Push-up Image Analysis Using Fastai

Hyeun-Jun, Kim
20201059

Kyeol, Heo
20201166

Abstract We analyze push-up images using models provided by Fastai Vision to create apps that will help user s with push-ups. Attempts using image segmentation models did not achieve sufficient learning due to structural limitations. We have successfully used an image classification model to distinguish all video frame images, and we have confirmed that we can use these results to count the exercise time and number of each motions

Key Words : Fastai,, push-up, classification

1. Introduction

1.1 Motivation

Push-ups are exercises that vary in 100 types depending on how we use our weight. In additi on to the operating position, the kinetic effect delivered to the body varies to an unexpected exte nt depending on the operating range and exercise time. Also, planning according to one's compete nce is essential because push-ups are relatively muscular exercises. As a result, users will be abl e to make significant progress in their exercise effectiveness in their personal areas if they can r ecord their own exercise posture, exercise time and number of exercises and use them to make plans. We want to explore ways to analyze and utilize push-up videos taken by users directly thr ough deep learning techniques. Through this, we expect to be able to incorporate this result into home training, which has recently attracted attention due to COVID-19.

1.2 About FastAI

The Fastai library provides a variety of models for image data analysis via deep-learning meth od. Basically, the model learns about the relationship between the image and the specified label b ased on the input image data. Among them, image classification uses this trained model to output labels to respond to new images. Lesson, provided by Fastai, showed identifying the breed of pet s by using the dataset(O. M. Parkhi et al., 2012)[1]. In addition to using a single label, it was also p ossible to produce multiple labels that were analyzed to be likely to respond to an image through multiple labels corresponding to an image. In image segmentation, the model trained by labeling m ask images corresponding to each pixel of the image. The model presents an output of labels that correspond to each pixel of the image. (Lesson used 'Cambid: Motion-based Segmentation and R ecognition Dataset by Brostow et al., 2008' to separate and analyze objects shown in car black bo x video.)

2. Point

2.1 Using image segmentation

Selection of methodologies should precede for push-up image analysis via deep learning models. We are inspired by BodyPix[2], which uses Real-Time Video to localize each structure area of the body. The model primarily separates the background from the human body and extracts body structures including the head, arms, stomach, and legs. It then visualizes and shows skeletal data from specific coordinates of the extracted area. We consider the labels that the input dataset should have by referring to segmentation models provided by BodyPix and Fastai.

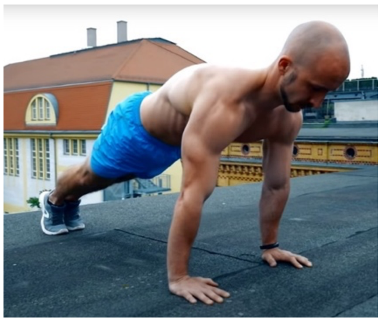


Figure 2 : Side view of push-up

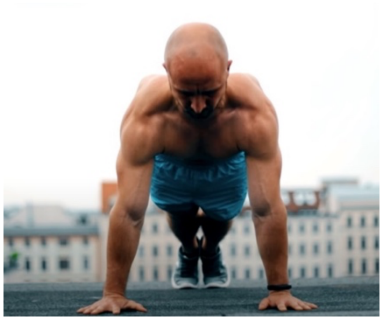


Figure 1 : Front view of push-up

[1] Oxford-IIIT Pet Dataset by O. M. Parkhi et al., 2012

[2] BodyPix: Real-time Person Segmentation in the Browser with TensorFlow.js, TensorFlow, 2019

2.1.1 Setup

To do so, the image to be given as input data must be understood from the point of view of the model. First of all, push-ups concentrate on exercise effects on the upper body. However, the push-up images taken by the user change the shape of the body significantly depending on the angle. When taking from the side, the range of motion can be fully checked, but the upper body's weight is reduced. On the other hand, images taken from the front can secure many details due to the high proportion of the upper body with high importance. Based on the front image, it can be seen that the gap between both arms widens when the posture is lowered, and the gap between the arms narrows as the posture increases. Also, it will be essential to distinguish between the left and right sides of the body for correct posture correction. This is because it is important to check whether the body is moving in a balanced manner while doing push-ups. Thus, the mask image to be labeled will need to be configured separately from the head, the shoulder, and the upper and lower body, including the arms, respectively.

We were able to obtain a dataset that meets the conditions through an organization[3] that provides body 3D modeling data. This is a total of about 23,000 datasets consisting of images of all body parts exposed, colors corresponding to each, and black-white mask images. Each body structure is localized with 31 labels, excluding backgrounds, to construct mask images. To apply this dataset to Fastai's segmentation model, we need to understand how the model processes images. Fastai's segmentation model dataizes the brightness of each pixel of the image by dividing it into three-dimensional vectors of RGB (Red, Green, Blue). A single image is replaced by a three-dimensional vector with the same size as the total number of pixels it has. The areas of the images identified by labels are recorded as elements of the tensor. We thought we could mathematically compute the center point from the area of the tensor consisting of the same elements. And this extr

[3] Unite the People – Closing the Loop Between 3D and 2D Human Representations, Christoph Lassner, 2019

acted focus is essential for bone-level image analysis.

Head	Upper Body	Lower Body	Arms	Background
🟥	🟩	🟪	🟨	⬛
CB1E40	82FF00	8742F5	FFDE59	000000

Figure 3 : Color of label with dataset configured

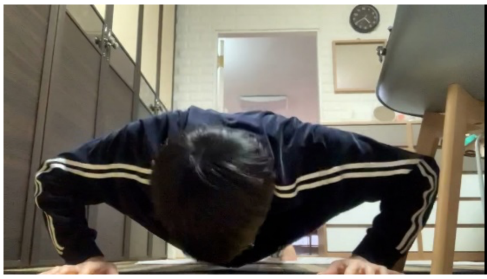


Figure 4 : Image extracted from image

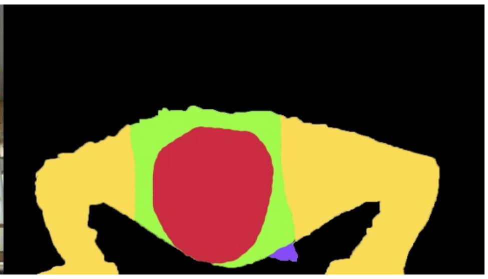
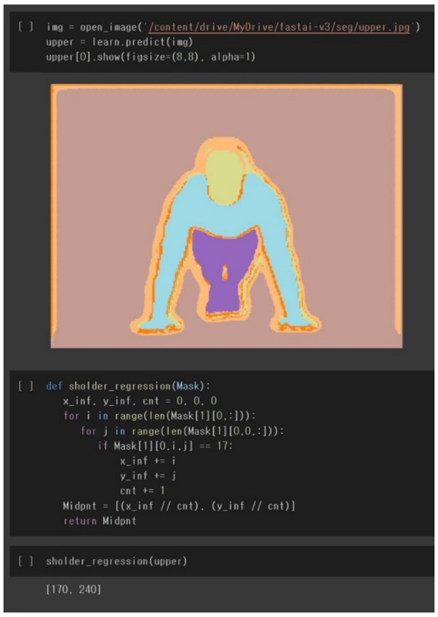


Figure 5 : Mask image

As a result of training the model using a dataset of about 200 chapters, which consists of these, we were able to obtain sufficient levels of accuracy. However, as the appearance of the background changes in the image, the extent of the body the model assumes has changed significantly. We attributed the lack of size and diversity of datasets to the lack of securing them, and thought they would be improved when utilizing real datasets. We have succeeded in deriving the coordinates of the midpoints from the upper body area that can be seen in the output, and we have also confirmed that this varies with posture.

2.1.2 Fail



```
[ ]  img = open_image('/content/drive/MyDrive/fastai-v3/seg/upper.jpg')
     upper = learn.predict(img)
     upper[0].show(figsize=(8,8), alpha=1)
```

```
[ ]  def sholder_regression(Mask):
         x_inf, y_inf, cnt = 0, 0, 0
         for i in range(len(Mask[1][0,:])):
             for j in range(len(Mask[1][0,0,:])):
                 if Mask[1][0,i,j] == 17:
                     x_inf += i
                     y_inf += j
                     cnt += 1
         Midpnt = [(x_inf // cnt), (y_inf // cnt)]
         return Midpnt
```

```
[ ]  sholder_regression(upper)

     [170, 240]
```

Figure 9 : The "sholder_regression" function to measure the midpoint of the shoulder area

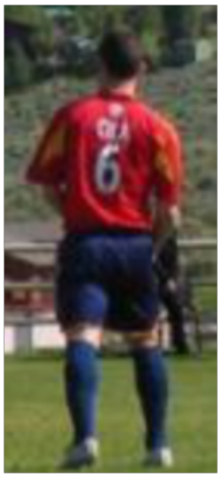


Figure 8 : Image

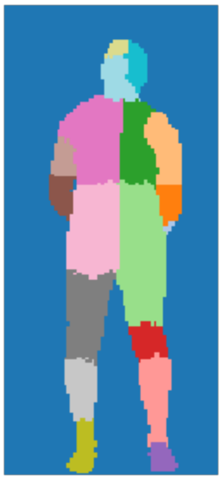


Figure 7 : Lable

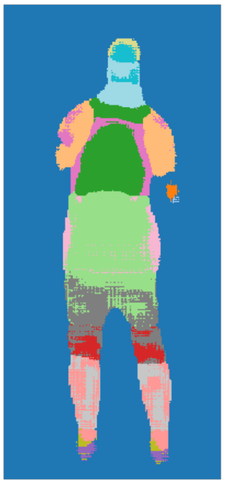


Figure 6 : Prediction

These results showed that utilizing the output of the segmentation model was meaningful enough, and we trained the model with large datasets that we had prepared in advance. The big difference is that the body structure is left and right, and consists of more detailed labels.

However, the results of using this data set were not good(Figure 6). Although we have provided sufficiently large datasets, sufficiently long learning times, and many learning times, the model has failed to distinguish between left and right differences from the locational features of the area. Furthermore, it has been found that the more detailed the labels are, the less independent the body structure of relatively small areas from the background. As in the above picture, if the upper body area is distributed throughout the image, the function of obtaining the coordinates of the focal points also fails to function. The results did not improve significantly by entering continuous images, such as those extracted from video. There was also a possibility that even if proper expectations were made for one image, it would not lead to good results for the very next image. We encounter a problem where the model does not learn properly in methodology utilizing segmentati on models through this result.

In response, we have considered approaching the ultimate objective through image clas sification. It will certainly not reach the final purpose of the push-up posture correction because no detailed domain analysis of the image is made. However, if image classificati on allows the number of movements count and exercise time measurement, it will be abl e to perform its enough role as an exercise helper app.

2.2 Using image classification

What is needed to analyze pushups is the classification results and possiblity of the scenes in th e video. To obtain this, we create a dataset to learn the learner made with Resnet34 and apply it to each frame of the image. Then, analyze the results and set the number of push-ups and time with the output.

2.2.1. Import library

```
import cv2
from fastai.vision import *
from matplotlib import pyplot as plt
import numpy as np
```

Import the required library. cv2 is used to extract and store frames from images. fastai.vision is a key library for machine learning. matplotlib creates a visual output, and numpy is necessary for analysis.

2.2.2. Class specification

```
path = Path('drive/MyDrive/dna/push-up dataset/updown')
classes = ['up','down']

for c in classes:
    verify_images(path/c, delete=True, max_size=500)

np.random.seed(42)
data = ImageDataBunch.from_folder(path, train=".", valid_pct=0.2,
ds_tfms=get_transforms(),size=224,num_workers=4).normalize(imagenet_stats)
```

Conduct a precursor work for machine learning. Specify the path where the datasets are gathered. The datasets are stored in two classes, up and down, each of which is a picture of when the arm is most stretched and bent in a pushup. The images are labeled to the class name, and the validation dataset is separated at a rate of 0.2%.

2.2.3 Training the model

A. fit one cycle

```
learn = cnn_learner(data, models.resnet34, metrics=error_rate)
learn.fit_one_cycle(4)
```

Specify learn as cnn learner. This model is trained four times using the Resent34 model. fit_one_cycle is a characteristic learning method for fastai. It is a method of learning the learning rate up and down periodically in a certain range, which dramatically increases the accuracy of learning.

B. Learning rate determination

```
learn.unfreeze()
learn.lr_find(start_lr = 1e-04)
learn.recorder.plot()
learn.fit_one_cycle(4,slice(1e-02,1e-01))
learn.save('well_done!_1')
```

It is the process of finding a suitable learning rate for learners who have passed fit_one_cycle. By looking at the graph and specifying an appropriate learning rate interval, the fit_one_cycle function learns by altering the learning rate within that interval. Once the learning is complete, we store the model for future use.

```
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix()
```

This is the necessary step to verify the learned data and confirm subsequent steps. If it is determined that it has been learned correctly, it is possible to move on to the next step, but otherwise it is necessary to find the correct learning method again. Several built-in functions of fastai are available as assistants in judgment, using the most intuitive confusion_matrix. This function tabulates the actual label of the validation data and the model's predictions.

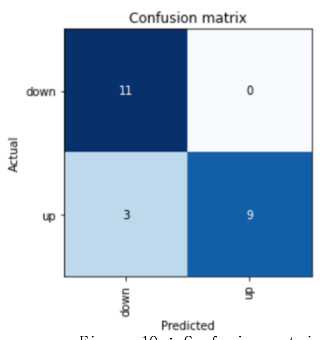


Figure 10 : Confusion matrix

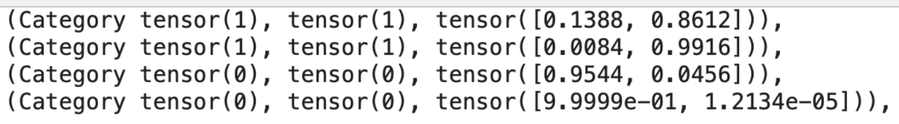
2.2.4. Frame extraction

```
vname = 'Video1.mp4'
v_path = ('/content/drive/MyDrive/dna/push-up dataset/video/%s'%vname)
f_path=('/content/drive/MyDrive/dna/push-up dataset/frames/%s'%vname)
path = Path(f_path)
path.mkdir(exist_ok=True)

vidcap = cv2.VideoCapture(v_path)
count = 0
while(vidcap.isOpened()):
    ret, image = vidcap.read()
    if np.any(image) == None : break
    cv2.imwrite('%s/frame%d.jpg'%(path,count), image)
    count += 1
vidcap.release()
```

Now, we will now apply the learned model to real-world situations. This cell is a step in extracting frames to analyze a user's image. In vname, enter the user's video name, and in v_path and f_path, specify the path where the video is stored and where the frame is stored. In real-world use, real-time images, not paths stored in the computer, will be used, but in this code, pre-recorded images are performed. Each frame of the captured video is stored under f_path.

2.2.5. Apply video

```
path=Path(f_path)
len(path.ls())
lis=[]

for i in range(len(path.ls())):
  img = open_image(Path('/content/drive/MyDrive/dna/push-up
dataset/frames/%s/frame%d.jpg'%(vname,i)))
  pre = learn.predict(img)
  lis.append(pre)
```



```
(Category tensor(1), tensor(1), tensor([0.1388, 0.8612])),
(Category tensor(1), tensor(1), tensor([0.0084, 0.9916])),
(Category tensor(0), tensor(0), tensor([0.9544, 0.0456])),
(Category tensor(0), tensor(0), tensor([9.9999e-01, 1.2134e-05])),
```

Figure 11 : Some of the elements of 'lis'

Since we have learned models and extracted images, the video can now be analyzed. Applying images stored in f_path to the learned model results in predictions and probability for each class as output. Use this to analyze the final output.

```
x = range(len(lis))
y = [float(lis[i][2][1]) for i in range(len(lis))]
plt.plot(x,y)
```

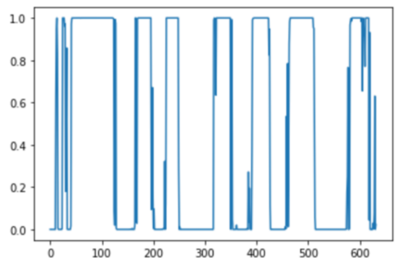


Figure 12 : Graph based on probablity

We used this because the analysis using probability was more reasonable than the zero-dimensional analysis using the prediction class. This is simply a graphical representation of the probability for the up class. This graph visually shows the user performing a push-up. The process of refining data is necessary because there are many data that are not visible from the graph but are sometimes uncertain, and the images themselves have not been refined.

```
x = range(len(lis)-10)
y = [float(lis[i][2][1]) for i in range(len(lis))]
y = [sum(y[i:i+10])/10 for i in range(len(lis)-10)]
plt.plot(x,y)
```

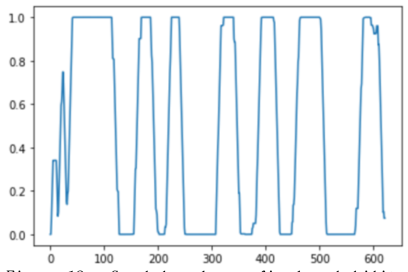


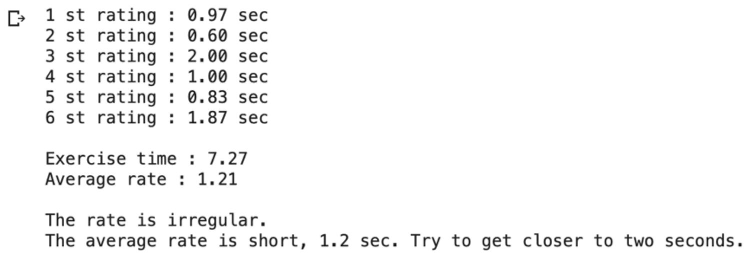
Figure 13 : Graph based on refined probability

If you look at the graph above, you can see that it looks much more refined and clean. The arithmetic mean was used with the surrounding 0.3 second data to eliminate particularly splashing data from the surrounding data.

```
cot=0
a=[]
for i in range(len(y)):
  if y[i]<0.1:
    a.append(1)
  else : a.append(0)

for i in range(len(a)-10):
  if a[i+1:i+10].count(1)==9 and a[i]==0:
    cot +=1
    print(cot)
```

The list ‘y’ contains values refined using arithmetic mean. The root of it is the probability of down. You can use this to count the number of times first. By counting from comparisons with surrounding frames, highly correlated data can be filtered even if it is not a push-up. have high accuracy.

```
ulis=[]
dlis=[]
for i in range(len(a)-10):
  if a[i+5:i+10].count(1)==5 and a[i:i+5].count(1)==0 :
    dlis.append(i)
  if a[i+5:i+10].count(1)==0 and a[i:i+5].count(1)==5 :
    ulis.append(i)
dlis, ulis

arr=[]
t=0
for i in range(cot):
  rate = (ulis[i]-dlis[i])/30
  arr.append(rate)
  print('%d st rating : %0.2f sec'%(i+1,rate))
  t += rate

m_arr = np.mean(arr)
s_arr = np.std(arr)

print('\nExercise time : %0.2f'%t)
print('Average rate : %0.2f'%m_arr)
if s_arr > 0.1 : print('\nThe rate is irregular.')
if m_arr < 1.5 : print('The average rate is short, %0.1f sec. Try to get closer
to two seconds.'%m_arr)
elif m_arr > 2.5 : print('The average rate is long, %0.1f sec. Try to get closer
to two seconds.'%m_arr)
else : print('\nWell done!')
```



```
1 st rating : 0.97 sec
2 st rating : 0.60 sec
3 st rating : 2.00 sec
4 st rating : 1.00 sec
5 st rating : 0.83 sec
6 st rating : 1.87 sec

Exercise time : 7.27
Average rate : 1.21

The rate is irregular.
The average rate is short, 1.2 sec. Try to get closer to two seconds.
```

Figure 14 : Final output. This is the screen that the user sees with the graph.

To feedback on the accuracy of the motion, measure the rate of each action. It is used to find the average time and standard deviation. If the standard deviation is too large or too small, the user is informed of the best rate because there is a problem with his exercise. The user can check the exercise by looking at the average rate and standard deviation of the rate from the output. So the next home training will be more efficient.

3. Output and application plan

As a result of using the program, it automatically counts the number of movements so that you can focus entirely on your body. Furthermore, more accurate feedback was possible due to the output of the exercise rate and the analysis results, and the graph provided visual confirmation of the results. However, it is regrettable that 200 datasets were not enough and that the samples were not diverse. Also, the limitation is that exercise is limited to push-ups. It is expected that a better training application will be created because if we can analyze using failed segmentation, we will be able to overcome this limitation, and posture correction will also be possible.

References

[1] BodyPix: Real−time Person Segmentation in the Browser with TensorFlow.js, TensorFlow, https://blog.tensorflow.org/2019/11/updated−bodypix−2.html, 2019

[2] Getting Started with Videos, OpenCV, https://docs.opencv.org/master/dd/d43/tutorial_py_video_display.html, 2017

[3] Oxford−IIIT Pet Datase, O. M. Parkhi et al., 2012

[4] Cambid: Motion−based Segmentation and Recognition Dataset, Brostow et al., 2008

[5] Weakly and Semi Supervised Human Body Part Parsing via Pose−Guided Knowledge Transfer, Hao−Shu Fang, Conell University, 2018

[6] Unite the People – Closing the Loop Between 3D and 2D Human Representations, Christoph Lassner, 2019

[7] Exercise Classification with Machine Learning, Trevor Philips, toward data science, 2019