Contents

Regression CNN Technical Report

Jing Zhang

January 2020

1 Introduction

Head circumference (HC) is one of key biometrics in fetus growing stage, it reflects if a fetus develops normally during pregnancy. And the measurements of HC are finished by experts or doctors in hospitals. However, manual measurement of HC is laborious and subjective. Nowadays, deep learning technology saves lots of time and has high accuracy on different tasks.

Generally, if we want to see if a fetus head growing healthy or not, first, the machine will take an ultrasound image for the fetus in pregnant woman, then, the doctors will analyse the fetus head according to several index, for example, they will compute head circumference to assess fetal size. So, to calculate head circumference easily, the contour of fetus head can be simply regarded as an ellipse.

Therefore, in deep learning based methods, the ultrasound images of fetus are the train data, the corresponding ellipse drew by doctors are the ground truth, and they are trained by specific model for segmentation task. Subsequently, the predicted results are the contours of fetal head, the shape of those contours is irregular ellipse, In order to calculate the circumference of these contours, they need further fitting to turn into a regular ellipse. This is a method that people can come up with logically. In [1], they give their solution based on this idea.

However, the idea above has to do two-step computing, prediction and fitting. Thus, we believe there should be a way that can be different from that.

There is a challenge called "Automated measurement of fetal head circumference" [2]. See Fig 1, the red ellipse is fetus head, the yellow line is the coordinate axis, the center point of coordinate axis is also the ellipse's center point, the two green lines are long axis and short axis of ellipse respectively. The goal of this challenge is that given the 2D ultrasound images and their ground truth, calculating the head circumference. According to their request, we should upload five parameters, they are: the center points, two axis of the ellipse, and the angle between one axis and coordinate axis. Once these parameters are obtained, the position and circumference of head can be computed easily.

Therefore, inspired from the challenge, in our method, instead of predicting the head contour of fetal, we predict the 5 parameters mentioned above directly using regression convolutional neural networks, so this method doesn't need to fitting ellipse, but also

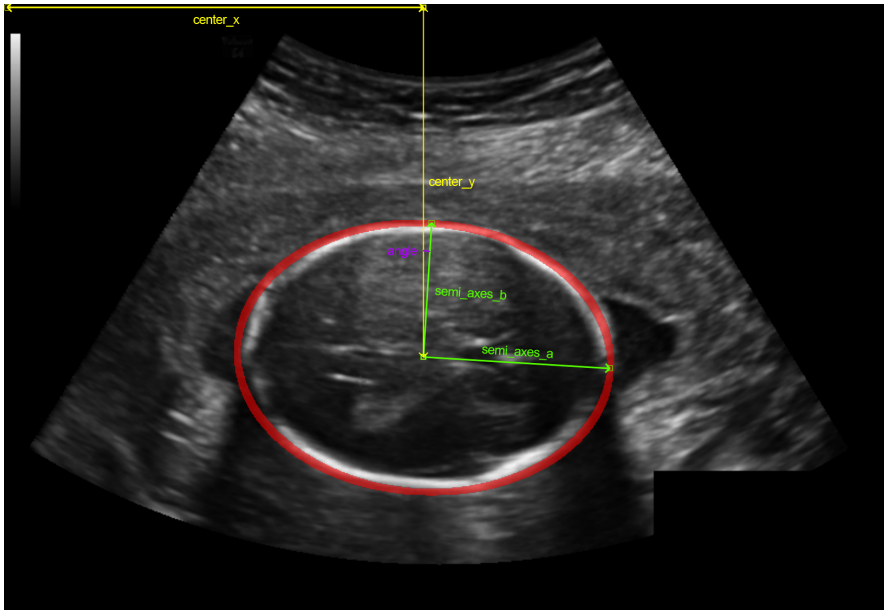

Figure 1: Ultrasound image of a fetal

the predicted results satisfies the uploading request of the challenge. The only thing we need to do is to do some preprocessing on the ground truth data.

2 Method

2.1 Regression CNN

Thanks to the challenge, we come up with the idea of regression CNN. It is a convolutional neural networks with a regression layer at last layer. In the main part of model, it is used for learning the features of input images. In the regression layer, it is used for estimating the relationships between dependent variable (predicted results) and independent variables (ground truth).

Before training the data, we need to create new ground truth data based on given ellipse ground truth. According to this ellipse, we can calculate the center point (x,y), long axis and short axis of ellipse, also the angle. And they are saved in the CSV file, and each image corresponds to five parameters.

As for the loss function, because the regression layer can be regarded as linear regression, so the loss function is also linear, such as mean absolute error, mean squared error and huber loss. And the huber loss takes on behavior of mean squared error when loss

is small, and mean absolute error when loss is large[1].

$$mae = \frac{1}{n}\sum_{i=1}^{n}|p_i - g_i| \tag{1}$$

$$mse = \frac{1}{n}\sum_{i=1}^{n}(p_i - g_i)^2 \tag{2}$$

$$hl = \begin{cases} \frac{1}{2}(p_i - g_i)^2, & if|p_i - g_i| < \delta \\ \delta * (|p_i - g_i| - \frac{\delta}{2}) \end{cases} \tag{3}$$

2.2 Keras model tuner

Usually, we have to face a problem that is how to design a robust model that have high accuracy or low loss vaule. For example, how many convolutional layers? How many filters to set at the first convolutional layer? Which optimizer to use? Which loss function to use? However, the answer is often got by trial and error! If we exhaustively list each combination, it can be extremely low efficient.

Because our method is implemented by Python and Keras package. Developers release a package called Keras Tuner[2], which saves a lot of time to select a relatively optimized model. And this tutorial is shared by sentdex[3].

The Keras Tuner including two steps. First, build the model, see the code below, we pass the hyper parameters which we want to optimize to the function.

```
!pip install keras-tuner

from tensorflow import keras
from tensorflow.keras.layers
import Conv2D, MaxPooling2D, Dense, Flatten, Activation
from tensorflow.keras import layers
def build_model(hp):
    model = keras.models.Sequential()
    model.add(Conv2D(hp.Int('input_units',
                            min_value=32,
                            max_value=256,step=32),
                            (3, 3),
                            input_shape=train_x.shape[1:]))
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

[1]http://www.cs.cornell.edu/courses/cs4780/2015fa/web/lecturenotes/lecturenote10.html
[2]https://keras-team.github.io/keras-tuner/
[3]https://pythonprogramming.net/keras-tuner-optimizing-neural-network-tutorial/

```
    for i in range(hp.Int('num_layers', 2, 20)):
        model.add(layers.Dense(units=hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=32),
                                            activation='relu'))
    model.add(Flatten())
    model.add(Dense(5))
    model.add(Activation("linear"))
    model.compile(
        optimizer=keras.optimizers.Adam
        (hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss = hp.Choice('loss',
        ["mean_squared_error", "mean_absolute_error", "huber_loss"]),
        metrics=['accuracy','mae'])
    return model
```

Second, tune the model, see the code below, it will rank the models by val_loss or val_acc and then select out the best model which has the best score. After this two steps, it will give us a summary about the parameters of each layer and hyper parameters of the best model.

```
dir = 'drive/My Drive/Colab Notebooks/HC-18/'

tuner = RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=20, # how many variations on model?
    executions_per_trial=3, # how many trials per variation?
    directory=dir,
    project_name='HC18trial')

tuner.search_space_summary()
tuner.search(train_x,train_y.values,
             epochs=1,
             batch_size=8,
             validation_data=(valid_x, valid_y.values))
tuner.get_best_hyperparameters()[0].values
tuner.get_best_models()[0].summary()
tuner.results_summary()
tuner.get_best_models()[0].evaluate(valid_x, valid_y)
```

3 Experiment

3.1 Model

Different from regular CNN models, the main feature of regression CNN model is that the last layer of activation function is replaced with a regression layer. See Fig 2. First, the learned features will be flattened to 5 parameters at the second last layer. Then the regression layer begins, these parameter go through a "linear" activation function and they are optimized by a linear loss function.

3.2 Data preprocessing

Before throwing the data into the model, some preprocessing is necessary. Because the size and pixel range of original images are not suitable for a regular neural networks and current computational ability, and may also lead to a low accuracy result. Therefore, in this work, the images are resized to 64*64, gray scale level, the pixel value is normalized to mean value is 0, standard deviation is 1. $img\_norm = (1/img\_std) * (img - img\_mean)$.

At the same time, data augmentation is added. We use rotation, shifting, flipping to create new images. The original number of images are 999, after rotation, it became 1998, after shifting, it became 3996, after flipping, it became 7992. Correspondingly, we calculate the five index (center x, center y, axis a, axis b, angle) of each image as ground truth, they are saved in CSV files.

3.3 Different loss functions

In our experiments, we use three loss functions, they are mean absolute error (mae) or mean squared error (mse) or huber loss (hl).

In our experiment, we set $\delta = 1$, and we compare these loss functions with different number of images. The result is in Tab 1.

Table 1: Comparison of different loss functions under different number of train images

#	Loss	loss	mae	acc
999	mse	7.402	13.399	0.988
	mae	10.673	10.673	0.988
	huber	11.945	12.379	0.988
1998	mse	197.583	8.508	0.974
	mae	8.735	8.735	0.97
	huber	7.978	8.390	0.97
3996	mse	236.148	9.672	0.981
	mae	8.854	8.854	0.984
	huber	8.751	9.160	0.982
7992	mse	375.339	11.133	0.982
	mae	12.105	12.105	0.982
	huber	12.496	12.905	0.982

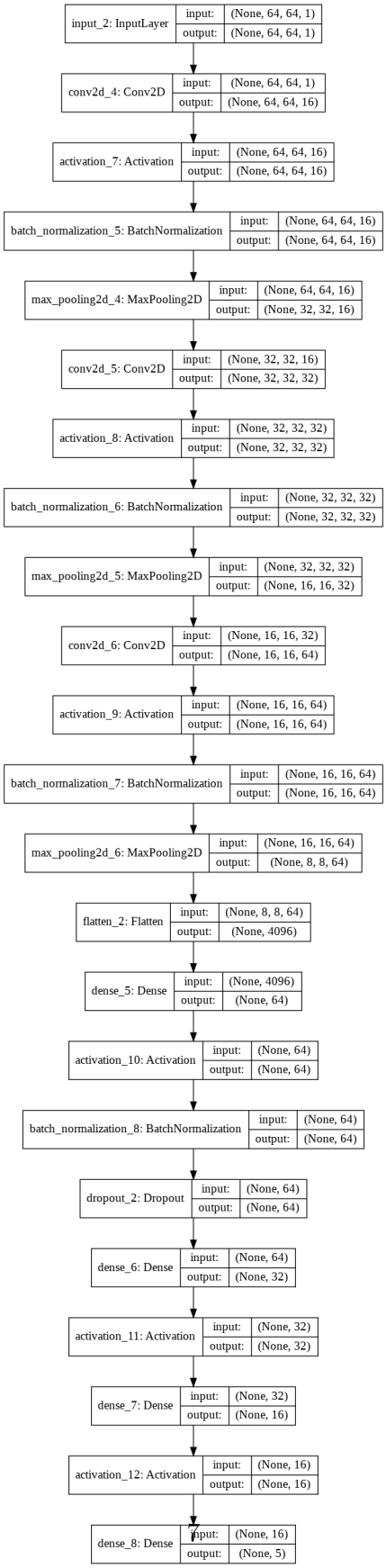

Figure 2: Regression CNN

References

[1] Samuel Budd, Matthew Sinclair, Bishesh Khanal, Jacqueline Matthew, David Lloyd, Alberto Gomez, Nicolas Toussaint, Emma C Robinson, and Bernhard Kainz. Confident head circumference measurement from ultrasound with real-time feedback for sonographers. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 683–691. Springer, 2019.

[2] Thomas L. A. van den Heuvel, Dagmar de Bruijn, Chris L. de Korte, and Bram van Ginneken. Automated measurement of fetal head circumference using 2d ultrasound images. *PLOS ONE*, 13(8):1–20, 08 2018.