UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Doodle Recognition using Convolutional Neural Networks

SCS_3253_024 Machine Learning

Group 8

Nitin Prakash Panicker

✦

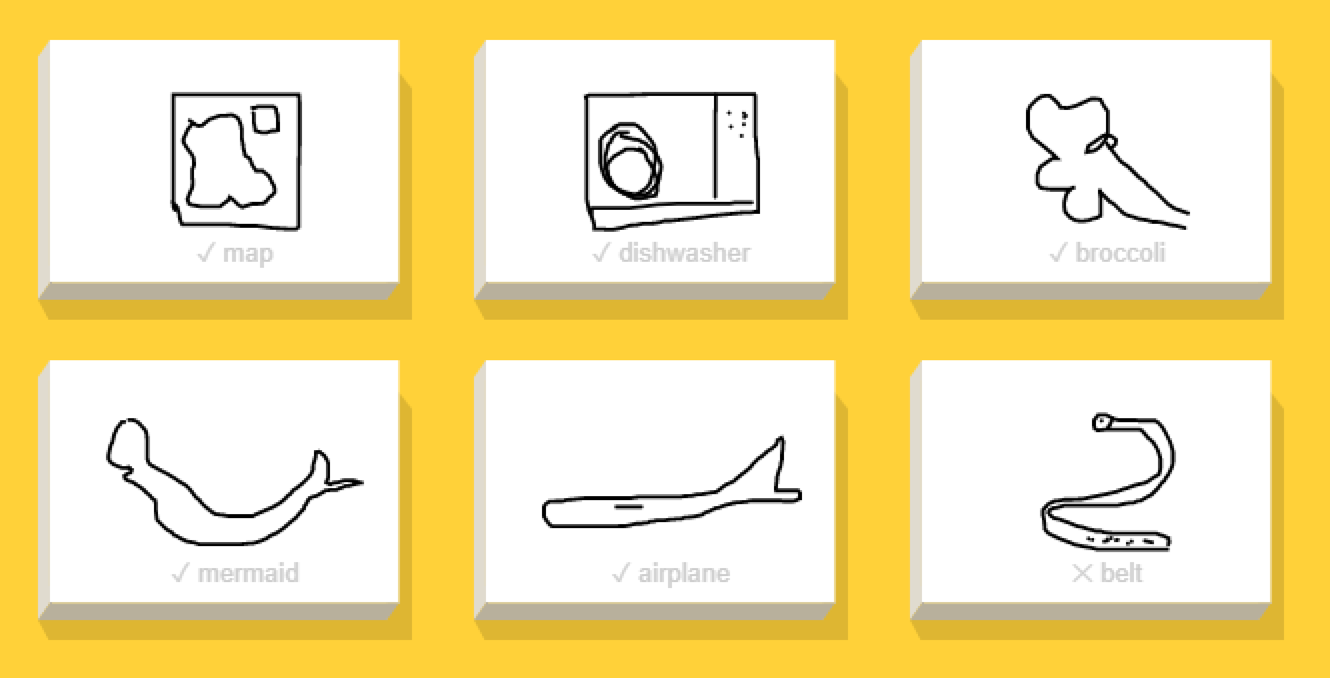
Syed Reza

Hello!

We want to explore the Quick-Draw dataset.

Quick-draw is a ‘neural network game, trying to guess what the user is drawing.

This is what Quick, Draw! Look Like



Clicking on this image will take you to the game.

How would you like to explore our project?

GitHub

YouTube

or, continue with this presentation…

First 2 inks will take you to external websites.

1.0 Business Objectives

Why are we interested in this project?

Discovering this doodle dataset will extend opportunities to explore implementations in real life projects like, handwritten text recognition, touch based complex interactions, etc.

“

Knowledge from this project work will help understand challenges for human computer interaction in touch-based applications.

“

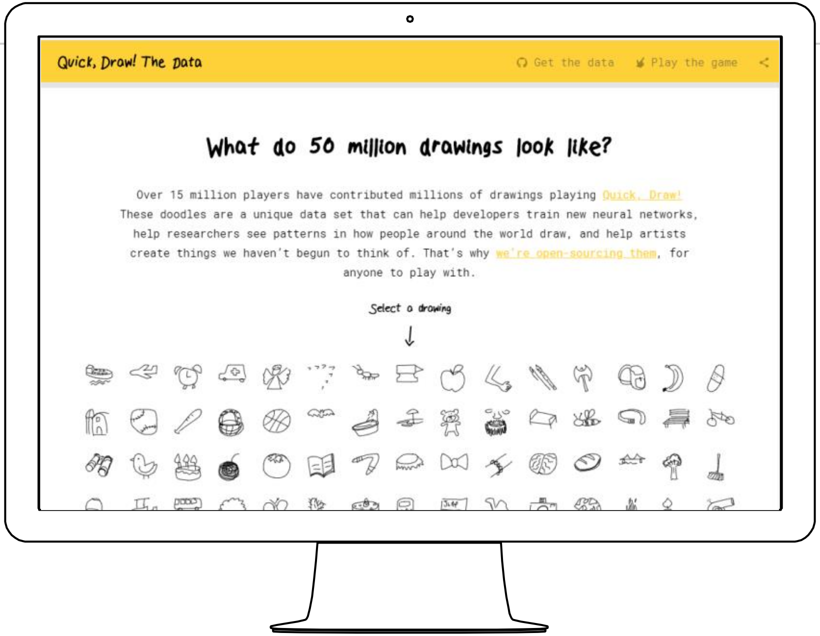
2.0

The Dataset

What do 50 million drawings look like?

Quick, Draw! The Data

Millions of players have contributed huge doodle drawings playing this game! We decided to use these doodles to train our convolutional neural network.



Clicking on this image will take you to the sample dataset.

Introduction to the Dataset

- 50 million drawings
- Classified into 345 classes
- 20 classes of drawings with first 10,000 image samples taken for data analysis and wrangling
- For CNN data modelling, we have used all 345 classes, but first 10,000 images

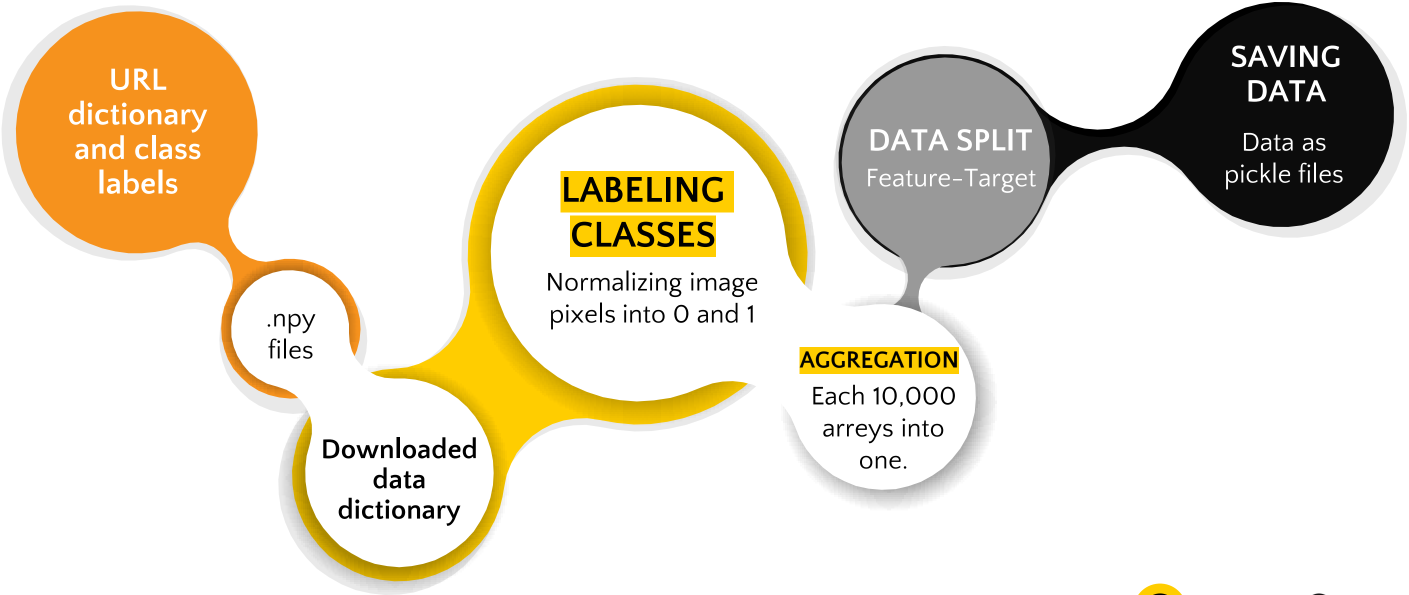
Where to get the data?

- Entire dataset is available in GitHub:
  https://github.com/googlecreativelab/quickdraw-dataset
- We are working with preprocessed dataset provided by Google Creative Lab.
- All the drawings have been rendered into a 28x28 px grayscale bitmap in numpy format.

2.0 Data Analysis

How data cleansing and wrangling were done?

Data Cleansing Process



Link to GitHub source.

Data Wrangling

- Created a dictionary for URL and class labels to download the entire dataset in .npy files.
- Then the dataset was downloaded into a dictionary.
- Normalized the image pixels between 0 and 1 and added class labels.

Data Wrangling

- Aggregated the first 10,000 arrays for every doodle into one array of doodles.
- Split the data into features and target labels.
- Save the train and test data as serialized data using pickle files.

2.1

Analysis Challenges

Colab couldn’t handle gigantic dataset!

Encountered a 'Memory Error' while trying to create a dictionary object using the downloaded data in Colab.

“

Memory Error

In [130]:

```
%%time

classes_dict = {}
for key, value in URL_DATA.items():
    response = requests.get(value)
    classes_dict[key] = np.load(BytesIO(response.content))
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<timed exec> in <module>

~\AppData\Local\Continuum\anaconda3\envs\nitin\lib\site-packages\requests\api.py in get(url, param
s, **kwargs)
     74
     75     kwargs.setdefault('allow_redirects', True)
---> 76     return request('get', url, params=params, **kwargs)
     77
     78
```

Running the code in Colab crashed the application and produced a RAM *error as well. Then solved by looping through every .npy file and .png image data for every .npy file to set up data for 345 samples to create our CNN model.*

“

Generating PNG by Looping all .npy

```python
%%time

# Looping through every npy file and generating png image data for every npy file.
for filename in os.listdir(data_file_path):
    if filename.endswith('.npy'):
        data = np.load(data_file_path+"/"+filename)
        print(filename)
        data = data[0:,:]

    if not os.path.exists(train_path+os.path.splitext(filename)[0]):
        try:
            os.makedirs(train_path+os.path.splitext(filename)[0])
        except OSError as exc:
            if exc.errno != errno.EEXIST:
                raise
    print(train_path+os.path.splitext(filename)[0])

    if not os.path.exists(test_path+os.path.splitext(filename)[0]):
        try:
            os.makedirs(test_path+os.path.splitext(filename)[0])
        except OSError as exc:
            if exc.errno != errno.EEXIST:
                raise
    print(test_path+os.path.splitext(filename)[0])

    for i in range(0,5):
        x=np.reshape(data[i],(28,28))
        img = Image.fromarray(x)
        img = img.convert('L')
        if i<4:
            img.save(train_path+os.path.splitext(filename)[0]+"/"+os.path.splitext(filename)[0]+str(i)+".png")
        else:
            img.save(test_path+os.path.splitext(filename)[0]+"/"+os.path.splitext(filename)[0]+str(i-4)+".png")
```

3.0

Data Analysis

Exploring the hidden layer

Exploring Test Data

```
In [19]: classes = ['star', 'apple', 'fish', 'moon', 'ice cream', 'tent', 'candle', 'fork', 'crab',
                    'cake', 'hammer', 'sun', 'diamond', 'table', 'light bulb', 'spider', 'face', 'eye',
                    'crown', 'shorts']

          #for filename in os.listdir("datasets/data/test_set"):
          #    classes.append(os.path.splitext(filename)[0])
```

Exploring Test Data

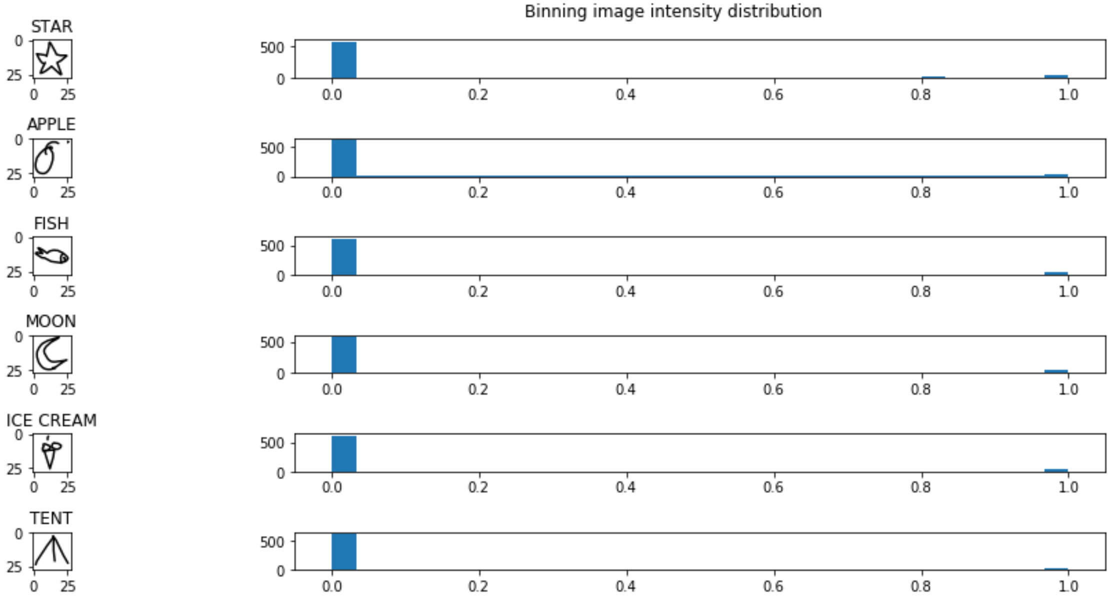
```
In [44]:  # Test set
          unique, counts = np.unique(y_test, return_counts=True)
          dict(zip(unique, counts))

Out[44]: {0.0: 2027,
          1.0: 2062,
          2.0: 2081,
          3.0: 2051,
          4.0: 2000,
          5.0: 2075,
          6.0: 1974,
          7.0: 2012,
          8.0: 1984,
          9.0: 1946,
          10.0: 2034,
          11.0: 1966,
          12.0: 1967,
          13.0: 1997,
          14.0: 1921,
          15.0: 1993,
          16.0: 1984,
          17.0: 1901,
          18.0: 2010,
          19.0: 2015}
```

Plotting Intensity Histogram

```python
# Image Intensity Histrogram for each class

fig, axs = plt.subplots(20, 2, figsize=(15,20))
plt.suptitle('Binning image intensity distribution', x=.7,y=1)
for i, (key,value) in enumerate(classes_dict.items()):
    axs[i,0].set_title(key.upper())
    axs[i,0].imshow(value[0,:784].reshape((28,28)), cmap="gray_r")
    axs[i,1].hist(value[0,:784].reshape((28,28)).flatten(), bins=30,)
plt.tight_layout()
```



4.0

Data Model

Building the CNN model

CNN Model Architecture for 345 Classes

```
In [110]: model = Sequential([Convolution2D(filters = 32,
                                   kernel_size = (3,3),
                                   activation = "relu",
                                   input_shape = img_size),
                               Convolution2D(filters = 32,
                                   kernel_size = (3,3),
                                   activation = "relu"),
                               MaxPooling2D(pool_size=(2,2)),
                               Dropout(.20),
                               Convolution2D(filters = 64,
                                   kernel_size = (3,3),
                                   activation = "relu"),
                               Convolution2D(filters = 64,
                                   kernel_size = (3,3),
                                   activation = "relu"),
                               MaxPooling2D(pool_size=(2,2)),
                               Dropout(.1),
                               Flatten(),
                               Dense(512, activation="relu"),
                               Dense(n_classes, activation="softmax")
                              ])
```

Model Summary

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_21 (Conv2D)           (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 24, 24, 32)        9248      
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 12, 12, 32)        0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 10, 10, 64)        18496     
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 8, 8, 64)          36928     
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 4, 4, 64)          0         
_________________________________________________________________
dropout_12 (Dropout)         (None, 4, 4, 64)          0         
_________________________________________________________________
flatten_6 (Flatten)          (None, 1024)              0         
_________________________________________________________________
dense_15 (Dense)             (None, 512)               524800    
_________________________________________________________________
dense_16 (Dense)             (None, 345)               176985    
=================================================================
Total params: 766,777
Trainable params: 766,777
Non-trainable params: 0
_________________________________________________________________
```

Model Training

```
train_datagen2 = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=False,
        fill_mode='nearest')



test_datagen2 = ImageDataGenerator(rescale=1./255)
```

In [99]:
```
%%time

training_set = train_datagen.flow_from_directory(
        train_path,
        target_size=(28, 28),
        color_mode="grayscale",
        batch_size=256,
        class_mode='categorical')
```

```
Found 2760000 images belonging to 345 classes.
Wall time: 1min 39s
```

In [100]:
```
%%time

test_set = test_datagen.flow_from_directory(
        test_path,
        target_size=(28, 28),
        color_mode="grayscale",
        batch_size=256,
        class_mode='categorical')
```

```
Found 690000 images belonging to 345 classes.
Wall time: 28.6 s
```

Model Optimizing

```
model.compile(optimizer = "nadam", loss="categorical_crossentropy", metrics=["accuracy"])
```

```
save_cb = ModelCheckpoint("conv_doodle_model.h5",
                          save_best_only=True)
```

Model Fitting

```
history = model.fit(
            training_set,
            steps_per_epoch=training_set.n//training_set.batch_size,
            epochs=15,
            validation_data=test_set,
            validation_steps=test_set.n//test_set.batch_size,
            callbacks=[save_cb, TensorBoard(log_dir="QuickDraw")])
```

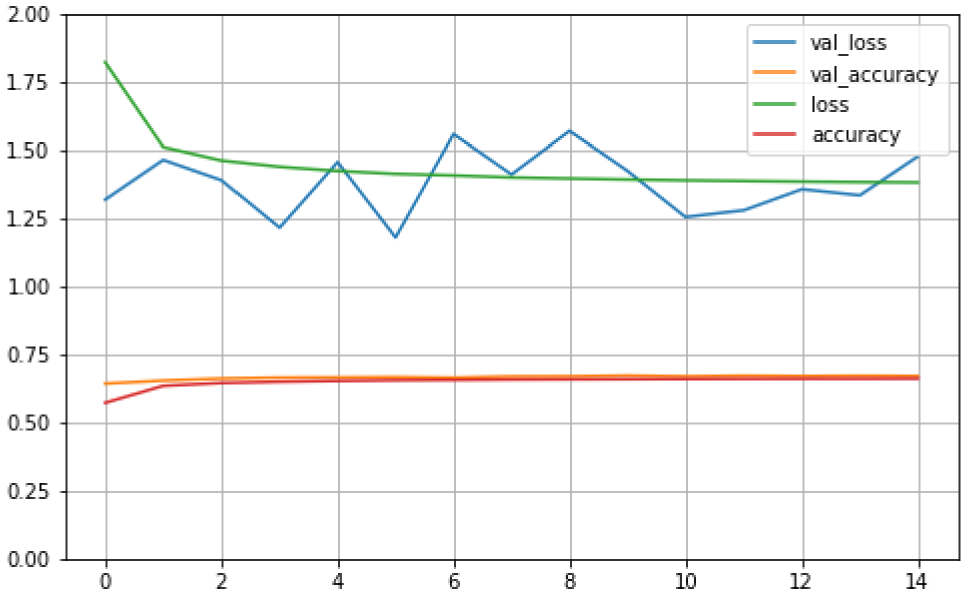
Do You See the Wall Time?

```
10781/10781 [==============================] - 5197s 482ms/step - loss: 1.3964 - accuracy: 0.6578 - val_lo
ss: 1.5704 - val_accuracy: 0.6688
Epoch 10/15
10781/10781 [==============================] - 5212s 483ms/step - loss: 1.3928 - accuracy: 0.6584 - val_lo
ss: 1.4241 - val_accuracy: 0.6715
Epoch 11/15
10781/10781 [==============================] - 5153s 478ms/step - loss: 1.3897 - accuracy: 0.6591 - val_lo
ss: 1.2549 - val_accuracy: 0.6686
Epoch 12/15
10781/10781 [==============================] - 5066s 470ms/step - loss: 1.3874 - accuracy: 0.6597 - val_lo
ss: 1.2798 - val_accuracy: 0.6711
Epoch 13/15
10781/10781 [==============================] - 5166s 479ms/step - loss: 1.3851 - accuracy: 0.6601 - val_lo
ss: 1.3570 - val_accuracy: 0.6697
Epoch 14/15
10781/10781 [==============================] - 6328s 587ms/step - loss: 1.3834 - accuracy: 0.6601 - val_lo
ss: 1.3347 - val_accuracy: 0.6707
Epoch 15/15
10781/10781 [==============================] - 5627s 522ms/step - loss: 1.3821 - accuracy: 0.6606 - val_lo
ss: 1.4774 - val_accuracy: 0.6698
Wall time: 1d 50min 55s
```

Model fitting was running little more than a day. The machine did not have enough hardware to contribute to the processing time.

“

Model Performance



Our CNN model did not perform well enough. We ended up with a validation accuracy of ~70%. Which isn't very attractive. Inclusion of all 345 classes might have been too ambitious for 10,000 sample images.

“

5.0

Lessons Learned

What could have been better

Selecting less than 345 classes could have shown a better result with added opportunity of testing several models.

“

Touch based human computer interaction for handicapped users could be improved more with exploratory datasets from various world regions.

“

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thanks!

Any questions ?

Please let us know in Quercus