# Convolutional Neural Network

## Introduction to technique

The convolutional neural network (CNN) is a kind of neural network which is most widely used in large-scale image processing tasks, e.g. image classification, segmentation. In recent years, the researchers also applied CNN in depth estimation, natural language processing tasks. In our assignment, we implemented our own CNN using TensorFlow [1], and achieved very good result in predicting the information of chess pieces given a chessboard image. We further integrated our classification module into a chess game [1] and realized prediction in real time.

## Specific pre-processing

Without using too much prior knowledge of the generation of the chess-position [1] dataset, we made several rational assumptions over the data we are going to accept as predictors, which are listed as below:

The train data format should be 400 x 400 pixels, 3-channel RGB image, with each consecutive 50 x 50 pixels representing a grid of a chessboard. The theme of the 64 grids in the same image should be the same (or could be different, depending on our further results.)

The classifier should predict a single grid without given any information of the other grids of the same board, meaning that the input to our classifier should be 50 x 50-pixel, 3-channel image.

## Implementation

Our implementation is a typical showcase of how to build a CNN using TensorFlow, the network itself is designed to be in a most simple form of CNN, A convolution layer of kernel size 3, followed by 2 identical convolution layers of convolutional layer of kernel size 3, then flatten is applied, and followed by a 128-dimension dense layer (fully connected layer) which stores the most important weight information of our whole CNN. After that, the final layer comes in the form of a dense layer of dimension 13, which corresponds to the number of labels in our case.

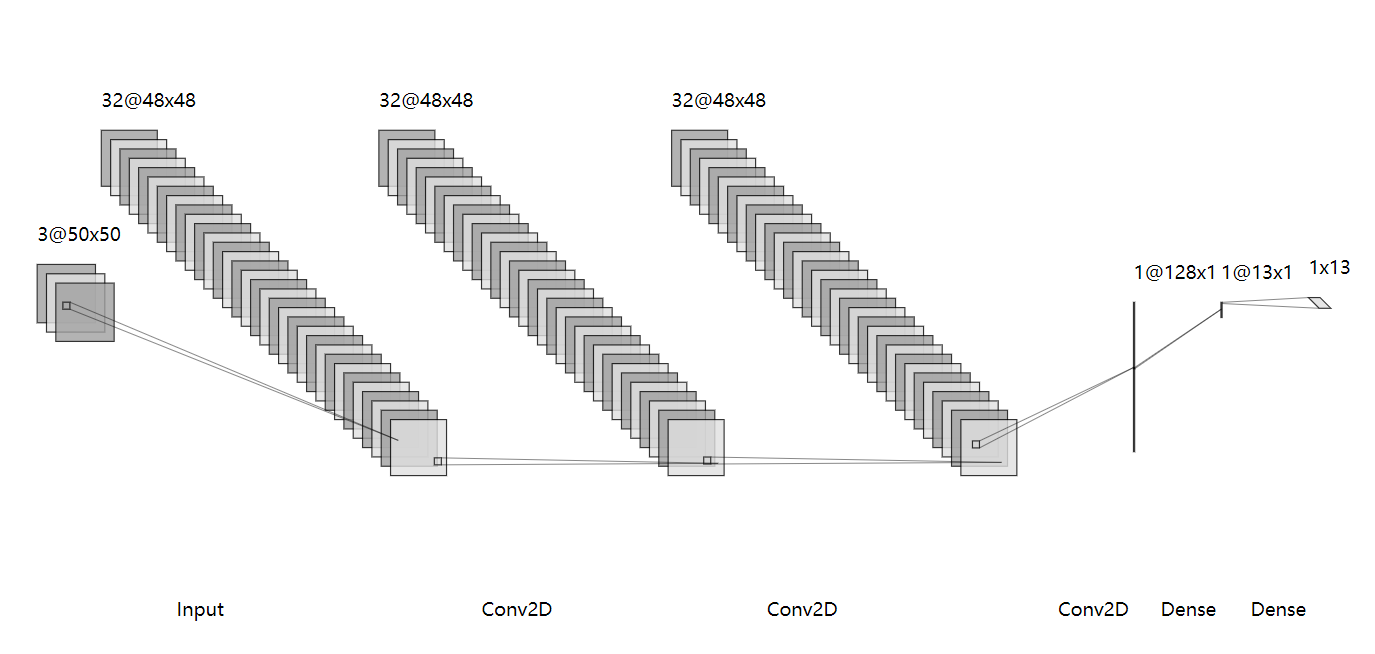


Fig. 1. Structure of our CNN with dropout layers omitted. Figure created with NN-SVG [1]

All the convolution layers and the 128-dimension dense layer use rectified linear unit (ReLU) as activation function.

We did not use max pooling layer since the input image size is relatively small, adding max pooling layer would simply prolong the feed-forward process and after a few tests, did not bring obvious advantage to our training or validating.

## Hyper parameter tuning

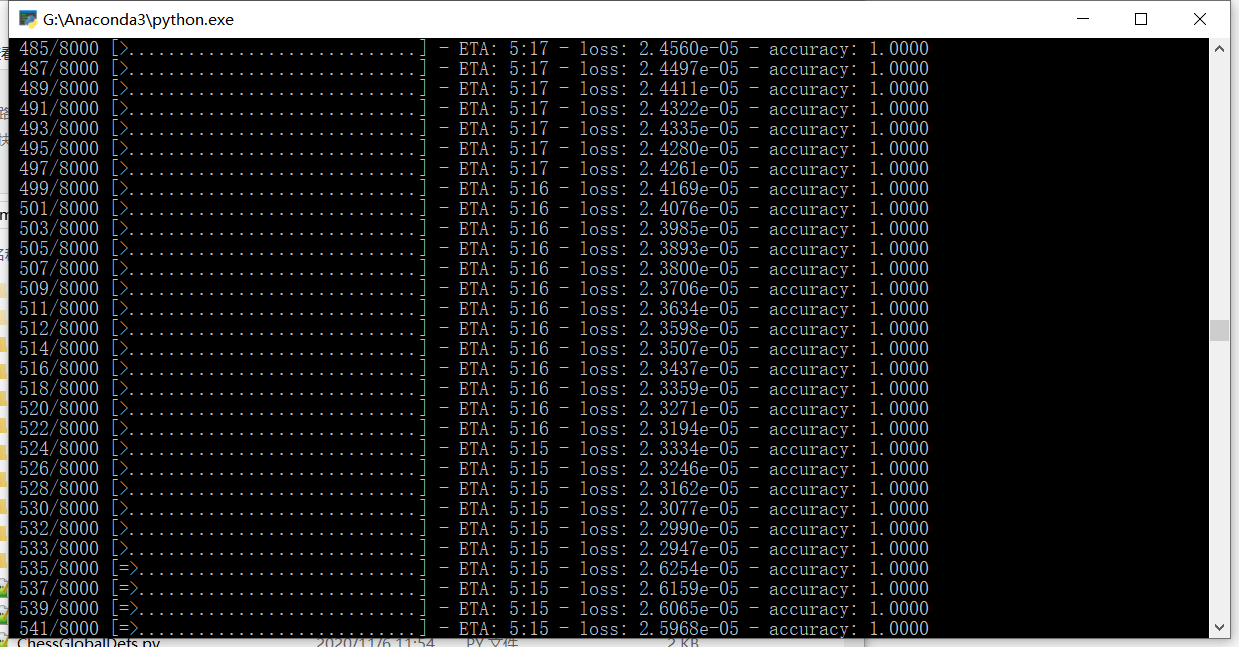
The fine-tuning includes three parts:

#### Max-Accuracy: Find the hyper parameters that result in the best accuracy.

With the default setup (64 grids of a board image is passed as a single batch, 1 epoch only), during the 1-hour training session we could observe that the loss kept converging. Even the training was run for only one single epoch, the training accuracy reached a surprising level of about 99.60%. After another setup where 5 epochs were run, the training accuracy reached 99.96%.

To avoid overfitting, we also added two Dropout layers in our network, one is placed between the first and second Conv2D layer, and the other is placed after the 128-dim dense layer. This could reduce the chance of overfitting to some extent.

It is worth noticing that in a particular case the 100% accuracy was reached, but later the accuracy dropped to around 99.92%.



*Fig. 2. A rare case where the accuracy achieves 100%*

#### Min-Training time: Fine tune the parameters of the layers of the network so that the training converges in a relatively small time (10 minutes or so).

To reduce the training time of our CNN, the first thing we did is to increase the batch size. While using a generator function would result in undetermined number of total batches, instead of changing the actual batch size, we tested different values for steps\_per\_epoch from train\_size / 1 (default setting) to train\_size / 100 (100x batch size compared to default), where the train\_size is the number of total images of our training, in our case, with the 10-fold cross validation setup, it is 72000 (0.9 / 0.1 ratio of train/validation).

With the batch size tuned, we could always finish training for one epoch in around 7 minutes and reach an accuracy of about 99.8% on our hardware setup. After each epoch, the weights are saved to hard disk to allow further testing.

#### Max-FPS: Without losing accuracy, fine tune the parameters so that the run-time speed of prediction is optimized (how many frames could be predicted in one second).

To further optimize the run time performance of our CNN, we added down-sampling before feeding the data to the CNN being trained, and tested different image sizes. This would result in changes of layer size changes of our network, so unit-tests were run for fine-tuning purpose.

TODO: benchmark tests of different image sizes.

## Output

TODO: use unified format of output: confusion matrix, accuracy, etc.

## Reference

1. TensorFlow API documentation, <https://www.tensorflow.org/api_docs/python/tf>
2. Python Easy Chess GUI, <https://github.com/fsmosca/Python-Easy-Chess-GUI>
3. NN-SVG, Publication-ready NN-architecture schematics, <http://alexlenail.me/NN-SVG/index.html>