1.Methodology

(1) Classifier:

(a) Model Structure:

(i) Structure of classifier is basically the same as the sample code , except that we add one more hidden layer.(See red pen marked above)

```
#train
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu', input_shape=(imheight, imwidth, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
###################################
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
###################################
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(680, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

(b) Optimizer:

(i) We also change the default settings of adam optimizer , change the original learning rate from 0.001 to 0.002 , which speeds up the learning process. (beta values are the same as default)

(ii) Moreover , we increase batch size to 64 which gives us better performance and set the argument shuffle to be true which randomly reorders the samples before each epoch. Also , we abort early stopping because early stopping activates when validation error goes up(similar to preventing overfitting) . But we don’t know whether it is because of leaving the local optimal or not. Finally , we set epoch 15(sample code is 22 and we found it is overfitting).

```
reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
                                   verbose=1, mode='auto', min_delta=0.005, cooldown=5, min_lr=0.0001)
earlystop = EarlyStopping(monitor='val_top_3_accuracy', mode='max', patience=5)
# callbacks = [reduceLROnPlat, earlystop]
callbacks = [reduceLROnPlat]
#0.001->0.002
adam = keras.optimizers.Adam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['accuracy', top_3_accuracy])

model.fit(x=X_train, y=y_train,
          ##################
          #32->64
          batch_size = 64,
          ##################
          epochs = 15,
          ##################
          validation_data = (X_val, y_val),
          ################
          shuffle=True,
          ################
          callbacks = callbacks,
          verbose = 1)
```

(2) Generator:

(a) DCGAN :

(i) We use the model of DCGAN on github(by reference).

(ii) We alter the model to let image with size of 64*64 to fit in.

(iii) We decrease the parameter of Dropout to maintain the information of each image.

(iv) We decrease the learning rate of Generator in order to enhance the accuracy and avoid local minimum.

(v) We Increase the learning rate of Discriminator to accelerate the training.

2. Evaluation & Test Result

(1) Classifier

(a) Use the test case provided py TA to evaluate the result.

(i) Accuracy:97%

(ii) Test Case:40/41

```
[7]
test = pd.read_csv('./drive/My Drive/input/demo.csv')[['drawing']]
imagebag = bag.from_sequence(test.drawing.values).map(draw_it)
testarray = np.array(imagebag.compute())
testarray = np.reshape(testarray, (testarray.shape[0], imheight, imwidth, 1))
# model = load_model("./drive/My Drive/output/model/classifier.h5")
testpreds = model.predict(testarray, verbose=0)
ttvs = np.argsort(-testpreds)
for idx in ttvs[:,[0,1,2]]:
    print(numstonames[idx[0]])
    # print(numstonames[idx[1]])
    # print(numstonames[idx[2]])
    print('----------')
```

```
bee
----------
cactus
----------
banana
----------
fork
----------
wine_bottle
----------
```

Testing Case

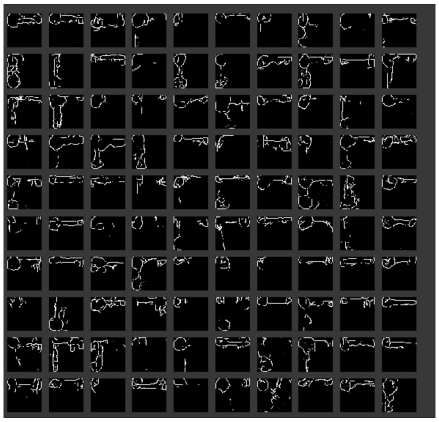
```
[9]
#test
test = pd.read_csv('./drive/My Drive/input/test.csv')[['drawing']]
imagebag = bag.from_sequence(test.drawing.values).map(draw_it)
testarray = np.array(imagebag.compute())
testarray = np.reshape(testarray, (testarray.shape[0], imheight, imwidth
# model = load_model("./drive/My Drive/output/model/classifier.h5")
testpreds = model.predict(testarray, verbose=0)
ttvs = np.argsort(-testpreds)

# for idx in ttvs[:,[0,1,2]]:
    # print(numstonames[idx[0]])
    # print(numstonames[idx[1]])
    # print(numstonames[idx[2]])
    # print('----------')

predict_label = [numstonames[ttvs[i][0]] for i in range(ttvs.shape[0])]
test['word'] = predict_label
test.to_csv('./drive/My Drive/output/output.csv')
```

```
10]
test_ans = pd.read_csv('./drive/My Drive/input/test_ans.csv',index_col=0)
# print(test_ans['word'])
```

(2) Generator

(a) Save the model : we alter the proper amount of image and epoch and batch size, than save model.

(b) Test: we randomly generate 100 images and feed our classifier, than compare the result.

3. Demo Result

(1) Classifier

(a) Result:

(i) Test Case:36/40

(ii) Accuracy : 90%

(2) Generator

(a) Result:

(i) Test Case: 100/100

(ii) Accuracy : 100%

4. Discussion & Conclusion

(1) Classifier

(a) Indistinguishable test case: we are confused about the result of our classifier, and eventually we found several ambiguous images

(b) Overfitting : Cancel Early Stopping

(2) Generator

(a) During training process of GAN , we save the generator every 1 ~ 3 epochs , and feed the generator into the classifier to perceive how many epochs the GAN has the best performance.

(b) Our classifier will list three most likely labels , we will use this to determine whether further training does better.

(c) Conclusion:

(i) Easy image label(e.g. hand) → Every doodle drawing game players can draw perfectly(high quality data) -->increase data size to 200000

(ii) Difficult label(e.g. raccoon) →Many low quality drawings -->decrease data size to 100000

Group1:曾靖渝（34%）陳維仁（34%）袁維澤（32%）