11	CZ	[[[28, 19, 3, 0, 3, 12, 3	6.30594E+15	TRUE	34:00.2	alarm clock
12	IQ	[[[7, 19, 39, 67, 98, 129	5.5354E+15	TRUE	33:11.6	alarm clock
13	US	[[[91, 54, 20, 10, 12, 17	6.16694E+15	TRUE	39:05.1	alarm clock
14	DE	[[[103, 82, 66, 48, 32, 2	5.3498E+15	TRUE	49:07.9	alarm clock
15	SK	[[[5, 9, 23, 79, 90, 145,	5.06008E+15	TRUE	12:55.6	alarm clock
16	US	[[[151, 135, 110, 68, 37,	5.65538E+15	TRUE	41:40.7	alarm clock
17	TW	[[[171, 143, 104, 64, 40,	4.56752E+15	TRUE	49:46.5	alarm clock
		[[148, 118, 81, 48, 25,	5.82676E+15	TRUE	48:21.9	alarm clock
		[[116, 98, 64, 58, 36, 3	4.93805E+15	TRUE	15:24.4	alarm clock
		[[65, 75, 81, 86, 87, 64	4.91452E+15	TRUE	04:47.0	alarm clock
		[[227, 254, 254, 246, 21	4.90909E+15	TRUE	04:06.8	alarm clock
		[[110, 90, 76, 54, 49, 5	5.05012E+15	TRUE	04:12.2	alarm clock
		[[53, 13, 6, 1, 0, 3, 22	6.34778E+15	TRUE	59:20.8	alarm clock
		[[183, 171, 149, 134, 10	5.62968E+15	TRUE	31:28.9	alarm clock
		[[193, 185, 178, 163, 14	4.95974E+15	TRUE	13:44.6	alarm clock
		[[84, 68, 53, 36, 25, 20	6.54329E+15	TRUE	46:43.9	alarm clock
		[[175, 173, 157, 132, 11	6.11118E+15	TRUE	54:51.0	alarm clock
		[[116, 112, 96, 50, 28,	4.54072E+15	TRUE	43:14.0	alarm clock
		[[54, 37, 28, 19, 11, 9,	5.81061E+15	TRUE	52:35.1	alarm clock
		[[90, 83, 37, 27, 11, 4,	5.00752E+15	TRUE	43:18.3	alarm clock
		[[115, 100, 82, 53, 27,	6.49919E+15	TRUE	38:32.6	alarm clock
		[[5, 24, 32, 43, 56, 77,	6.38686E+15	FALSE	06:00.1	alarm clock
		[[183, 159, 146, 136, 7	5.19961E+15	TRUE	50:21.7	alarm clock
		[[34, 75, 135, 155, 180,	5.41797E+15	FALSE	37:02.0	alarm clock
		[[150, 124, 110, 99, 80,	6.55332E+15	TRUE	10:45.9	alarm clock



Read The Data

```
[ ]  train = pd.DataFrame()
     for file in os.listdir('/gdrive/My Drive/Project_Data/train_simplified/')[:10]:
         train = train.append(pd.read_csv('/gdrive/My Drive/Project_Data/train_simplified/' + file, usecols=[1, 5], nrows=600))


     train = shuffle(train, random_state=123)
     print(train)
```

```
124  [[[39, 33, 35], [133, 113, 215]], [[38, 31, 68...  animal migration
593  [[[0, 101, 162, 155, 158, 172, 207, 225, 244, ...         ambulance
493  [[[143, 177, 200], [24, 7, 36]], [[74, 114, 11...  animal migration
383  [[[9, 8, 0], [14, 8, 0]], [[80, 114, 117, 118,...  animal migration
391  [[[45, 31, 30, 36, 46, 61, 58, 46, 38, 38, 42,...  animal migration
61   [[[24, 26, 33, 30, 23, 7, 0, 3, 15, 27, 40, 50...               ant
373  [[[72, 54, 45, 36, 46, 84, 134, 156, 170, 175,...       alarm clock
299  [[[81, 42, 28, 11, 6, 6, 11], [0, 33, 50, 82, ...               arm
92   [[[85, 89, 96, 126, 135, 141, 140, 134, 125, 9...             angel
523  [[[185, 171, 140, 87, 56, 33, 30, 44, 63, 94, ...       alarm clock
264  [[[30, 54, 88, 93, 93, 86, 67, 45, 0, 7], [42,...  animal migration
398  [[[0, 24, 30, 42, 47], [62, 45, 46, 54, 52]], ...  animal migration
366  [[[46, 49, 47, 33, 23, 26, 37, 42, 51, 54, 65,...         asparagus
146  [[[10, 8, 11, 45, 126, 208, 249, 255, 242, 231...       alarm clock
92   [[[8, 15, 28, 29, 54, 59, 59, 74, 74, 60, 60, ...         asparagus
468  [[[111, 115, 113, 100, 64, 36, 21, 10, 2, 0, 6...         asparagus
556  [[[44, 11, 21, 53, 92, 105, 117, 116, 111, 78,...         asparagus
..                                                 ...               ...
28   [[[54, 39], [83, 131]], [[49, 79], [81, 130]],...  animal migration
595  [[[86, 70, 50, 48, 48, 193, 194, 189, 148], [1...             anvil
97   [[[87, 76, 71, 76, 95, 106, 129, 136, 136, 132...             angel
559  [[[112, 110, 119, 135], [44, 21, 16, 15]], [[1...       alarm clock
279  [[[41, 119, 202, 247, 255, 243, 203, 181, 147,...         asparagus
597  [[[43, 59, 72, 97], [96, 148, 204, 255]], [[57...         asparagus
39   [[[27, 6, 0, 3, 24, 53, 203, 204, 209, 222, 22...          airplane
250  [[[88, 78, 57, 0, 29, 135, 164, 166, 165, 157,...             angel
```

Get the class and Labels for each features

```
[ ]  train['word'] = train['word'].replace(' ', '_', regex=True)
     #replece the whitespace to '_' then the label will be a words
     classes_names = train['word'].unique()#get the class name, I only read 10 csv file, so there are 10 classed
     labels = pd.get_dummies(train['word']).values
     print(len(classes_names))
     print(len(labels))
```

```
10
6000
```

Using Opencv to transfer node vectors to NP array

```
[ ]  def drawing_to_np(drawing, shape=(28, 28)):
         # evaluates the drawing array
         drawing = eval(drawing)
         fig, ax = plt.subplots()
         for x,y in drawing:
             ax.plot(x, y, marker='.')
             ax.axis('off')
         fig.canvas.draw()
         # Close figure so it won't get displayed while transforming the set
         plt.close(fig)
         # Convert images to numpy array
         np_drawing = np.array(fig.canvas.renderer._renderer)
         # Take only one channel
         np_drawing =np_drawing[:, :, 1]
         # Normalize data
         np_drawing = np_drawing / 255.
         return cv2.resize(np_drawing, shape) # Resize array
```

```
[ ]  train.drop(['word'], axis=1, inplace=True)
     # Transform drawing into numpy arrays
     train['drawing_np'] = train['drawing'].apply(drawing_to_np)
     # Reshape arrays
     train_drawings = np.asarray([x.reshape(28,28,1) for x in train['drawing_np'].values])
```

[Matplotlib canvas as numpy array artefacts]

Reference:https://stackoverflow.com/questions/50437426/matplotlib-canvas-as-numpy-array-artefacts

Get train, val data and those labels

From sklearn.model_selection import train_test_split

This is a super powerful class to split data showed above to train data, val_data, train labels and val_labels.

```
[ ]  #y_train and y_val are the labels,x_train and x_val are data
     x_train, x_val, y_train, y_val = train_test_split(train_drawings, labels, test_size=0.1, random_state=1)
```

Train model and model summary

```
[ ]  from keras import layers
     from keras import models


     model = models.Sequential()
     #model.add(conv_base)
     model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(28, 28, 1)))
     model.add(layers.Conv2D(32, (3, 3), activation='relu'))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Dropout(0.25))
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Dropout(0.25))
     model.add(layers.Conv2D(128, (3, 3), activation='relu'))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Flatten())
     model.add(layers.Dropout(0.25))
     model.add(layers.Dense(512, activation='relu'))
     model.add(layers.Dense(10, activation='softmax'))
```

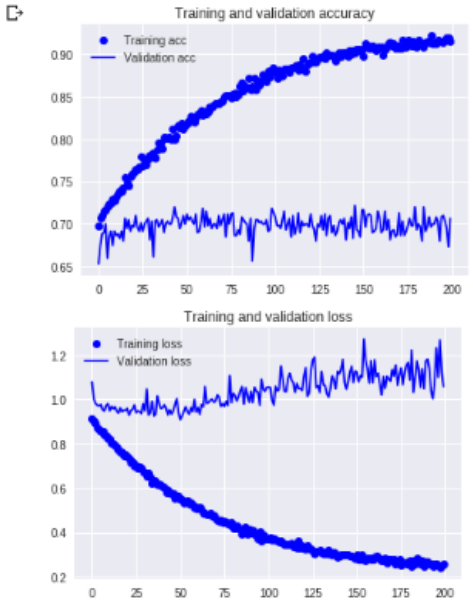
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 32)	0
dropout_1 (Dropout)	(None, 12, 12, 32)	0
conv2d_3 (Conv2D)	(None, 10, 10, 64)	18496
conv2d_4 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
conv2d_5 (Conv2D)	(None, 2, 2, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 1, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 512)	66048
dense_2 (Dense)	(None, 10)	5130

Total params: 210,026
Trainable params: 210,026
Non-trainable params: 0

```
[ ]  from keras import optimizers

     model.compile(loss='categorical_crossentropy',
                   optimizer=optimizers.RMSprop(lr=1e-4),
                   metrics=['acc'])
```

Result

```
[ ]  acc = history.history['acc']
     val_acc = history.history['val_acc']
     loss = history.history['loss']
     val_loss = history.history['val_loss']

     epochs = range(len(acc))

     plt.plot(epochs, acc, 'bo', label='Training acc')
     plt.plot(epochs, val_acc, 'b', label='Validation acc')
     plt.title('Training and validation accuracy')
     plt.legend()

     plt.figure()

     plt.plot(epochs, loss, 'bo', label='Training loss')
     plt.plot(epochs, val_loss, 'b', label='Validation loss')
     plt.title('Training and validation loss')
     plt.legend()

     plt.show()
```



Thanks