In project_2 I use CNN to solve this problem. (cnn.py, import tensorflow)

At beginning I transform all the picture to size 100*100*3, where 3 means RGB three channels. And I use the CNN to deal with the data. I use 4 hidden layers and pooling layers, two dense layers, most of them use RELU as activation function for the number of hidden layers and finally use Softmax function as our loss function.

When train data, I set batch size as 64 and in every loop I split train data randomly and use those data to train the model.

I try this model with different epoch and the accuracy on val.txt can be higher than 60%. Which is good enough compare with classifying the picture randomly.

Finally, I get the result, and we also save the model learned with different epoch and some of the model we trained have been uploaded. (we total get four files when upload to github, I only upload three small files for the last one is too large)

Maybe, add hidden layer or change the parameters can improve the accuracy, but I think the no matter how you change the model, finally the improvement of accuracy is limited. If we want to improve the accuracy, we should change the model.

Besides, I also find someone else uses transform learning or retraining

of some models like Inceptionv3, which give a higher accuracy (even higher than 90%). For example, in official website of tensorflow and keras give us a tutorial to deal with similar problem.

So, I try this way in keras, after Inceptionv3, I use flatten to change the shape, add dense layer, then a dropout layer in order to not overfit and finally use softmax function. (keras_simple.py) And the accuracy on val.txt could be much higher, compared with cnn.py and higher than 90% on val. This is a better way to deal with the problem.