Unet Based Cyclist Posture Analysis System

Pee Kian Soon, Mahathir Bin Humaidi and Dong Xiaoguang

***Abstract*— The Cyclist Posture Analysis System is aimed to provide a solution which is able to measure rider's frontal area (A_p) directly from digital photographs captured by affordable smartphones. Effective recognition and segmentation will be the key to identify and pull out the area of pixels belonging to the rider in a digital photograph. An SVM model is placed following the U-Net model to do the inference of the frontal area in actual.**

I. BACKGROUND

Kenyan Riders was founded in 2009 and is a pioneering bicycle racing team based in Iten, Kenya. It is a social enterprise that drives economic change and social good through cycling. One of the key roles of the Kenyan Riders project is to introduce the sport of cycling to Kenya, and from that, to spot and develop cycling talent. We organise local bicycle races to encourage the greater use of the bicycle so that more children, who live further away, can attend school. The races are also a means to identify talented children and award them scholarships. From that we plan to give people careers as professional cyclists, bike mechanics, physiotherapists, coaches, and spin instructors.

In cycling, CdA is important. It is an abbreviation for the coefficient of aerodynamic drag. It's a dimensionless number (no units), which is the result of a body's drag size, shape and surface texture. The above cyclist going 36 km/h with 200 watts on a flat course has a CdA of .28. By improving that CdA by 10% and dropping it to .252 on that same 200 watts, the cyclist will go 37.5 km/h. With a world class CdA of .205, he will go 40 km/h on that same 200 watts.

CdA depends on the size, shape and surface texture of the object. The most important component of the size is the frontal area, i.e., the area in contact with the wind from the front. The CdA measure enables the cyclist to find the optimal position to improve his aerodynamic efficiency while preserving his ability to generate power. It also allows her or him to make equipment choices such as helmet and clothing. The cyclist can thus experiment to find the most efficient setup for him.

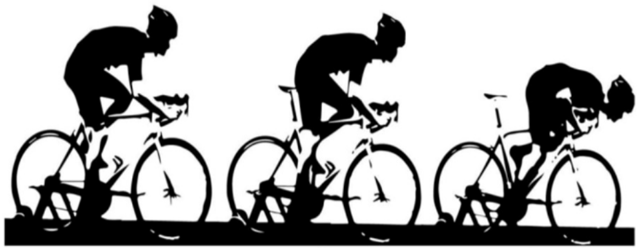


Fig. 1: cyclist to find the optimal position to improve aerodynamic efficiency

After a few years of running the project, we soon realised that many of the cyclists were wasting energy through aerodynamic inefficiency. That usually gets solved through hiring an aero coach and access to a wind tunnel. Unfortunately, the nearest wind tunnel is 3800km away in South Africa. So we developed a protocol to measure the co-efficient of drag of the cyclists. Over the years, the method has been refined and tested, and proven to be reliable and predictive.

For now, there is no way for cyclists to know their CdA at a consumer level. Therefore, we would now like to automate our protocol into an app that could inform cyclists and triathletes about their CdA. The aim is to commercialise the app and eventually use the revenue to sponsor the team and fund the grassroots projects in Kenya which we are most passionate about.

II. PROJECT PIPELINE

A. Data collection and mask creation stage

This stage is further divided into three sub-sections:

1) Photo collection: Take a photo of a cyclist's front view with a phone camera(28mm focal length). Take another photo of the same cyclist front view (same posture as previous) with a telephoto camera (200mm focal length). Collect 100 sets of the pair of photos.

2) Camera photo mask creation: For each of the images taken with the phone camera, create a manual mask using label me annotation tool. Mask should consist of 3 regions: cyclist and bike(excluding wheel), wheel, background.

3) Telephoto frontal area calculation: For each of the images taken with the telephoto camera, calculate the frontal area of the cyclist. Actual area can be derived from the frontal area which served as a known variable.

B. *Neural network training and comparing stage*

During this stage, three models are explored and compared. The models chosen are PSPNet, UNet and Mask R-CNN. The model with the most comprehensive outcome is chosen to be applied to the further inference stage. The output of this stage is the masks with 3 classes segmentation which are cyclist and bike (excluding wheel), wheel, background.

C. *Pixel Number Inference stage*

In this stage, input data will be frontal view images of cyclists taken with phone cameras. The model will inference the cyclist front view mask as well as normalised area of cyclist front view. Normalised area of cyclist front view = Area of cyclist / Area of wheel.

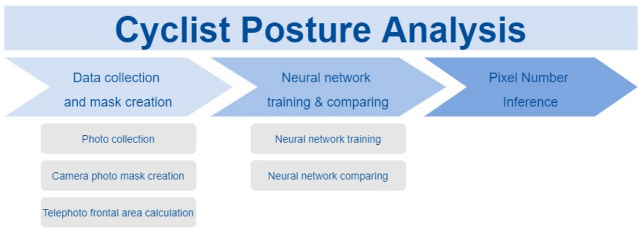

Cyclist Posture Analysis
Data collection and mask creation
Neural network training & comparing
Pixel Number Inference
Photo collection
Camera photo mask creation
Telephoto frontal area calculation
Neural network training
Neural network comparing

Fig. 2: Project pipeline

III. Working details of the model

A. *Overview*

The Client Kenyan Riders would like to develop a mobile training app for professional and armature cyclists. As a racing team company, they developed a protocol to measure the CdA of the cyclists. Kenyan Riders would like to implement this protocol on a phone app to enable cyclists to determine their own CdA in a hassle-free manner. The app will require a precise dimensions (true area) of the frontal view of the cyclist in his or her usual competitive cycling posture. These will be provided using the cyclist's mobile phones. However due to the extreme surface curvature of wide angle lens, images taken via phone inevitably suffer from barrel distortion. This distortion can be avoid by taking the photo with a zoom-in lens mounted on a standalone camera (SLR etc.) but this defeats the app's hassle-free purpose. To resolve the distortion issue, our team proposed a deep learning network that takes in the distorted image of the cyclist posture as input and generates its corrected dimensions as output. The network will consist of two separate models; a segmentation model for generating segmentation masks and a linear regression model for predicting the correct dimensions.

Annotation tools Labelme and python code are used to produce segmentation masks from cyclist posture images taken from mobile phone. Each segmentation mask consists of 3 separate regions or classes namely "Cyclist with bike" (Class A), "Front wheel" (Class B) and "Background" (Class C). The images and their corresponding masks are used as training data for training a multi-class segmentation model to

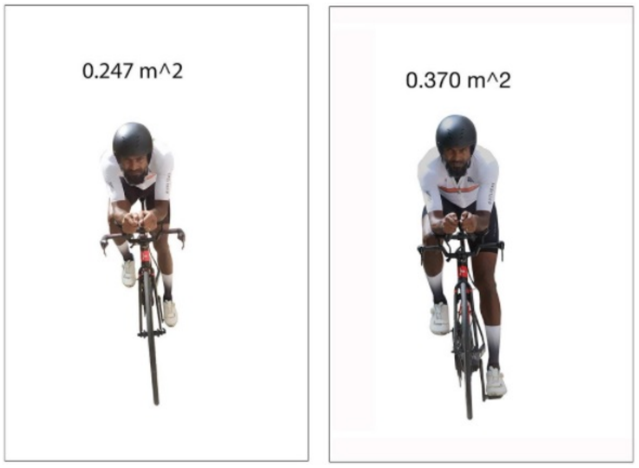


Fig. 3: with same posture, the first image is distorted from a digital camera

output a segmentation mask containing the above mentioned 3 classes. The numbers of pixels that fall under Class A and Class B in the segmentation mask are calculated. To account for variation in object sizes among images (due to variation in object to camera distance during photo taking sessions), the pixel count is normalised by dividing the pixel summation by the number of pixels that falls under Class B.

$$C_{\text{Np}} = \frac{C_{\text{Ap}} + C_{\text{Bp}}}{C_{\text{Bp}}} \tag{1}$$

Attributes are:
1. C_{Np} - Count of Normalized Pixels
2. C_{Ap} - Count of Pixels in Class A
3. C_{Bp} - Count of Pixels in Class A

The second part of the network consists of a regression model that predicts the true frontal area from a normalised pixel count.

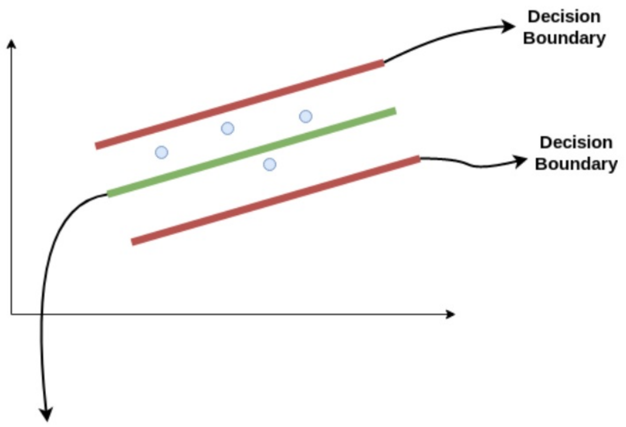


Fig. 4: Consider these two red lines as the decision boundary and the green line as the hyper plane. Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line. Our best fit line is the hyper plane that has a maximum number of points.

The training data to train this regression model comes from "ground truth" area derived from images taken with a zoom-in lens SLR camera and normalised pixel count generated from the earlier segmentation model.

B. *Comparison - U-Net vs. Mask R-CNN*

There are 2 model structures in option which are U-Net and Mask R-CNN. These 2 models are separately trained and implemented with the cyclist dataset. After training, it is found that both of the 2 models can successfully segment the cyclist.The masking results are similar to each other.

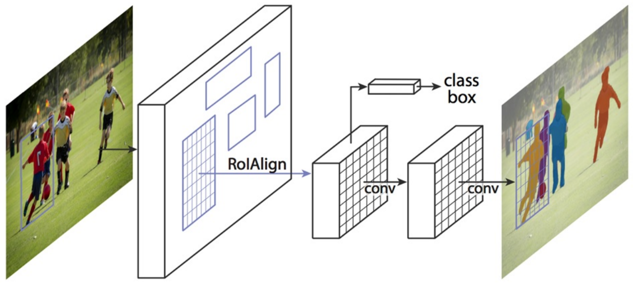
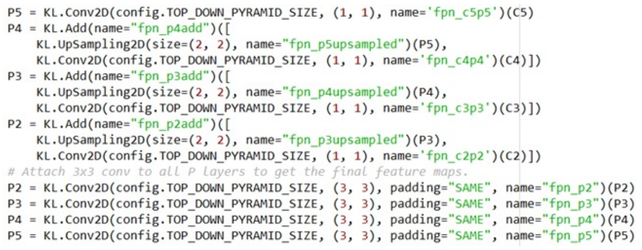


Fig. 5: Mask R-CNN structure

However when it comes to detailed parts eg. bicycle handle grip, U-Net has better performance in this case. Some research has also been done to explain this result difference between these 2 models. It turns out that our dataset is relatively small so that U-Net can perform better prediction via small dataset training while Mask R-CNN is likely to have better prediction on large dataset. Another thing is model complexity and training time, Mask R-CNN consumes more resource as the network is more complicated.



```
P5 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c5p5')(C5)
P4 = KL.Add(name="fpn_p4add")([
    KL.UpSampling2D(size=(2, 2), name="fpn_p5upsampled")(P5),
    KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c4p4')(C4)])
P3 = KL.Add(name="fpn_p3add")([
    KL.UpSampling2D(size=(2, 2), name="fpn_p4upsampled")(P4),
    KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c3p3')(C3)])
P2 = KL.Add(name="fpn_p2add")([
    KL.UpSampling2D(size=(2, 2), name="fpn_p3upsampled")(P3),
    KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c2p2')(C2)])
# Attach 3x3 conv to all P layers to get the final feature maps.
P2 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME", name="fpn_p2")(P2)
P3 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME", name="fpn_p3")(P3)
P4 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME", name="fpn_p4")(P4)
P5 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME", name="fpn_p5")(P5)
```

Fig. 6: FPN Layers of Mask R-CNN

Regarding to above reasons, U-Net is finally chosen as the project model.

C. *Working Structure of U-Net*

1) Overview: U-Net architecture contains two paths. First path is the contraction path (also called as the encoder) which is used to capture the context in the image. The encoder is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions. Thus it is an end-to-end fully convolutional network (FCN), i.e. it only contains Convolutional layers and does not contain any Dense layer because of which it can accept image of any size.

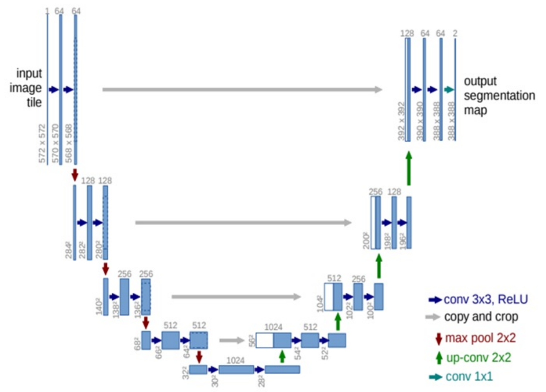


Fig. 7: U-Net structure

2) Strategies Used in Creating and Training Project Segmentation Model: Essential points:
1. UNET model with pretrained MobilenetV2 as encoder
2. Decoder structure (number of convolution filters, dropout etc)
3. Transfer Learning (retraining follow by fine tuning)
4. Image Augmentation (flip horizontal, contrast, translation)
5. Various optimisers (Adam, SGD, Cyclic)
6. Loss functions (cross entropy, dice loss)
7. Batch size, epoch size, image size (2-32; 50; 112x112, 224x224,448x448)
8. Class weights for imbalance class
9. Learning rate (1e-4 to 1e-2)
Sampling:
No. of training samples: 105
No. of validation samples: 20
No. of testing samples 5

D. *Regression Model*

In Cyclist Posture Analysis System, SVM is used as regression model. The input of SVM is the frontal area (number of pixels) of the segmented cyclist in digital camera photo while output is the predicted true frontal area (number of pixels).From there it is possible to calculate out the real frontal area of the cyclist.

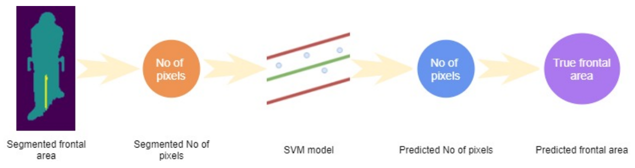


Fig. 8: SVM Model

IV. Tools Have Used

For U-Net, Labelme is used for annotation while Colab is used for coding and training. For Mask R-CNN, VGG Image Annotator is used for annotation and traing is done in local device.

V. Data Preparation

A. Raw Data

Frontal photographs of individual cyclists on a stationary bicycle were taken with a telephoto lens camera and a camera phone respectively. The photos were paired and numbered with suffixes to indicate the type of camera. For example, the paired photos of the cyclist below are numbered 116_a and 116_b respectively. The digits (116) refer to an individual cyclist and the suffixes refer images using telephoto (a)camera - Fig.9 or (b)phone camera - Fig.10.

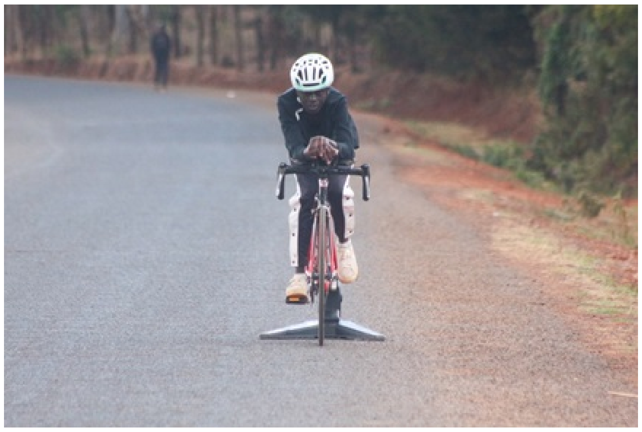


Fig. 9: Telephoto camera image(116_a)

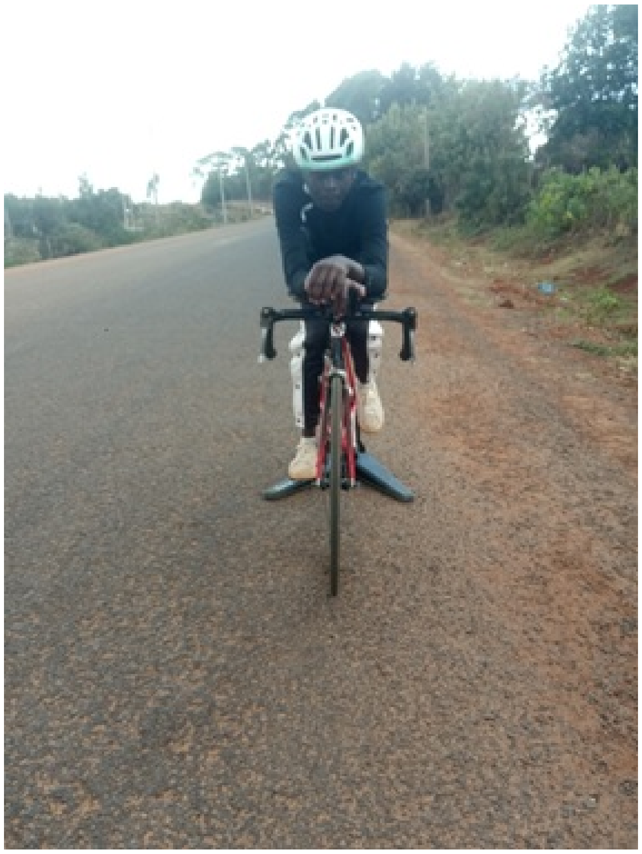


Fig. 10: Telephoto camera image(116_b)

B. Mask creation

The open source image annotation software, LabelMeAnnotationTool, is used to create two masks,i) cyclist and bicycle and ii) front wheel, from the raw images [ref].

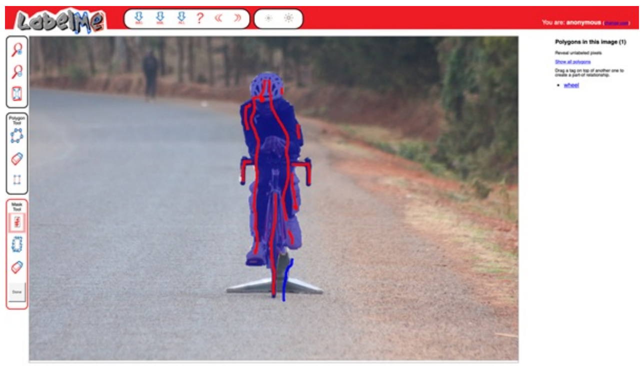


Fig. 11: Labelme (https://github.com/CSAILVision/LabelMeAnnotationTool)

The red scribbles are drawn within the shape of the target object while the blue scribbles are drawn in the area that is excluded. The tool extracts the mask shape and saves it in png format. Suffix "mask_0" is added to the file name for the Total Frontal mask - Fig. 12.

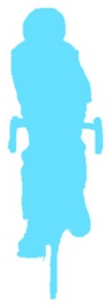


Fig. 12: Annotated mask

The second annotation part is the front wheel.

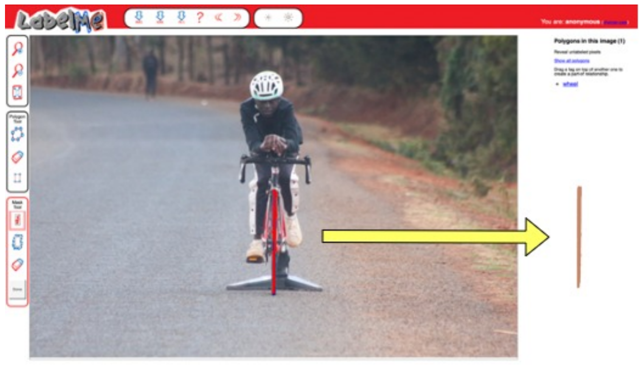


Fig. 13: Front Wheel mask of 116a.jpg

105 sets of images (65 sets of telephoto and 40 sets of phone camera) were selected for the training dataset, and 25 sets of images (16 sets of telephoto and 9 sets of phone camera) were selected for the validation dataset.

When it comes to the final determination of the frontal area, the frontal mask and front wheel mask are combined into calculation.

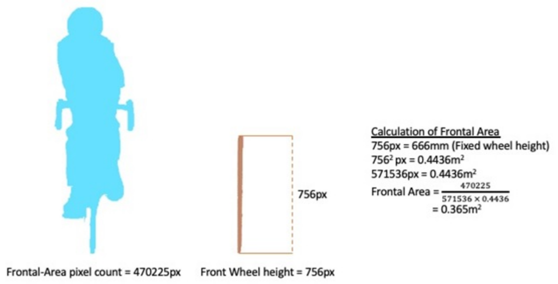

Calculation of Frontal Area
756px = 666mm (Fixed wheel height)
756² px = 0.4436m²
571536px = 0.4436m²
Frontal Area = 470225 / 571536 × 0.4436
= 0.365m²
756px
Frontal-Area pixel count = 470225px
Front Wheel height = 756px

Fig. 14: Fontal area calculation

VI. Training

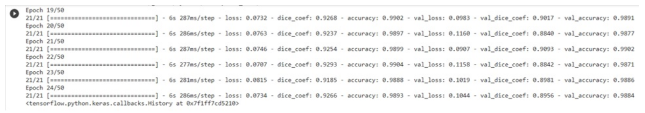
A. Benchmark Model

The project used a MobileNetV2 network pretrained on Imagenet as the encoder. The pre-trained weights are downloaded from https://github.com/fchollet/deep-learning-models/releases/download/v0.6/.

We identified the layers within MobileNetV2 to be used as skip connections (to be concatenate) to the later layers in the decoder. The skip connection layers are namely ”input_image”, ”block_1_expand_relu”, ”block_3_expand_relu” and ”block_6_expand_relu”. The feature map spatial dimensions of these layers are 224x224, 11x112, 56x56 and 28x28. The decoder consists of 4 similar blocks of Conv2DTranspose + concatenate + conv2d_block + Dropout operations. The Conv2DTranspose layer up-sample the feature map to twice its input size and then concatenate it with the skip connection coming from the encoder. The conv2d_block consists of a series of normal convolution, batch normalisation and ReLU (rectified linear activation function) for learning the image representation. As part of transfer learning, weights of all encoder layers were freeze during training; only weights of decoder layers were updated by backpropagation.

Initial configuration of Conv2DTranspose filter numbers along the decoder path was 96, 84, 72, 60. Zero drop-out was employed after each conv2d_ block. Adam optimiser together with Dice Loss were initially used to compile the model. Adam optimiser was chosen as it is well known to be highly versatile and works in many scenarios. Initial learning rate was set at 1e-3. Dice loss was chosen as it is reputed to be better performing in imbalanced class dataset. The segmentation problem involves segmenting three classes: cyclist, wheel, and background; the wheel class accounts for less than 5% of the total image pixel count.

Due to the small training dataset available, a batch size of 5 was initially used. Epoch was fixed at 50 with early stopping while network input size was set at MobileNetV2 default value of 224x224. Image Augmentation and class weights were not implemented for the benchmark model.

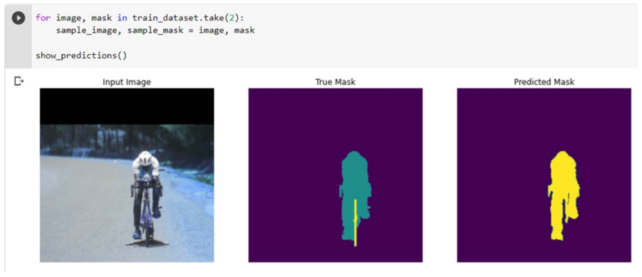


```
Epoch 19/50
21/21 [==============================] - 6s 287ms/step - loss: 0.0732 - dice_coef: 0.9268 - accuracy: 0.9902 - val_loss: 0.0983 - val_dice_coef: 0.9017 - val_accuracy: 0.9891
Epoch 20/50
21/21 [==============================] - 6s 286ms/step - loss: 0.0763 - dice_coef: 0.9237 - accuracy: 0.9897 - val_loss: 0.1160 - val_dice_coef: 0.8840 - val_accuracy: 0.9877
Epoch 21/50
21/21 [==============================] - 6s 287ms/step - loss: 0.0746 - dice_coef: 0.9254 - accuracy: 0.9899 - val_loss: 0.0907 - val_dice_coef: 0.9093 - val_accuracy: 0.9902
Epoch 22/50
21/21 [==============================] - 6s 277ms/step - loss: 0.0707 - dice_coef: 0.9293 - accuracy: 0.9904 - val_loss: 0.1158 - val_dice_coef: 0.8842 - val_accuracy: 0.9871
Epoch 23/50
21/21 [==============================] - 6s 281ms/step - loss: 0.0815 - dice_coef: 0.9185 - accuracy: 0.9888 - val_loss: 0.1019 - val_dice_coef: 0.8981 - val_accuracy: 0.9886
Epoch 24/50
21/21 [==============================] - 6s 286ms/step - loss: 0.0734 - dice_coef: 0.9266 - accuracy: 0.9893 - val_loss: 0.1044 - val_dice_coef: 0.8956 - val_accuracy: 0.9884
<tensorflow.python.keras.callbacks.History at 0x7f1ff7cd5210>
```

Fig. 15: U-Net training metrics

With this configuration, the model converged at 24 Epochs and achieved validation dice coefficient of 0.8956. The model was unable to pick up the wheel class during both training and testing stage. Average validation dice coefficient for a test sample size of 5 is 0.8086.

Below shows the sample U-Net segmentation on training data.



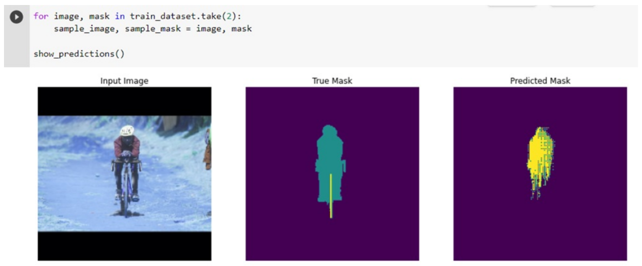
```
for image, mask in train_dataset.take(2):
    sample_image, sample_mask = image, mask

show_predictions()
```

Fig. 16: mask prediction of training sample

B. Parameter Optimization

Image augmentation was applied while keeping the configuration unchanged. Horizontal flip, random-contrast and random-zoom operations was tested out separately on the training dataset. Interesting, the model performance did not improve and in fact deteriorated by a few percentage points especially when random zoom was applied.

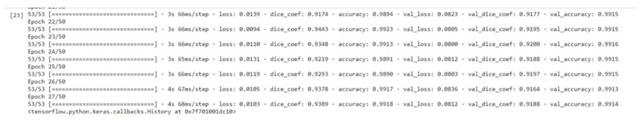


```
for image, mask in train_dataset.take(2):
    sample_image, sample_mask = image, mask

show_predictions()
```

Fig. 17: mask prediction with image augmentation

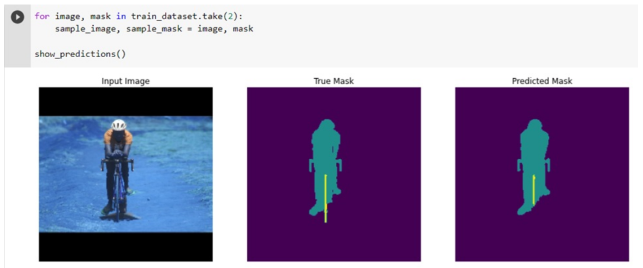
Influence of batch size was then investigated. Batch size was progressively reduced from 32 to 2 and the best performing batch size was found to be 2. Two other optimizers, SGD and cyclic with various initial learning rates from 1xe-2 to 1xe-4 were also tried out. The best performing was Adam with initial learning rate of 1xe-3. Next, the size of neutral network was progressively reduced by reducing the number of Conv2DTranspose filters in the decoder layer. The optimum filter number was determined to be 48, 42, 36, 30 - half the size of the initial configuration. To further mitigate the effect of class imbalance, a higher-class weight was given to the minority wheel class. With the above tunning, the model was able to pick up the minority wheel class. However, a significant number of wheel pixels were still miss-classified as background pixels. Below results were obtained with these parameter settings.



```
[23] 53/53 [==============================] - 3s 66ms/step - loss: 0.0139 - dice_coef: 0.9174 - accuracy: 0.9894 - val_loss: 0.0823 - val_dice_coef: 0.9177 - val_accuracy: 0.9915
     Epoch 22/50
     53/53 [==============================] - 3s 66ms/step - loss: 0.0094 - dice_coef: 0.9443 - accuracy: 0.9923 - val_loss: 0.0805 - val_dice_coef: 0.9195 - val_accuracy: 0.9915
     Epoch 23/50
     53/53 [==============================] - 3s 66ms/step - loss: 0.0110 - dice_coef: 0.9348 - accuracy: 0.9913 - val_loss: 0.0800 - val_dice_coef: 0.9200 - val_accuracy: 0.9916
     Epoch 24/50
     53/53 [==============================] - 3s 65ms/step - loss: 0.0131 - dice_coef: 0.9219 - accuracy: 0.9891 - val_loss: 0.0812 - val_dice_coef: 0.9188 - val_accuracy: 0.9915
     Epoch 25/50
     53/53 [==============================] - 3s 66ms/step - loss: 0.0119 - dice_coef: 0.9293 - accuracy: 0.9890 - val_loss: 0.0803 - val_dice_coef: 0.9197 - val_accuracy: 0.9915
     Epoch 26/50
     53/53 [==============================] - 4s 67ms/step - loss: 0.0105 - dice_coef: 0.9378 - accuracy: 0.9917 - val_loss: 0.0836 - val_dice_coef: 0.9164 - val_accuracy: 0.9913
     Epoch 27/50
     53/53 [==============================] - 4s 68ms/step - loss: 0.0103 - dice_coef: 0.9389 - accuracy: 0.9918 - val_loss: 0.0812 - val_dice_coef: 0.9188 - val_accuracy: 0.9914
     <tensorflow.python.keras.callbacks.History at 0x7f701001dc10>
```

Fig. 18: U-Net training metrics with augmentation

Essential points:
• Batch size: 2
• Image Size: 224 x 224
• No. of Conv2DTranspose filters in decoder layer: 48, 42, 36, 30
• Dropout (after each conv2d_ block): 0.1
• Optimiser: Adam with initial learning rate of 1xe-3
• Loss Function: Dice Loss
• Class-weights ratio (background: cyclist: wheel): 1:1:4
• Image Augmentation: None



```
for image, mask in train_dataset.take(2):
    sample_image, sample_mask = image, mask

show_predictions()
```

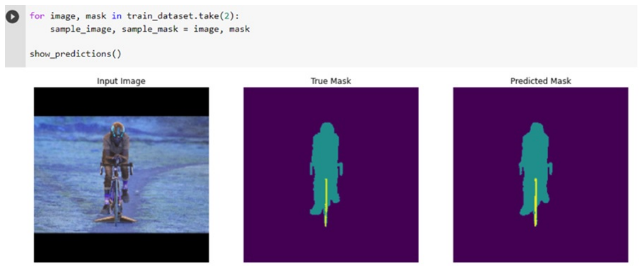
Fig. 19: mask prediction of training sample with augmentation

The tunning not only pick up the minority wheel class, but also increased the accuracy of prediction. The accuracy is increased from 0.966 to 0.970.

C. Final Model

The same parameter optimization process was tried out on cross entropy loss function. Interesting the result was much better than using dice loss function. The model was able to accurately classify the wheel pixels even without class-weights application. Further the performance (as seen in the mask prediction of test image) was slightly better with a larger image size. Though the model misclassified a few cyclist pixels as background pixels.

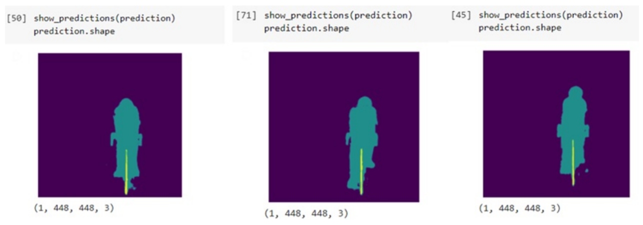
Below results were obtained with these parameter settings.
• Batch size: 2
• Image Size: 448 x 448
• No. of Conv2DTranspose filters in decoder layer: 48, 42, 36, 30
• Dropout (after each conv2d_ block): 0.1, 0.2, 0.3
• Optimiser: Adam with initial learning rate of 1xe-3
• Loss Function: Cross Entropy
• Class-weights ratio: None
• Image Augmentation: None



```
for image, mask in train_dataset.take(2):
    sample_image, sample_mask = image, mask

show_predictions()
```

Fig. 20: mask prediction of training sample with final model



```
[50] show_predictions(prediction)
     prediction.shape
(1, 448, 448, 3)

[71] show_predictions(prediction)
     prediction.shape
(1, 448, 448, 3)

[45] show_predictions(prediction)
     prediction.shape
(1, 448, 448, 3)
```

Fig. 21: mask prediction of testing sample with final model

VII. Challenge

The problem domain is quite niche and there no publicly available dataset. All training data has to be obtained from the sponsor company. However the company did not have any ready images so they had to get their cycling athletes in Kenya to pose as photoshoot model for the cycling posture photographs. The data generation (photo taking) process was very slow. In the end, the project team was provided with only 200 sets of photographs. Many of the images could not be used due to poor resolution, clutter background and non-synchronised posture between phone and tele-photo lens images. After discarding the non-usable images, the team was only left with 120 images and this quantity had to be shared among the training, validation and testing dataset.

In order to maintain a mapping consistency between phone images and tele-photo lens images, the project team proposed a fixed distance between the subject and the camera; 1.5m for phone and 10m for tele-photo lens. Unfortunately, it was discovered later in the project stage that these requirement was not adhered to by the photo taking personnel. There was quite a significant variant in the actual subject to camera distances.

Image augmentation method was tried out to artificially expand the available dataset for model training. But the model did not response well to image augmentation even when applying a simple horizontal-flip or varying contrast operation to the dataset.

VIII. Future Improvement

Improve on the data collection (photo-taking) process to ensure all project requirements pertaining to dataset are well adhered to. A minimum size of 200 images should be prepared for the model training.

To try out other segmentation model particularly PSPNET and DeepLab. The team did try out an implementation of PSPNET in Keras by Divamguta (https://github.com/divamgupta/image-segmentation-keras) but the result was not promising as being a ready-to-use depository, hypermeters were not easily changeable without adjusting much of existing code. As for DeepLab, most of the existing implementations were in Pytorch which the team was not familiar with. The team realised many existing models especially newer ones were mainly implemented in Pytorch, not easy to find Keras implementation. The team would like to pick up Pytorch in the near future so as to widen our skill repertoire.

The app should be versatile enough to compute accurate output even if the image is not taken from the above mentioned 1.5m subject distance. Since the physical wheel dimension is fixed and known upfront, its corresponding pixel count variation with respect to subject distance can be utilised to derive some form of neutral network independent distance correction algorithm or interpolation formula. A more ambitious possibility is to employ pose estimation technique (OpenPose etc) to provide real time posture correction feedback to the user.

REFERENCES

[1] Olaf Ronneberger, Philipp Fischer, Thomas Brox *[U-Net: Convolutional Networks for Biomedical Image Segmentation], https://arxiv.org/abs/1505.04597*.

[2] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, Jiaya Jia *[Pyramid Scene Parsing Network], https://arxiv.org/abs/1612.01105*.

[3] *[DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs], http://liangchiehchen.com/projects/DeepLab.html*.

[4] *[Quick intro to semantic segmentation: FCN, U-Net and DeepLab], https://kharshit.github.io/blog/2019/08/09/quick-intro-to-semantic-segmentation*.

[5] Pallawi *[Understand Semantic segmentation with the Fully Convolutional Network U-Net step-by-step], https://medium.com/@pallawi.ds/understand-semantic-segmentation-with-the-fully-convolutional-network-u-net-step-by-step-9d287b12c852*.

[6] Yann Le Guilly *[Semantic Segmentation with tf.data in TensorFlow 2 and ADE20K dataset], https://yann-leguilly.gitlab.io/post/2019-12-14-tensorflow-tfdata-segmentation*.

[7] Andrew Ng, Kian Katanforoosh *[Building a data pipeline], https://cs230.stanford.edu/blog/datapipeline*.

[8] Derrick Mwiti *[Image Segmentation in 2021: Architectures, Losses, Datasets, and Frameworks], https://neptune.ai/blog/image-segmentation-in-2020*.

[9] *[Loss Function Reference for Keras & PyTorch], https://www.kaggle.com/bigironsphere/loss-function-library-keras-pytorch*.

[10] *[image-segmentation-keras], https://github.com/divamgupta/image-segmentation-keras*.

[11] *[How Air Resistance of the Cyclist Affects Cycling Speed], https://ridefar.info/bike/cycling-speed/air-resistance-cyclist/*.

[12] Rainer Pivit *[Drag Force Formulas], https://www.sheldonbrown.com/rinard/aero/formulas.html*.