Small Vehicle Image Classification for Pedestrian-Friendly Lanes and Streets

Deep Learning Assignment 2

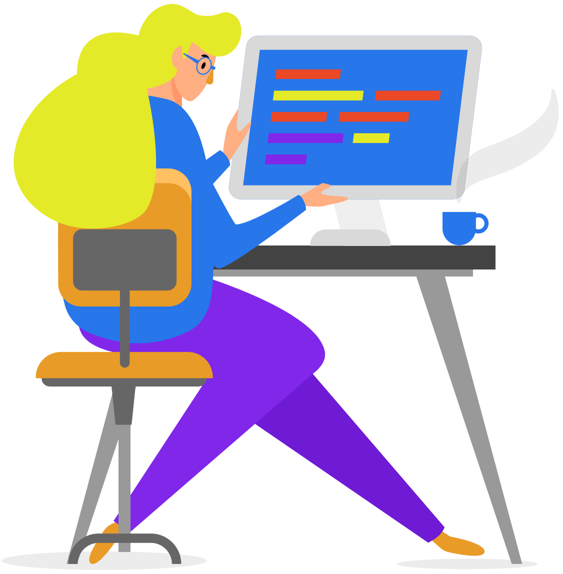
Highlights

- This project compare 5 difference transfer learning models (VGG16, EfficientNetB7, MobileNetV2, ResNet50, InceptionResNetV2) for 3 classes classification problem (mountain bike, tricycle and tuk tuk)

- You will see what the performance would be for transfer learning

- The interesting question is whether a more complicated model would result a better performance? You can find the answer in this project

- This project applying Grad-CAM to VGG16 model to understand how the model classifies each class



Agenda

01 Introduction

02 Data Scraping and Preparation

03 Network architecture and Training

04 Model Evaluation

05 Ablation Study (Grad-CAM)

06 Discussion and Conclusion

Why these small vehicles for image classification?

Cities across the globe become more welcoming to pedestrians and tend to reduce motorized transport amid the self-isolation to curtail the COVID-19 pandemic.

This allows citizens to return to the street at a social distance rather than banning traffic.

Many strategy execution processes for pedestrian-friendly streets are employed nowadays. For example, they provide adequately sized sidewalks and amenities for pedestrians and transit riders. They ensure a good co-existence between motorists and pedestrians.

However, pedestrian's experience should be more pleasant and any inconvenience or danger should be minimized.

Why these small vehicles for image classification?

There are other ways to enhance pedestrian-friendly routes and design to reduce other smaller vehicle-bikes such as mountain bikes, tricycles and Thai tuk-tuk accessing these areas.

Several tech organizations and various disciplines have benefited from CNN and image processing techniques.

Hence, we would like to apply the image classification techniques via CNN to fill this gap and prevent other unwanted vehicles.

ImageNet was also used to explore this because of an extensive image database that has helped advance CNN's research into computer vision and deep learning.

Purpose

The aims of this study are:

- To study multi-label classification task in the effectiveness of small vehicle image classification using five CNN models for pedestrian-friendly lanes and streets *
- To compare the effectiveness of small vehicle image classification using five CNN models for pedestrian-friendly lanes and streets *

*** The five CNN models in this study are:**
1)VGG-16 2) EfficientNetB7 3) MobileNetV2 4)ResNet50 5)InceptionResNetV2
**** These models are randomized to investigate the results according to our class lesson.**

Classes

The dataset contains 3 different classes:

- 1. Mountain bike, all terrain-bike, off-roader (class id on IMAGENET: 671)
- 2. Tricycle, trike, velocipede (class id on IMAGENET: 870)
- 3. Tuk tuk (no class on IMAGENET)

Image scraping

```python
timeStarted = time.time()
while True:

    imageElement = driver.find_element_by_xpath("""//*[@id="Sva75c"]/div/div/div[3]/div[2]/c-wiz/
    imageURL= imageElement.get_attribute('src')

    if imageURL != previewImageURL:
        #print("actual URL", imageURL)
        break

    else:
        #making a timeout if the full res image can't be loaded
        currentTime = time.time()

        if currentTime - timeStarted > 10:
            print("Timeout! Will download a lower resolution image and move onto the next one")
            break


#Downloading image
try:
    download_image(imageURL, folder_name, i)
    print("Downloaded element %s out of %s total. URL: %s" % (i, len_containers + 1, imageURL))
except:
    print("Couldn't download an image %s, continuing downloading the next one"%(i))
```

```
Waiting...
300
Downloaded element 1 out of 301 total. URL: https://upload.wikimedia.org/wikipedia/commons/6/6e/Ve
Downloaded element 2 out of 301 total. URL: https://cdn.britannica.com/69/19469-004-9BCC238A/Veloc
es-M-Ives-1869.jpg?w=400&h=300&c=crop
Downloaded element 3 out of 301 total. URL: http://img.kansasmemory.org/00617259.jpg
Downloaded element 4 out of 301 total. URL: https://merriam-webster.com/assets/mw/images/gallery/g
e1698295689-2097-c406a28e99fca87db3c0feac78b61723@1x.jpg
Downloaded element 5 out of 301 total. URL: https://upload.wikimedia.org/wikipedia/commons/c/ce/Ve
Downloaded element 6 out of 301 total. URL: https://i0.wp.com/ageofrevolution.org/wp-content/uploa
pg?fit=2569%2C1788&ssl=1
Downloaded element 7 out of 301 total. URL: https://www.ccpl.org/sites/default/files/Goddard_veloc
Downloaded element 8 out of 301 total. URL: https://img.pixers.pics/pho_wat(s3:700/FO/43/64/58/81/
9af53f14244bfd93f7ff.jpg,700,670,cms:2018/10/5bd1b6b8d04b8_220x50-watermark.png,over,480,620,jpg)/
Downloaded element 9 out of 301 total. URL: http://dict.drkrok.com/wp-content/uploads/2016/07/velo
Downloaded element 10 out of 301 total. URL: https://www.prints-online.com/p/164/latest-style-amer
60.jpg.webp
Downloaded element 11 out of 301 total. URL: https://wikiimg.tojsiabtv.com/wikipedia/commons/thumb
de.jpg/225px-The_American_Velocipede.jpg
```

The dataset was mainly scraped from the open dataset and free for use, for instance, pixabay, in which the composition of the images didn't have an identity that can identify the person/people in the images unless the images have the URL reference.

Image preparation

```
for filename in arr:
    file_destination = folder_path + filename
    img = cv2.imread(file_destination)
    try:
        if img.shape[0] < 224 or img.shape[1] < 224 or img.shape[2] != 3:
            print(filename, img.shape)
    except:
        print(filename)
```

```
x = []
y = []

for filename in arr:
    file_destination = folder_path + filename

    img = cv2.imread(file_destination)
    RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #CAUTION! SAVED IN RGB NOT BGR
    img = cv2.resize(RGB_img, desired_size)
    #print(img.shape)
    plt.imshow(img)
    x.append(img)
    y.append(1)

print(x)
print(y)
```

```
[array([[[124, 111,  77],
         [134, 119,  91],
         [ 92,  76,  53],
         ...,
         [ 81,  71,  59],
         [ 85,  73,  61],
         [ 89,  77,  63]],

        [[ 88,  75,  40],
         [116, 101,  71],
         [ 95,  79,  55],
         ...,
         [ 78,  68,  56],
         [ 83,  71,  59],
         [ 90,  78,  66]],
```

The selected image will be converted to size 224*224 and converted to an array with labels indicating which class they belong to. With this process, the images whose pixel size are smaller than the desired size (224*224) are eliminated. Eventually, there are 200 images/arrays left per class.

Training

In order to show the big picture of training process, we'll introduce the VGG-16 model to be the based model for transfer-learning and fine-tuning with initial set up as shown below.

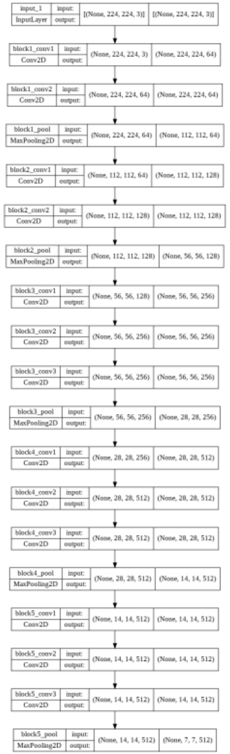
- Python 3.8.5
- Numpy 1.22.0 fixed random seed = 1234
- Tensorflow 2.7.0 fixed random seed = 5678

Training

1. Load the VGG-16 model as pretrained model with only feature extractor section

```
#Load VGG-16 model
vgg_extractor = tf.keras.applications.vgg16.VGG16(weights = 'imagenet', include_top = False, input_shape = (224, 224, 3))
vgg_extractor.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```



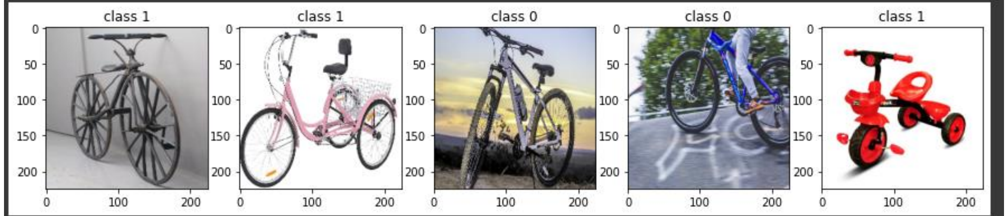
Training

2. Input the data which were prepared to be 3 classes (0 = mountain bike, 1 = tricycle, 2 = tuk tuk) and split data into train and test sets with test size = 20% as shown below.

```
#Train Test Split
from sklearn.model_selection import train_test_split

test_size = 0.2
x_train, x_test = train_test_split(x, test_size = test_size, random_state = 3)
y_train, y_test = train_test_split(y, test_size = test_size, random_state = 3)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(480, 224, 224, 3)
(480,)
(120, 224, 224, 3)
(120,)
```

```
#visualize the first 5 image
plt.figure(figsize = (15,5))
for i in range(5):
  plt.subplot(150 + 1 + i).set_title(f'class {y_train[i]}')
  plt.imshow(x_train[i])
plt.show()
```



Training

3. Freeze the layers in model by setting the Trainable to False

```python
#Recursively freeze all layers in the model first
vgg_extractor.trainable = False

for i, layer in enumerate(vgg_extractor.layers):
  print(f'Layer {i}: Name = {layer.name}, Trainable = {layer.trainable}')
```

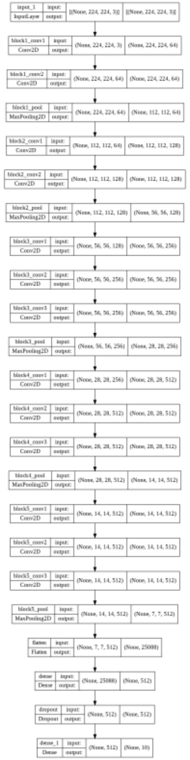
```
Layer 0: Name = input_1, Trainable = False
Layer 1: Name = block1_conv1, Trainable = False
Layer 2: Name = block1_conv2, Trainable = False
Layer 3: Name = block1_pool, Trainable = False
Layer 4: Name = block2_conv1, Trainable = False
Layer 5: Name = block2_conv2, Trainable = False
Layer 6: Name = block2_pool, Trainable = False
Layer 7: Name = block3_conv1, Trainable = False
Layer 8: Name = block3_conv2, Trainable = False
Layer 9: Name = block3_conv3, Trainable = False
Layer 10: Name = block3_pool, Trainable = False
Layer 11: Name = block4_conv1, Trainable = False
Layer 12: Name = block4_conv2, Trainable = False
Layer 13: Name = block4_conv3, Trainable = False
Layer 14: Name = block4_pool, Trainable = False
Layer 15: Name = block5_conv1, Trainable = False
Layer 16: Name = block5_conv2, Trainable = False
Layer 17: Name = block5_conv3, Trainable = False
Layer 18: Name = block5_pool, Trainable = False
```

Training

4. Add new classification head for tuk tuk classification

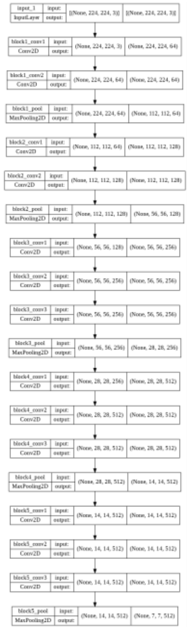
```python
x = vgg_extractor.output

#Add our custom layer(s) to the end of the existing model
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(512, activation = 'relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
new_outputs = tf.keras.layers.Dense(10, activation = 'softmax')(x)

#construct the main model
model = tf.keras.models.Model(inputs = vgg_extractor.inputs, outputs = new_outputs)

model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

flatten (Flatten)            (None, 25088)             0

dense (Dense)                (None, 512)               12845568

dropout (Dropout)            (None, 512)               0

dense_1 (Dense)              (None, 10)                5130

=================================================================
Total params: 27,565,386
Trainable params: 12,850,698
Non-trainable params: 14,714,688
_________________________________________________________________
```

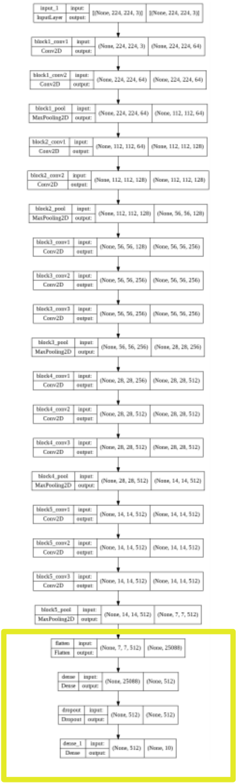


Training

Vgg extractor



Our new model



The classification section
for tuk tuk image classifier

Training

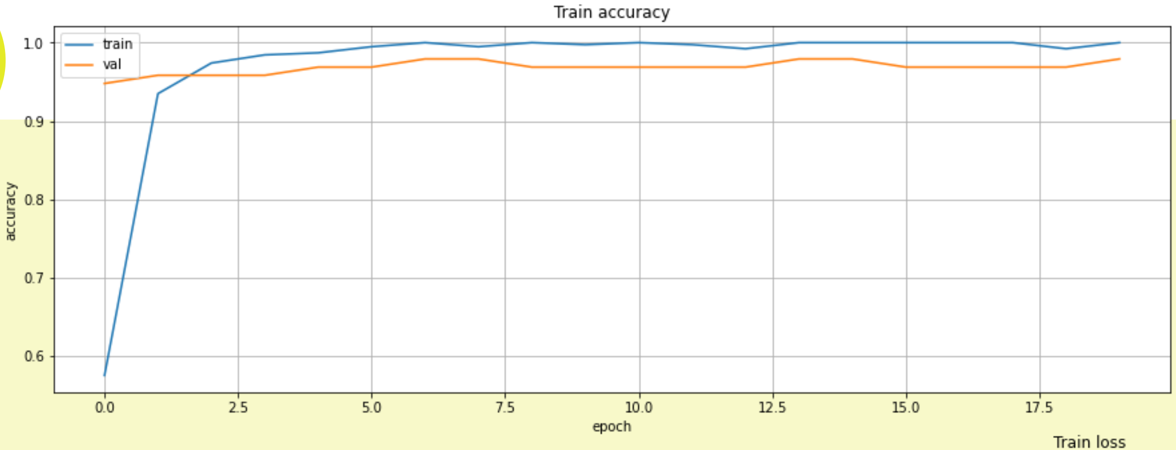
5. Model training by transfer-learning

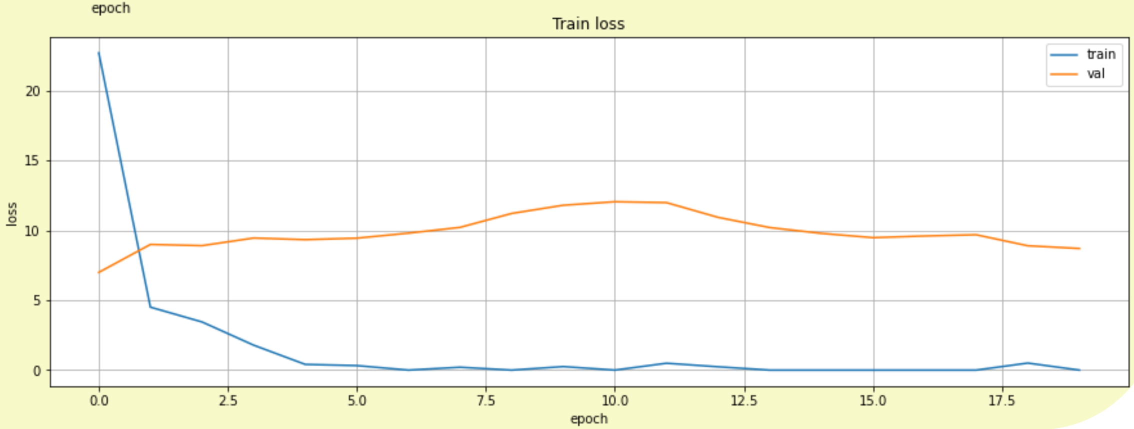
- Loss : Cross Entropy (Sparse Categorical)
- Optimizer : Adam
- Metrics : Accuracy
- Batch Size : 128
- Epoch Number : 20

```
model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['acc'])
history = model.fit(x_train_vgg, y_train, batch_size = 128, epochs = 20, verbose = 1, validation_split = 0.2)
```

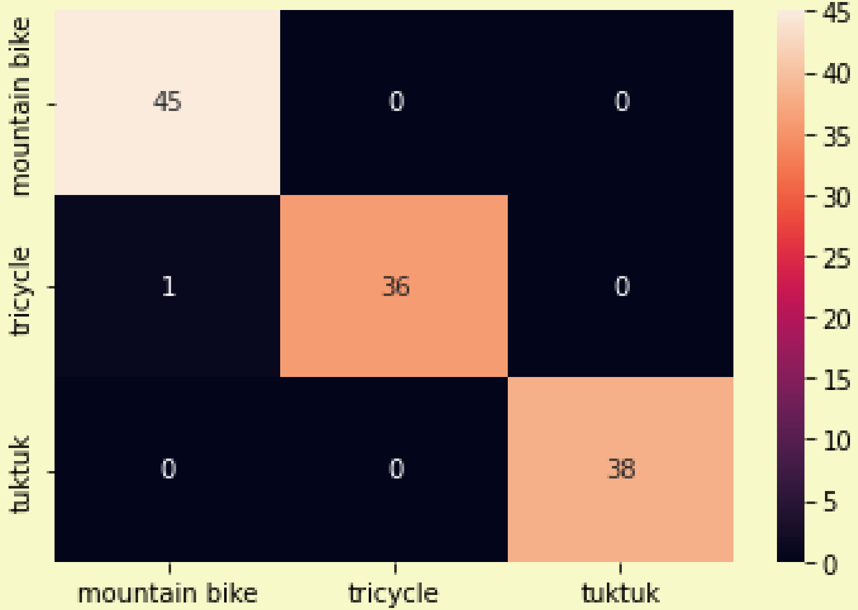
```
Epoch 1/20
3/3 [==============================] - 34s 5s/step - loss: 22.7171 - acc: 0.5755 - val_loss: 6.9981 - val_acc: 0.9479
Epoch 2/20
3/3 [==============================] - 4s 2s/step - loss: 4.5083 - acc: 0.9349 - val_loss: 8.9988 - val_acc: 0.9583
Epoch 3/20
3/3 [==============================] - 4s 2s/step - loss: 3.4484 - acc: 0.9740 - val_loss: 8.9134 - val_acc: 0.9583
Epoch 4/20
3/3 [==============================] - 4s 2s/step - loss: 1.7867 - acc: 0.9844 - val_loss: 9.4579 - val_acc: 0.9583
Epoch 5/20
3/3 [==============================] - 4s 2s/step - loss: 0.4076 - acc: 0.9870 - val_loss: 9.3418 - val_acc: 0.9688
Epoch 6/20
3/3 [==============================] - 4s 2s/step - loss: 0.3216 - acc: 0.9948 - val_loss: 9.4479 - val_acc: 0.9688
Epoch 7/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 9.8066 - val_acc: 0.9792
Epoch 8/20
3/3 [==============================] - 4s 2s/step - loss: 0.2044 - acc: 0.9948 - val_loss: 10.2192 - val_acc: 0.9792
Epoch 9/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 11.2178 - val_acc: 0.9688
Epoch 10/20
3/3 [==============================] - 4s 2s/step - loss: 0.2509 - acc: 0.9974 - val_loss: 11.7979 - val_acc: 0.9688
Epoch 11/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 12.0260 - val_acc: 0.9688
Epoch 12/20
3/3 [==============================] - 4s 2s/step - loss: 0.4837 - acc: 0.9974 - val_loss: 11.9456 - val_acc: 0.9688
Epoch 13/20
3/3 [==============================] - 4s 2s/step - loss: 0.2291 - acc: 0.9948 - val_loss: 11.0191 - val_acc: 0.9688
Epoch 14/20
3/3 [==============================] - 4s 2s/step - loss: 1.5522e-09 - acc: 1.0000 - val_loss: 10.3491 - val_acc: 0.9792
Epoch 15/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 10.0057 - val_acc: 0.9792
Epoch 16/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 9.7385 - val_acc: 0.9792
Epoch 17/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 9.5249 - val_acc: 0.9792
Epoch 18/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 9.3796 - val_acc: 0.9688
Epoch 19/20
3/3 [==============================] - 4s 2s/step - loss: 0.3560 - acc: 0.9974 - val_loss: 9.3767 - val_acc: 0.9688
Epoch 20/20
3/3 [==============================] - 4s 2s/step - loss: 0.0000e+00 - acc: 1.0000 - val_loss: 9.3413 - val_acc: 0.9688
```

VGG-16 Model


Train accuracy
train
val
accuracy
1.0
0.9
0.8
0.7
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

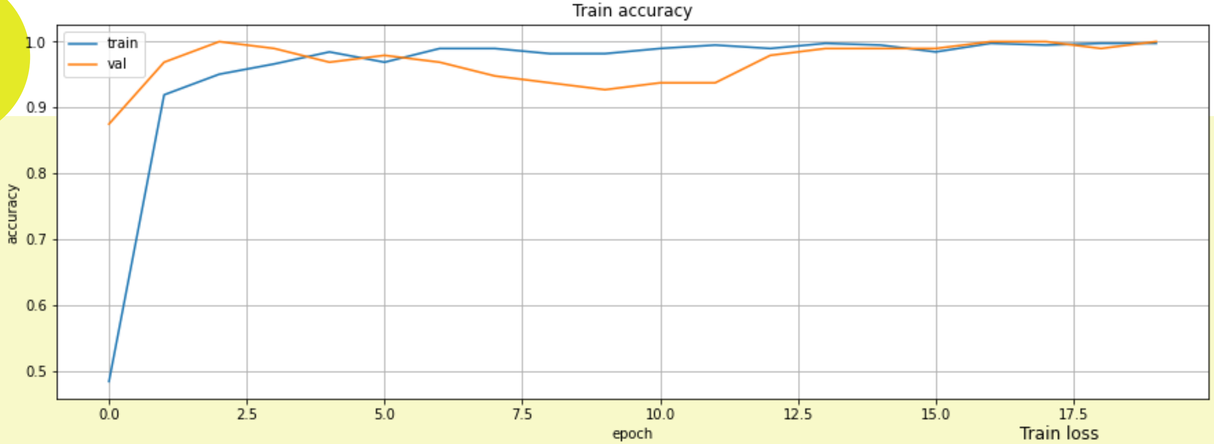

Train loss
train
val
loss
20
15
10
5
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

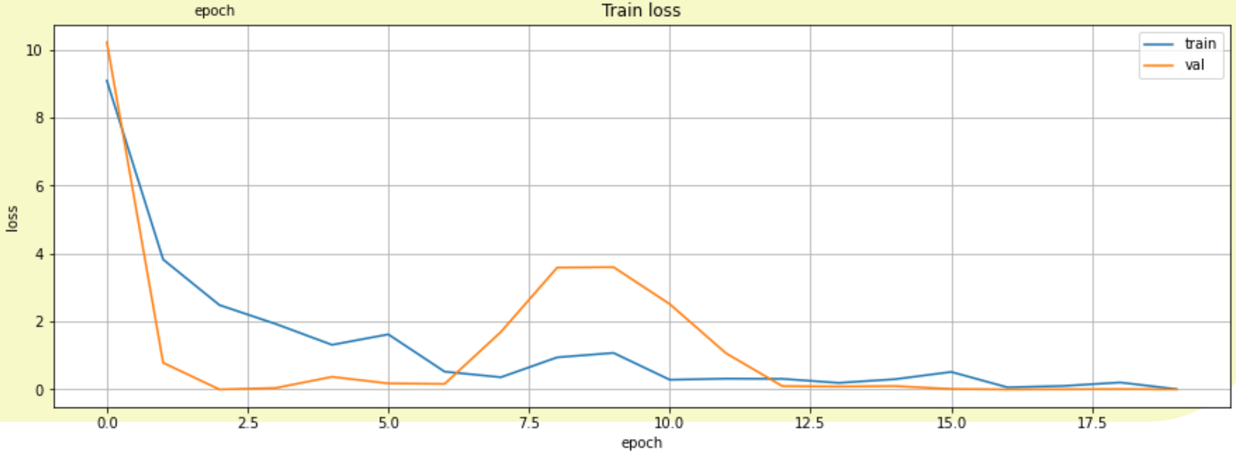
VGG-16 Model



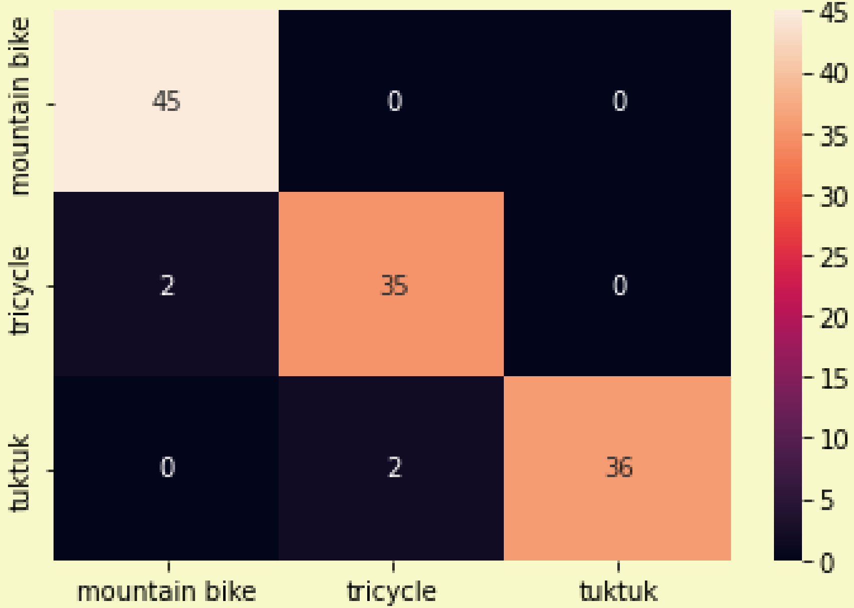
mountain bike
tricycle
tuktuk
45
0
0
1
36
0
0
0
38
mountain bike
tricycle
tuktuk
45
40
35
30
25
20
15
10
5
0

EfficientNetB7 Model

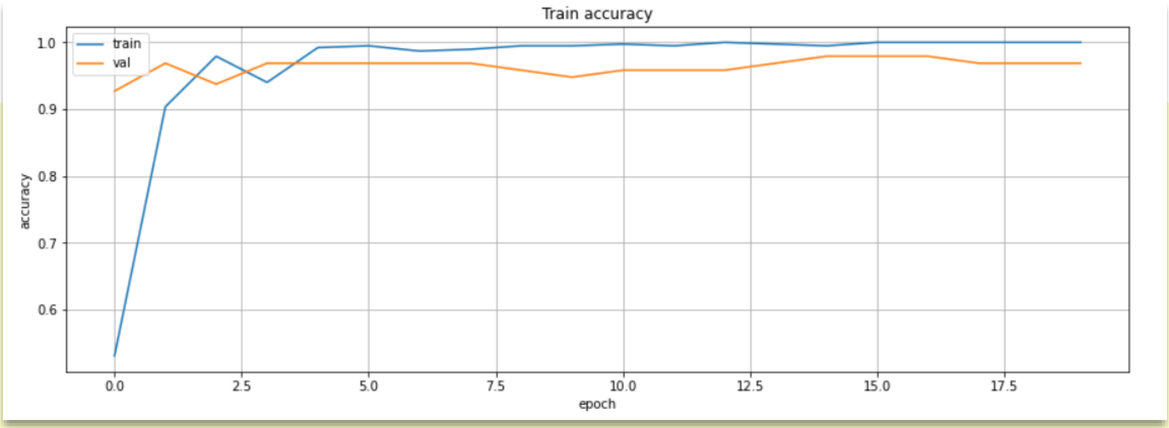


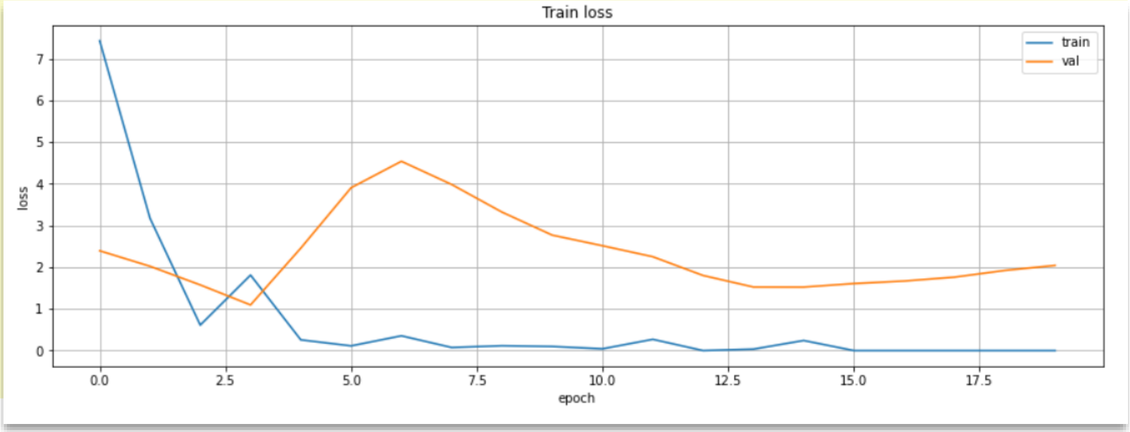


EfficientNetB7 Model

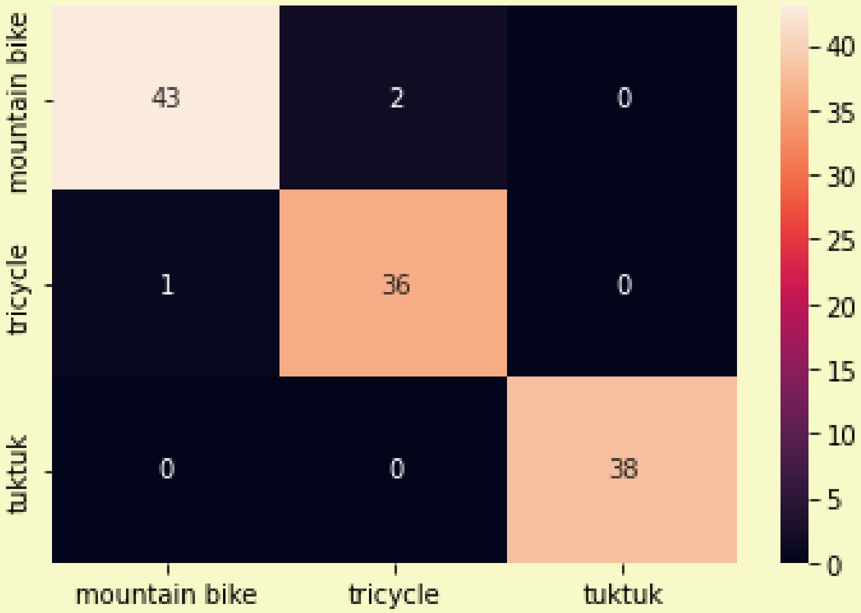


MobileNetV2 Model

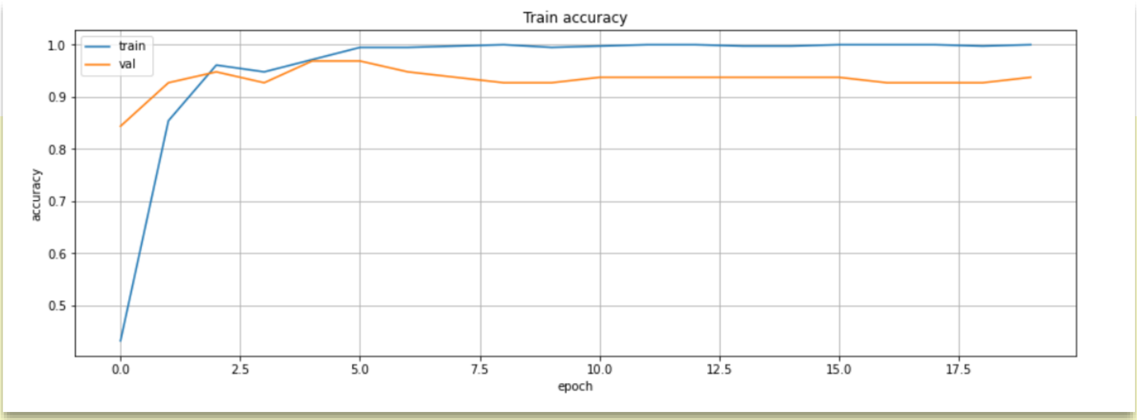


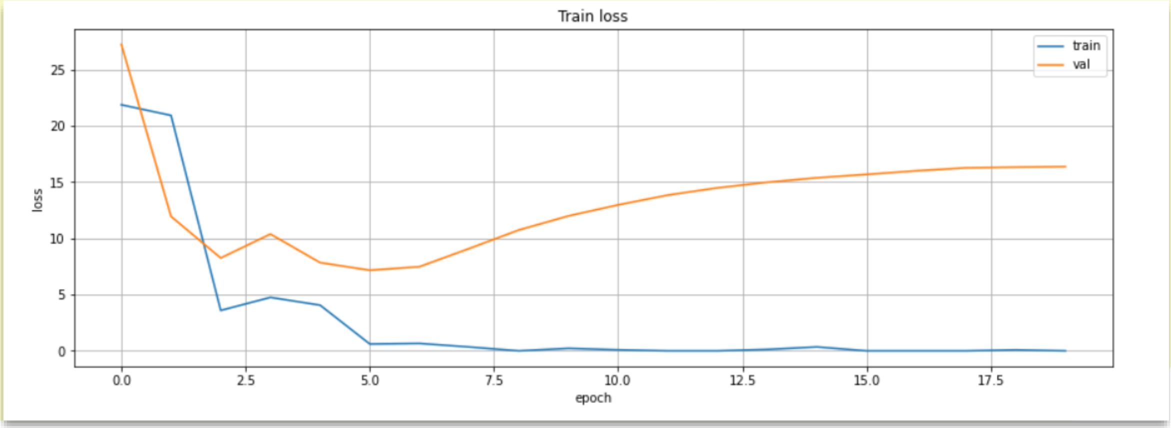


MobileNetV2 Model

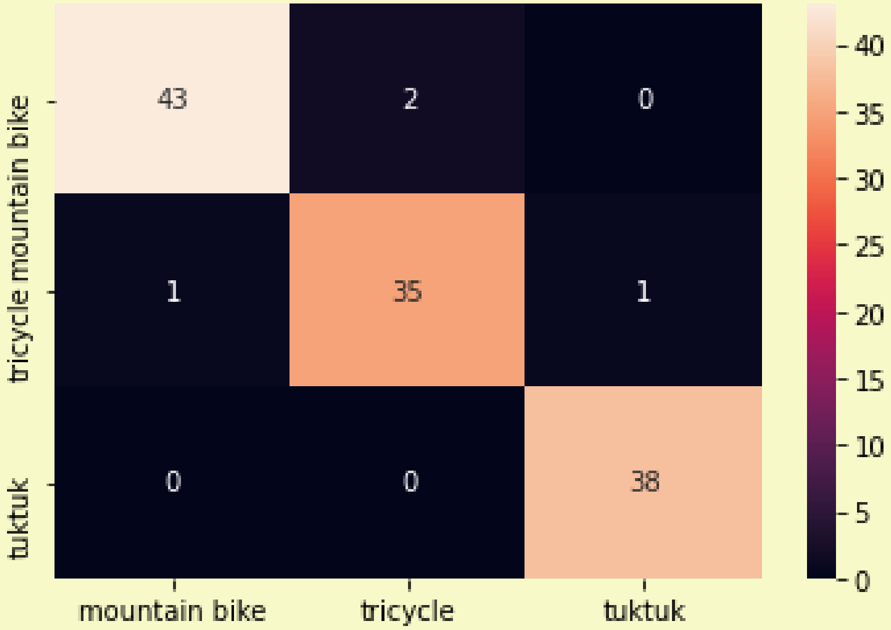


ResNet50 Model

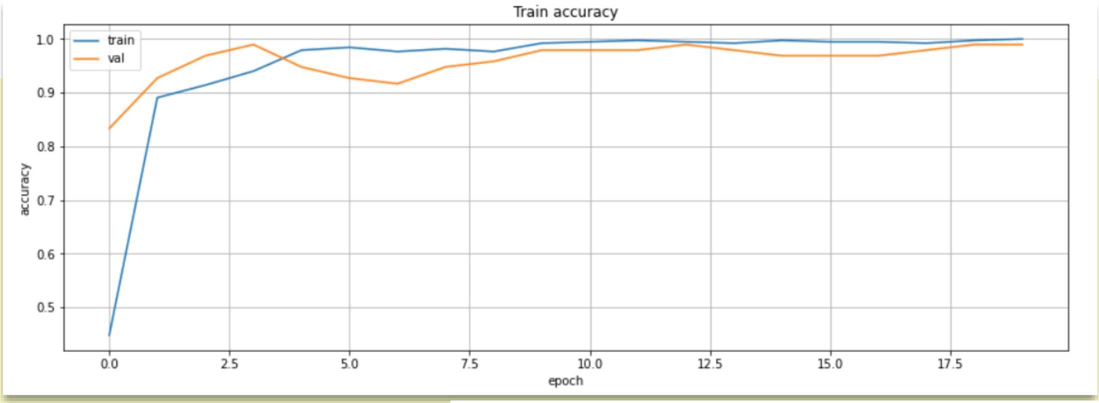


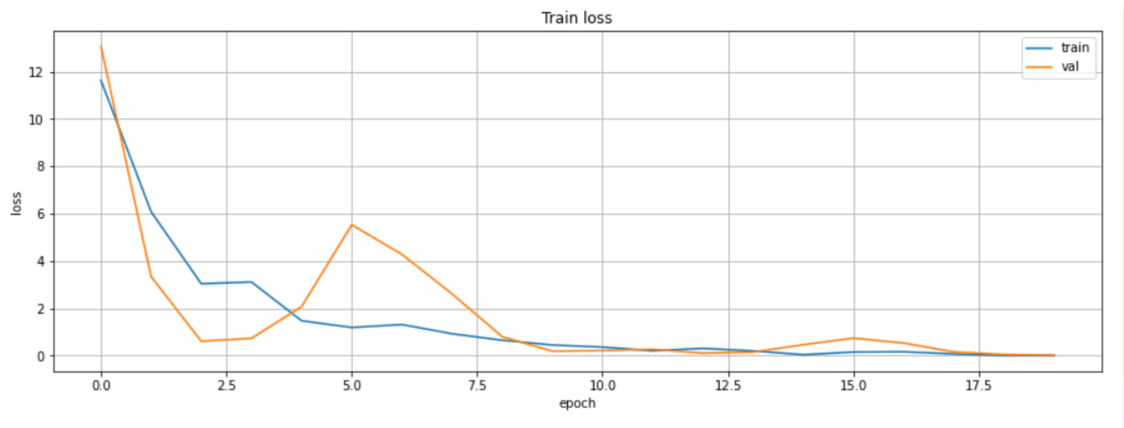


ResNet50 Model

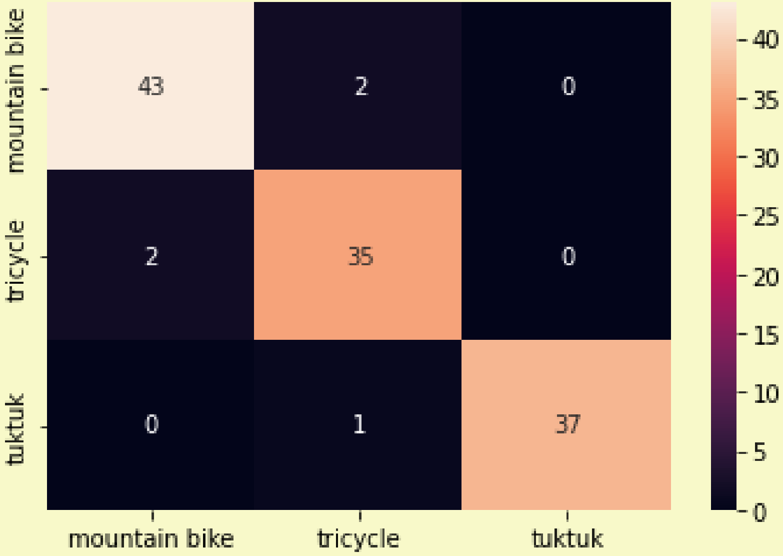


InceptionResNetV2 Model





InceptionResNetV2 Model



Model Evaluation on Test Set

Models	Accuracy	Loss
VGG-16	0.992	2.265
EfficientNetB7	0.967	2.894
MobileNetV2	0.975	0.436
ResNet50	0.967	2.737
InceptionResNetV2	0.958	0.952

Conclusion

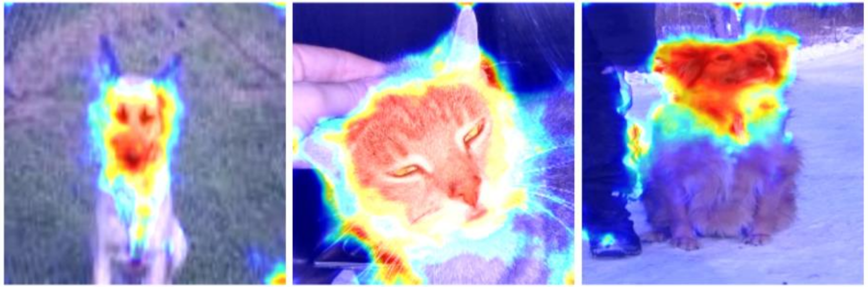
According to the experiment, Google was a source for our image scraping and collecting. As the result of five CNN model studies, there were accuracy between 0.958-0.992 and loss between 0.436-2.737. However, the VGG-16 model was shown the best accuracy (0.992) at a loss of 2.265. Thus, we suggest applying the VGG-16 model to classify smaller vehicle bikes for pedestrian-friendly streets

Gradient-weighted Class Activation Mapping (Grad-CAM)

- *Grad-CAM uses the gradients of target flowing into the final convolutional layer to produce a map highlighting regions in the image for predicting the concept* (Ref: Understand your Algorithm with Grad-CAM | by Daniel Reiff | Towards Data Science)

- Applied Grad-CAM to our modified VGG16 model to study which image area the model focus on to classify the mountain bike, tricycle, and tuk-tuk

- The higher heatmap value (colored red in 'jet' colormap) shows the important area for prediction probability

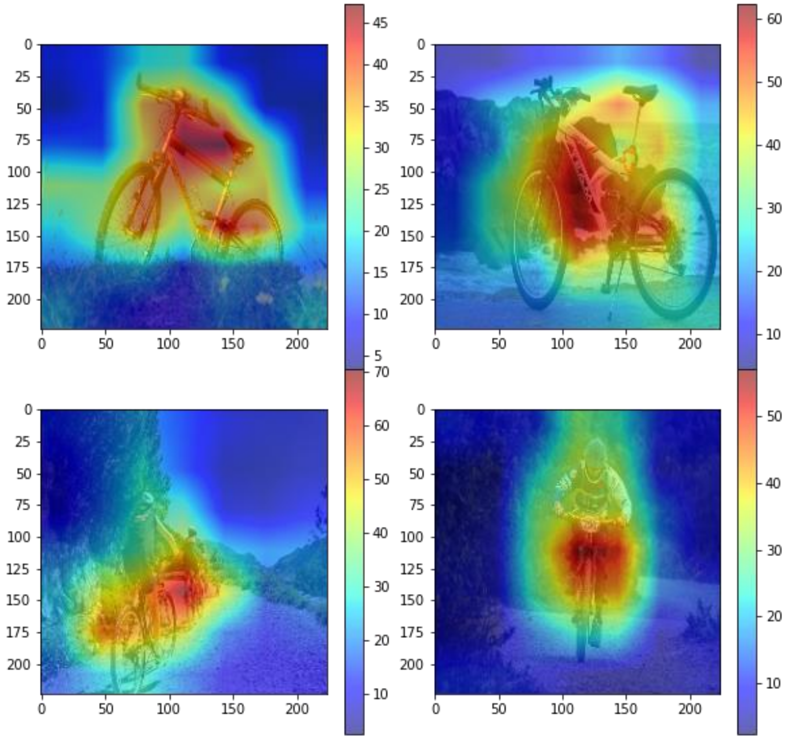


Grad-CAM applied on dog and cat image
(Ref: Understand your Algorithm with Grad-CAM | by Daniel Reiff | Towards Data Science)

Grad-CAM for Mountain Bike

Finding for **Mountain Bike**

- The model seems to identify the Mountain Bike by looking at the center core part of the mountain bike

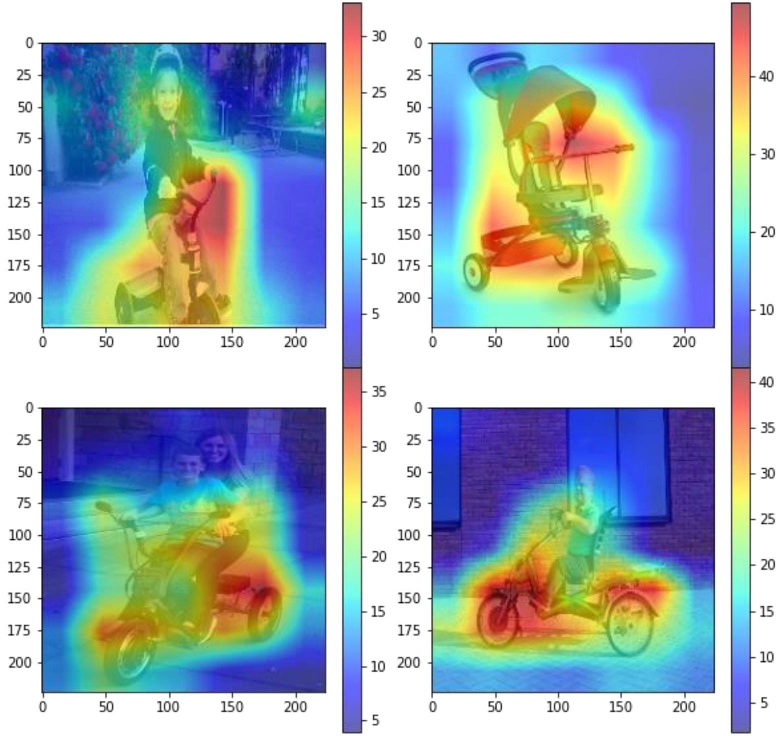


Grad-CAM applied on mountain bike image

Grad-CAM for Tricycle

Finding for **Tricycle**

- The model seems to identify the Tricycle by looking at
- ❑ the wheel
- ❑ the handle

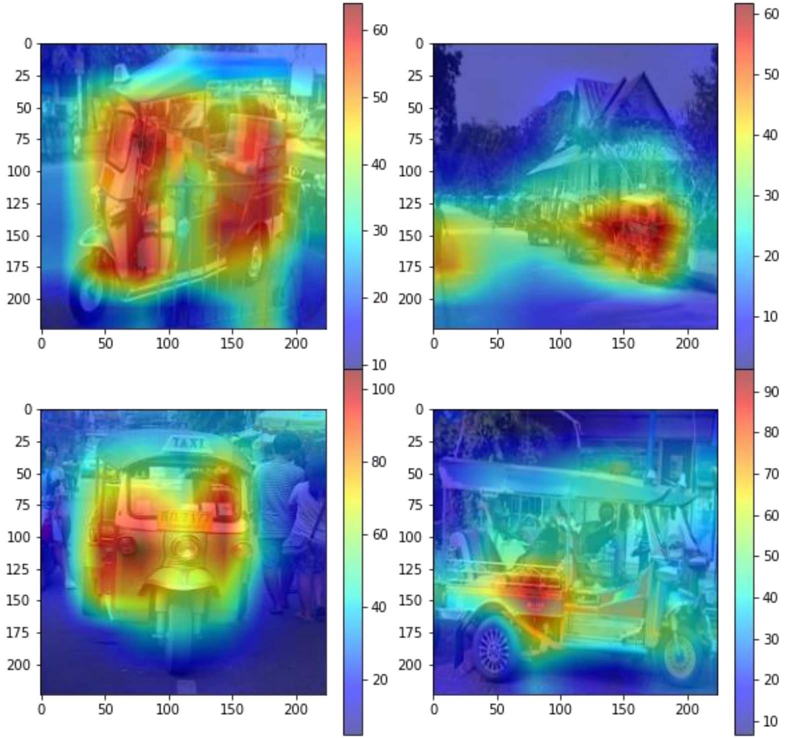


Grad-CAM applied on tricycle image

Grad-CAM for Tuk Tuk

Finding for **Tuk Tuk**

- The model seems to identify the Tuk-Tuk by looking at overall unique characteristics of tuk-tuk such as
- ❑ the front lights
- ❑ the side
- ❑ the front windshield



Grad-CAM applied on tuk tuk image

Gradient-weighted Class Activation Mapping (Grad-CAM)

Grad-CAM Conclusion

- Because the differences between the each class are quite obvious
- Applying Grad-CAM to each class shows that the model classifies each class by looking at the class's outstanding features

Grad-CAM Code

```python
def alter_model_for_GradCAM(model, last_conv_layer_name):
    last_conv_output = model.get_layer(last_conv_layer_name).output

    old_weights = [x.numpy() for x in model.layers[-1].weights]
    new_config = model.layers[-1].get_config()
    new_config['activation'] = tf.keras.activations.linear
    new_config['name'] = 'prediction_linear'
    out_linear = tf.keras.layers.Dense(**new_config)(model.layers[-2].output)
    out_softmax = tf.keras.activations.softmax(out_linear)

    new_model = tf.keras.Model(inputs=model.inputs, outputs=[out_softmax, out_linear, last_conv_output])
    new_model.layers[-2].set_weights(old_weights)

    return new_model

def my_CNN_GradCAM(model, in_img, class_index):
    in_img = tf.cast(in_img, tf.float32)

    with tf.GradientTape() as tape:
        tape.watch(in_img)
        y_softmax, y_linear, last_conv_activation = model(in_img)
        one_class_score = y_linear[..., class_index]
    gradient = tape.gradient(one_class_score, last_conv_activation)

    gradient = gradient.numpy().squeeze(axis=0)
    alpha = np.mean(gradient, axis=(0,1))

    last_conv_activation = last_conv_activation.numpy().squeeze(axis=0)
    heatmap = np.dot(last_conv_activation, alpha)

    heatmap = np.maximum(0, heatmap)

    return heatmap

new_model = alter_model_for_GradCAM(model, 'block5_pool')
new_model.summary()
```

```python
# SHOW 4X4 GRAD-CAM IMAGES FROM SELECTED CLASS

# Input #########
selected_class = 2 #class index {0:'mtbike', 1:'tricycle', 2:'tuktuk'}
#################

import random
import cv2

predict_encode = {0:'mtbike', 1:'tricycle', 2:'tuktuk'}
rand_idx = []

if selected_class == 0:
    xxx = x0
    yyy = y0
elif selected_class == 1:
    xxx = x1
    yyy = y1
elif selected_class == 2:
    xxx = x2
    yyy = y2
else:
    print('Wrong Class Provided')

for i in range(4): #4 images to show
    rand_idx.append(random.randint(0,len(xxx)))

fig = plt.figure(figsize=(10,10))

for i in range(len(rand_idx)):
    print(rand_idx[i])
    x_input = xxx[rand_idx[i]][np.newaxis, ...]
    y_pred = np.argmax(model.predict(x_input))
    print(f'Predicted Class: {predict_encode[y_pred]}')
    print(f'Actual Class: {predict_encode[yyy[rand_idx[i]]]}')

    heatmap = my_CNN_GradCAM(new_model, x_input, y_pred)
    img = xxx[rand_idx[i]]

    ax = fig.add_subplot(2,2,i+1)
    ax.imshow(img)
    alpha = 0.6
    im = ax.imshow(cv2.resize(heatmap, img.shape[:2]), cmap='jet', alpha=alpha)
    plt.colorbar(im, ax=ax)

plt.subplots_adjust(wspace=0.1, hspace=0)
plt.show()
```

Discussion

- The results are quite good with 99.2% accuracy for the VGG-16 model, and all 5 models we selected have more than 95% accuracy. For the next step, we may consider running all models with the same CPU and GPU condition to compare the processing time for each model which will be another key criterion in case of a real implementation.
- Another improvement is to add more classes for small vehicle image classification such as pedestrian, trolley, food truck, etc. To be more beneficial and can be used for more purposes.

References

Python Version: 3.8.5

Python Library
- Matplotlib 3.5.1
- Numpy 1.22.0
- OpenCV 4.5.5
- Tensorflow 2.7.0

Source Code
- Transfer Learning: BADS7604 by Asst. Prof. Thitirat Sribiborbornratanakul (CNN2_ex3)
- Grad-CAM: BADS7604 by Asst. Prof. Thitirat Sribiborbornratanakul (CNN3_ex2)

Datasets
- Please look at the Citing page

Citing

- For those who want to use the dataset images,
- ❑ For **mountain bike** dataset, please read and use the image under the **pixabay.com policy**
- ❑ For **tricycle** dataset, referenced URL are provided in **image_references_tricycle.txt**
- ❑ For **tuk-tuk** dataset, cite them as a format provided in **image_references_tuktuk.txt**

- For those who want to use any other image but not the dataset images, please reference the image in **bibtex** format

Team Members

6310412018	Pongsarat Chootai	**(25%)** Develop image scrapping coding for team, Image Scraping (Tricycle), Develop original coding for model transfer-learning for team and evaluate VGG-model, Result Discussion for improvement
6310412021	Theethut Narksenee	**(25%)** Image Scraping (Mountain bike), Image preparation, Transfer Learning of EfficientNetB7 model and InceptionResNetV2 model
6310412024	Saranchai Angkawinijwong	**(25%)** Highlights, Image Scraping (Tuk Tuk), VGG Transfer Learning, Grad-CAM, Miscellaneous(References, Citing, End Credit)
6420400001	Krittipat Chuenphitthayavut	**(25%)** Introduction composition, Image scaping, Experimental variation on resnet50 and MobileNetV2

End Credit

This project is a part of **Deep Learning** course

Course code: **BADS7604**

Syllabus: Business Analytics and Data Science (**BADS**)

Institute: National Institute of Development Administration (**NIDA**)

Semester: **2/2021**

Thank you for your attention