Machine Learning Engineer Nanodegree

Capstone Project - **Quick, Draw! Doodle Recognition Challenge** (from Kaggle)

Gopala krishna kumar Thangavel
July 5th, 2019

1. Definition

Project Overview

AI/ML plays a big role in entertainment field as well. Nowadays anyone can reach their customer/consumer easily since everyone is connected to internet especially through their mobile devices. This project is part of that aim; to have an interface in mobile platform that will entertain users also in background predict results based on their actions.

I have taken this problem statement from Kaggle. This problem statement comes from the domain of Gaming done using Image classification. "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. Since the training data comes from the game itself, drawings can be incomplete or may not match the label. The challenge is 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. By advancing models on this dataset, pattern recognition solutions can be improved broadly and they can be used for scenarios like Handwriting recognition, OCR (Optical Character Reading), ASR (Automatic Speech Recognition) etc.

Problem Statement

Even though this use case comes from a gaming background, I’m planning to take up the part of simple classification of pre-made custom doodles of any objects/things (one among the 340 categories). So in essence, the problems statement is to categorize doodles drawn by user that fits in to the category set of 340 labels.

This problem can be solved by classifying the images (doodle data) with neural networks. I’m planning to use Convolutional Neural Networks (CNN). I will build a stack of different layers of

CNN along with supporting layers like Dropout, Dense etc. Once figuring out the proper network and its respective configuration I will feed the preprocessed data and train the network. Once the model is trained enough I can feed any doodle data and predict that category it belongs to (one among 340 categories example: airplane, alarm clock, car etc).

Metrics

I'm planning to majorly use following 2 evaluation metrics: 1. Accuracy score and 2. F1-score. Since the problem statement is a classification based scenario and it has quantifiable categories with datasets that are well balanced, both accuracy score and f-1 score will be best options to evaluate the model's performance.

Accuracy = Number of correct predictions/ Number of inputs

F1 Score = 2* (Precision * Recall)/ (Precision + Recall)

I have considered F1 Score because accuracy only includes the correct predictions not the wrong predictions (basically loss). But F1 Score includes both Precision and Recall. Because of this F1-score will also provide an advantage while reducing the overall loss value. More over F1-score will be helpful for evaluating the model against the benchmark model (contender have used f1-score to evaluate the model).

2. Analysis

Data Exploration

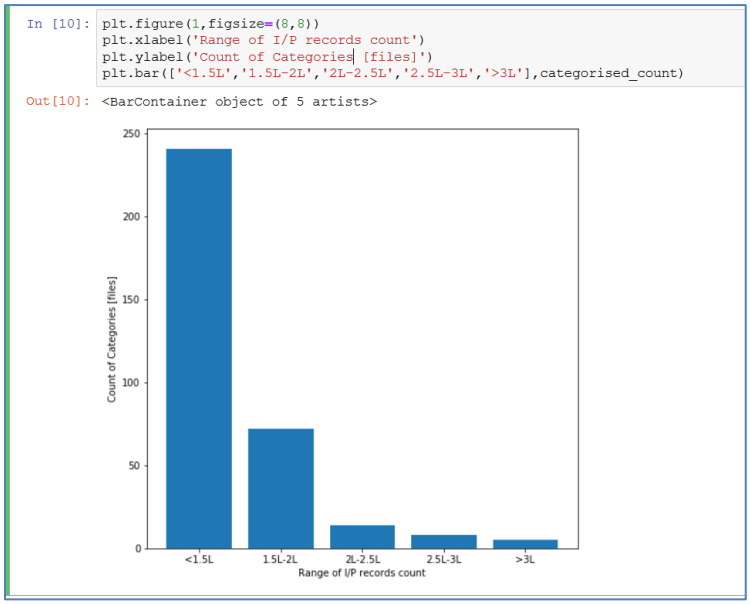
The Quick Draw Dataset is a collection of millions of drawings across 300+ categories, contributed by players of Quick, Draw! The drawings were captured as time stamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located.

Two versions of the data are given. The raw data is the exact input recorded from the user drawing, while the simplified version removes unnecessary points from the vector information. (For example, a straight line may have been recorded with 8 points, but since you only need 2 points to uniquely identify a line, 6 points can be dropped.) The simplified files are much smaller and provide effectively the same information. For this problem, either datasets can be used to solve or both can be used together. I'm planning to use simplified files for starting phase, since it will give an edge in terms of time and computation load.

Regarding the distribution of target classes, the count of datasets belonging to certain categories varies from 100,000+ to 300,000+. Whereas most of categories fall under 100,000 to 200,000

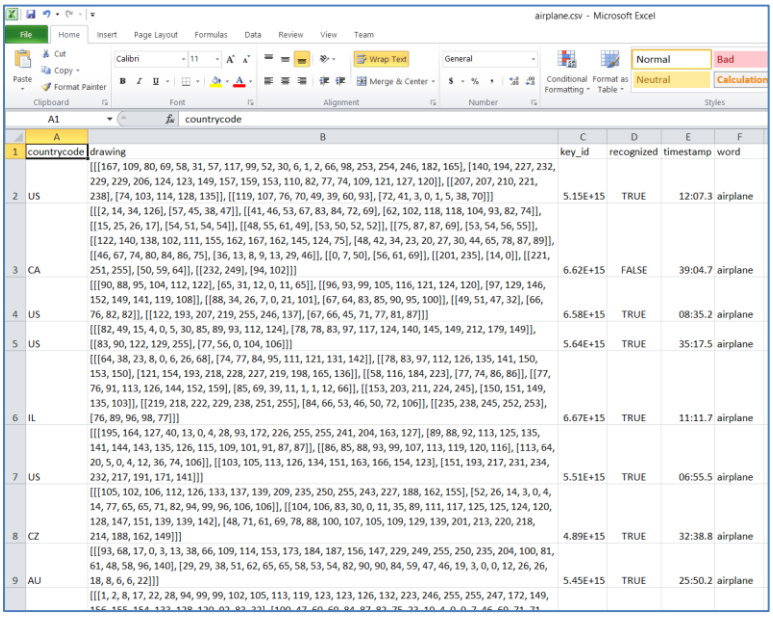
(almost around 80%) rest 20% are distributed till ~340,000 nearly. So there are enough data points to train each and every category so we can assume that the data points are well balanced.

Statistical representation of Input records count range vs Categories (files):



All doodles are provided in vector (matrix) format. It varies on number of strokes and color used by user for each stroke. So the data needs to preprocessed and restructured a bit so that all data will be of similar shape before being fed in to the network layers. Please refer below for a sample of dataset:

Input data in 'csv' file format:



Sample data visualized in code:

```
In [4]: sample_df.head()
```

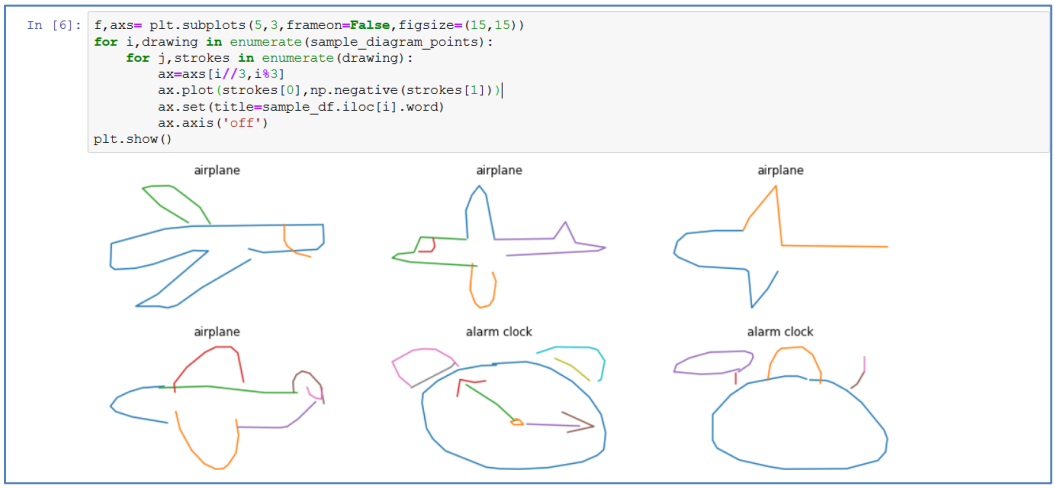
Out[4]:

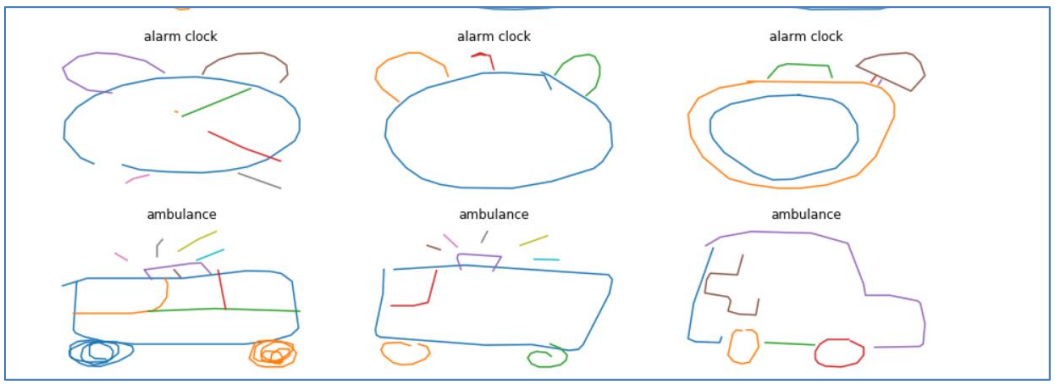
	countrycode	drawing	key_id	recognized	timestamp	word
0	US	[[[167, 109, 80, 69, 58, 31, 57, 117, 99, 52, ...	5152802093400064	True	2017-03-08 21:12:07.266040	airplane
1	US	[[[90, 88, 95, 104, 112, 122], [65, 31, 12, 0,...	6577010312740864	True	2017-03-23 02:08:35.229980	airplane
2	US	[[[82, 49, 15, 4, 0, 5, 30, 85, 89, 93, 112, 1...	5643224746033152	True	2017-03-10 00:35:17.531970	airplane
3	IL	[[[64, 38, 23, 8, 0, 6, 26, 68], [74, 77, 84, ...	6670046841536512	True	2017-01-23 18:11:11.658170	airplane
4	US	[[[111, 148, 161, 175, 199, 218, 231, 236, 234...	5159910851477504	True	2017-03-21 13:02:16.246170	alarm clock

Exploratory Visualization

Since these are customer's doodle drawing data, I tried to first visualize some of the sample data in image form. Also along with that I tried to map their corresponding category. In order to do these, I have utilized 'matplotlib' library. Please refer below for some of sample data in image form,

```
In [6]: f,axs= plt.subplots(5,3,frameon=False,figsize=(15,15))
        for i,drawing in enumerate(sample_diagram_points):
            for j,strokes in enumerate(drawing):
                ax=axs[i//3,i%3]
                ax.plot(strokes[0],np.negative(strokes[1]))
                ax.set(title=sample_df.iloc[i].word)
                ax.axis('off')
        plt.show()
```





Algorithms and Techniques

The input dataset provided is a collection of vectors. They represent the user's strokes along with color values. So the data will be of lengths (difference in number of strokes). So first I intend to convert the vector based data in to image data (multi-dimensional arrays). Once I convert them to uniform sized image data then I can feed them to stack of Convolutional layers.

Keras provides all the required support to create a Convolutional network along with all dependencies to train the same. So I will be using the keras libraries modules to create a stack of Convolutional layer (Conv2D) and other supporting layers (Maxpooling, Dropout, Dense etc) and thereby create a network.

I chose CNN because this problem statement can be approached as an image classification problem and CNNs are the best technique to process the image data and produce results. Basically CNNs segregate different characteristics (or features) in an image by processing through each section at a time and club them in to weights. Based on these values it predicts the category (or label) of that corresponding image.

While training, model will come up with different set of weights during each set of iteration. In order to capture the best set of weights I have added a 'ModelCheckpoint'. This checkpoint keeps track of the best set weights (lowest loss value) by storing the latest best value in a mentioned physical location in the form of 'hdf5' file. Once the model's training is over then we can simply load the best weights from the file in to the model. This handy technique is provided by keras library.

Benchmark

I'm planning to consider the model provided by one of the competitors in Kaggle platform for the same problem statement, as my benchmark model. Also this model can be verified in the following link: https://www.kaggle.com/nicstar92/quick-draw-doodle-recognition-challenge. From the code part given we can know that overall final score of this benchmark model is around ~0.65 (f1-score), and hence this model can be used as benchmark.

3. Methodology

Data Preprocessing

- From the previous section it can be known that provided datasets many columns like 'countrycode', 'key_id' and 'timestamp' which are not required for classifying the doodles. So I have removed the columns which are not required and kept only the required ones, namely 'drawing', 'recognized' and 'word'.

- Next step was to remove the invalid data. The dataset provided has all the user's doodle data: recognized doodles by AI/ML algorithm and unrecognized doodles as well. The unrecognized doodle data might have many uncertainties. The reason for failure in recognition was maybe poor doodle by the user or incomplete doodle etc. So it is better to skip all the unrecognized doodle data and consider only recognized ones.
- Next I converted the strokes data in to image data by adding all different strokes of a single diagram as the image's values. In order to do so I have utilized PIL library. Also I resized all the doodle data in to a uniform size of '32X32' so that the input data will be uniform and also easy to compute. At last I have scaled down the image data from 0-255 range to 0-1 range. So that CNN network can easily compute the values and scaled version of data is the ideal way to approach before processing.
- In case of the label data (y input), I converted it to category based matrix data (from single to double dimension). This has to be done because our model has outputs equal to the number of classes (categories) available. So input label data has to match the same dimension. I have utilized keras's 'to_categorical' functionality for this.
- Finally the data has to be split as training, testing and validation data. I have taken 10% of training data as validation dataset.

Implementation

- Building Model:

  The first step in implementation is to build model. As I have already mentioned I intend to use CNN model for this problem. So first I created a model with below stack,

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 64, 64, 32)        320
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 680)               11141800
_________________________________________________________________
dropout_3 (Dropout)          (None, 680)               0
_________________________________________________________________
dense_2 (Dense)              (None, 340)               231540
=================================================================
Total params: 11,392,156
Trainable params: 11,392,156
Non-trainable params: 0
_________________________________________________________________
```

Also the above model is compiled with loss function ‘categorical_crossentropy’ and ‘rmsprop’ as optimizer.

- Training Model:
  Before training the model, a checkpoint has to be added in order to capture the best results. I have used ‘ModelCheckpoint’ class of keras library to achieve that.
  Once all the above configurations are properly done, model’s training can be started. I trained the model for 50 epochs with batch size of 2048 and also provided both training and validation dataset. So that model, while training, it can validate itself against validation dataset in every single iteration.
  Below is a sample training output:

```
Train on 459000 samples, validate on 51000 samples
Epoch 1/50

Epoch 00001: val_loss improved from inf to 3.54635, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 100s - loss: 4.6232 - acc: 0.0978 - val_loss: 3.5464 - val_acc: 0.2374
Epoch 2/50

Epoch 00002: val_loss improved from 3.54635 to 2.75706, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 85s - loss: 3.3360 - acc: 0.2582 - val_loss: 2.7571 - val_acc: 0.3610
Epoch 3/50

Epoch 00003: val_loss improved from 2.75706 to 2.38484, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 75s - loss: 2.8207 - acc: 0.3429 - val_loss: 2.3848 - val_acc: 0.4299
Epoch 4/50

Epoch 00004: val_loss improved from 2.38484 to 2.21252, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 75s - loss: 2.5403 - acc: 0.3942 - val_loss: 2.2125 - val_acc: 0.4684
Epoch 5/50

Epoch 00005: val_loss improved from 2.21252 to 2.02401, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 71s - loss: 2.3574 - acc: 0.4282 - val_loss: 2.0240 - val_acc: 0.5019
Epoch 6/50

Epoch 00006: val_loss improved from 2.02401 to 2.02168, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 71s - loss: 2.2323 - acc: 0.4534 - val_loss: 2.0217 - val_acc: 0.4995
Epoch 7/50

Epoch 00007: val_loss improved from 2.02168 to 1.87638, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 72s - loss: 2.1358 - acc: 0.4715 - val_loss: 1.8764 - val_acc: 0.5304
Epoch 8/50

Epoch 00008: val_loss improved from 1.87638 to 1.82214, saving model to saved_models/model.weights.best.hdf5
459000/459000 - 71s - loss: 2.0605 - acc: 0.4861 - val_loss: 1.8221 - val_acc: 0.5405
```

Refinement

- During the initial phase of training I dint get the expected output. Model dint perform adequate enough. So I started analyzing the layers one by one and looking out for portions that can be improved or modified.
- The initial experimentation gave only around ‘50+%’ of accuracy. Also the validation loss was quite high somewhere around ‘2.1’. So in order achieve I tried ‘Image Augmentation’ technique.
- In Image Augmentation technique instead of providing the input image dataset to model for training, a data generator has to be provided to the model which generates set of augmented (slightly modified) image data. Modification includes horizontal & vertical swapping of image, horizontal & vertical shift in image content to a certain pixels etc.
- But after I tried Image Augmentation also the results didn’t improve, rather it started degrading. Then I understood that Image Augmentation doesn’t go very well with doodle

drawing data since this is a collection of strokes and if the strokes are shifted, thereby lost to some extent then the doodle itself might become incomplete or wrong.

- So in the end I made the CNN a little more deep with more supporting 'Dropout' layers. Once I did that slight restructuring to model and started training I got considerably good results. The final model gave around '61.5%' of accuracy.
  Please refer below the final model's summary,

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 680)               1393320
_________________________________________________________________
dropout_3 (Dropout)          (None, 680)               0
_________________________________________________________________
dense_1 (Dense)              (None, 340)               231540
=================================================================
Total params: 1,717,532
Trainable params: 1,717,532
Non-trainable params: 0
_________________________________________________________________
```

4. Results

Model Evaluation and Validation

I ran an inbuilt model evaluation function using the test data (unseen data). The model predicted considerably fast and gave a decent result. Please refer below image for the same,

```
model.load_weights('saved_models/model.weights.best.hdf5')
results = model.evaluate(x_test,y_test,batch_size=4096)
```

```
170000/170000 [==============================] - 11s 64us/sample - loss: 1.4776 - acc: 0.6151
```

```
print("Accuracy: %f%%"%(results[1]*100))
```

```
Accuracy: 61.514115%
```

Apart from this, I also created a 'classification report' matrix in order to get the 'f1 score' of the model. F1 score came around '~0.61'. Please refer the same below,

```
170000/170000 [==============================] - 6s 36us/sample
```

	precision	recall	f1-score	support
0	0.61	0.77	0.68	464
1	0.71	0.65	0.68	515
2	0.53	0.52	0.52	501
3	0.74	0.75	0.74	508
4	0.39	0.58	0.46	462
5	0.54	0.66	0.60	501
6	0.77	0.71	0.74	547
7	0.73	0.87	0.79	513
8	0.52	0.47	0.49	496
9	0.48	0.57	0.52	456
10	0.74	0.67	0.70	496
11	0.65	0.48	0.56	510
12	0.61	0.70	0.65	509
13	0.72	0.84	0.78	536
14	0.61	0.67	0.64	482
...	...	...	...	...
326	0.74	0.61	0.67	502
327	0.55	0.42	0.48	499
328	0.72	0.80	0.75	508
329	0.55	0.48	0.51	473
330	0.56	0.56	0.56	529
331	0.62	0.53	0.57	538
332	0.50	0.81	0.62	486
333	0.69	0.69	0.69	530
334	0.69	0.83	0.75	498
335	0.85	0.88	0.86	492
336	0.71	0.56	0.63	526
337	0.52	0.34	0.41	500
338	0.57	0.71	0.63	495
339	0.65	0.76	0.70	493
accuracy			0.62	170000
macro avg	0.61	0.62	0.61	170000
weighted avg	0.61	0.62	0.61	170000

I have to train the model a lot more in order to gain better results. Also there is a scope improve the model more. Since training a model is very time consuming and costly in terms of computing I couldn't do much more in that prospect. Also I have to read and learn a lot more on CNNs in order to make it better in future.

Justification

The model I created has performed decently by predicting with accuracy of ~61%. I guess for this problem statement this model is a good initial solution for recognizing the users' doodle. Of course there is a lot of scope for improvement. With better training and tuning it will perform well enough. Please find below the comparison with benchmark model,

Model	Accuracy	F1 Score
Benchmark	~65%	~0.64
CNN based model	~61%	~0.61

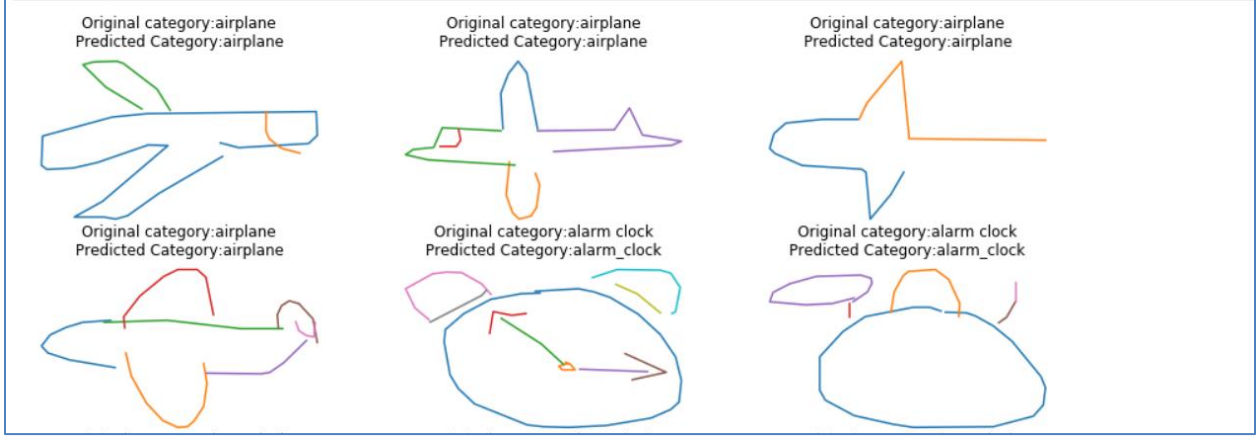
The final model's performance is not better than benchmark model but it's almost near. Accuracy difference is about 4% and f1 score difference is about 0.03.

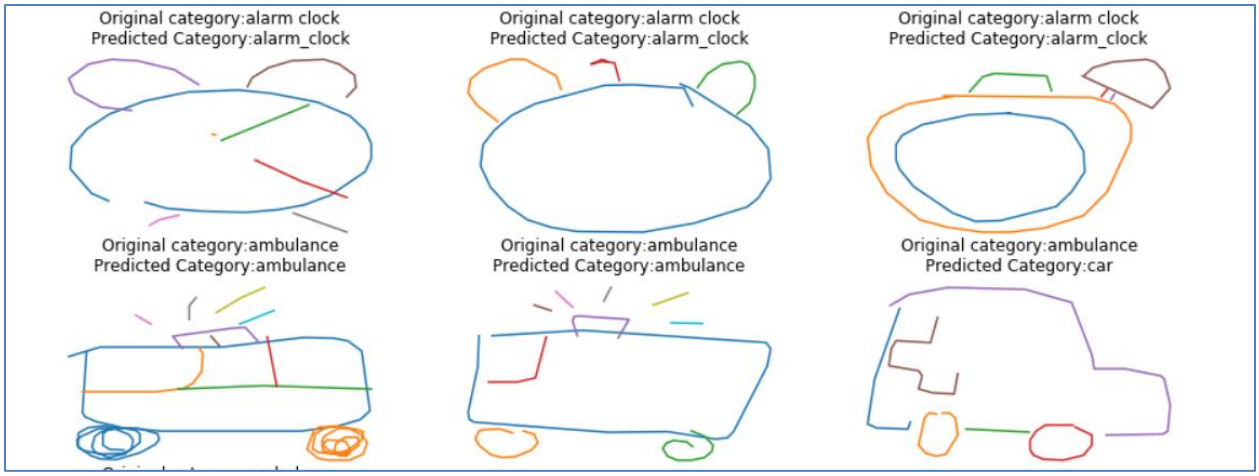
5. Conclusion

Free-Form Visualization

The final step was to predict some doodle data using the final model. Please refer the below images for the same,

```
prediction = model.predict(file_data_array)
prediction = np.argmax(prediction,axis=1)
f,axs= plt.subplots(5,3,frameon=False,figsize=(15,15))
for i,drawing in enumerate(sample_diagram_points):
    for j,strokes in enumerate(drawing):
        ax=axs[i//3,i%3]
        ax.plot(strokes[0],np.negative(strokes[1]))
        ax.set(title=('Original category:'+sample_df.iloc[i].word+'\nPredicted Category:'+categories[prediction[i]]))
        ax.axis('off')
plt.show()
```





From the above images we can tell that the model has performed significantly well.

Reflection

This problem could have been approached in numerous different ways also. Since the input dataset is not exactly image but vector data of user's strokes, it could have been approached in different way also. But I chose image classification approach and in order to achieve the same I utilized my knowledge in CNN. Since the data was already much clean and simplified there wasn't much preprocessing and cleaning steps (Thanks to Google!). At first the difficulty was to convert the strokes vector data in to uniform image data. Once that is done I tried with a minimalistic model, a simple CNN network. After getting some insights, then I tried improving the models architecture then I was able to get good results. On side note I also tried Image Augmentation technique but that dint work out well enough. Maybe some other image processing techniques can be utilized that might work well. The most difficult part was to run the training model job since it was too much time consuming and cpu intensive. One rather interesting aspect was how the model was quicker during evaluating test dataset (around 17k records) in matter of seconds. So I guess CNN are best and fast enough to do image classification or in this particular case real time doodle recognition.

Improvement

- There is a lot of scope for improvement in this project. First it can be well implemented in such way the overall file reading and data processing can be done faster than the current speed.
- The architecture of the model can be fine-tuned more and also the model can be trained more so that it produces better results.

- Instead of providing the doodle data if some other image data is provided to my model and with slight change in the output layer (last layer) then it might be able to predict that as well. So in general my model is a good starting point for image classification.
- For sure, I would consider trying imaging techniques similar to Image Augmentation so that overall performance of the model improves.
- Yes there are lots many solutions better than mine from Kaggle itself.