Low-altitude small-sized object detection using lightweight feature-enhanced convolutional neural network - Revisited

Logan Schraeder
Graduate Student, MS in Data Science, Wichita State University
CS 797O Project 2

Abstract

Based on the work of Tao et al. (1) in the 2021 work of the same name, we revisit the theory, real-world application, and proposed next steps for Project 2. Tao and team's work improved upon the general processing time and the accuracy of feature extraction and labeling of small, low-flying objects. The major benchmark considered was YOLOv4-tiny, run on MS COCO and VOT-RT2019 datasets. LSL-Net, Tao's CNN, reportedly achieved a mAP of 90.97% at a frame rate of 147 FPS on relatively standard hardware. Following Tao's work, we attempt to recreate this network and the team's results.

Abbreviations, Acronyms, and Definitions

Item	Definition
ADM	Accurate Detection Module
CNN	Convolutional Neural Network
CSP	Cross-Stage Partial
EFM	Enhanced Feature processing Module
FPS	Frames Per Second
IAW	In Accordance With
LSM	Lightweight and Stable feature extraction Module
mAP	Mean Average Precision
SWaP	Size, Weight, and Power

Network Creation

This implementation of LSL-Net was created in PyCharm and broken into two primary .py files. The first contains the "main" running script for the entire network and includes input image preprocessing while the second file contains all of the method/functions for each sub-network within LSL-Net. While different methods and modules from core libraries were used in each file, the core libraries were relatively consistent and included Tensorflow (and Keras), Tensorflow Hub, Numpy, and os. Unfortunately programming the entirety of LSL-Net was both time-consuming and buggy; therefore a compromise was made where all components except for the final image recognition module ("Accurate Detection Module" in Tao (1)). In lieu of the ADM, EfficientDet[1] (2) was used to run object detection and classification tasks on an open-source drone dataset (3) to show an end product comparable to what LSL-Net should have produced in it's full implementation.

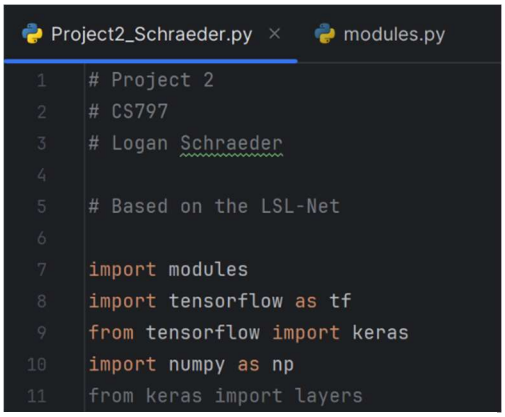

Figure 1. PyCharm. Screenshot of the Python files and libraries used.

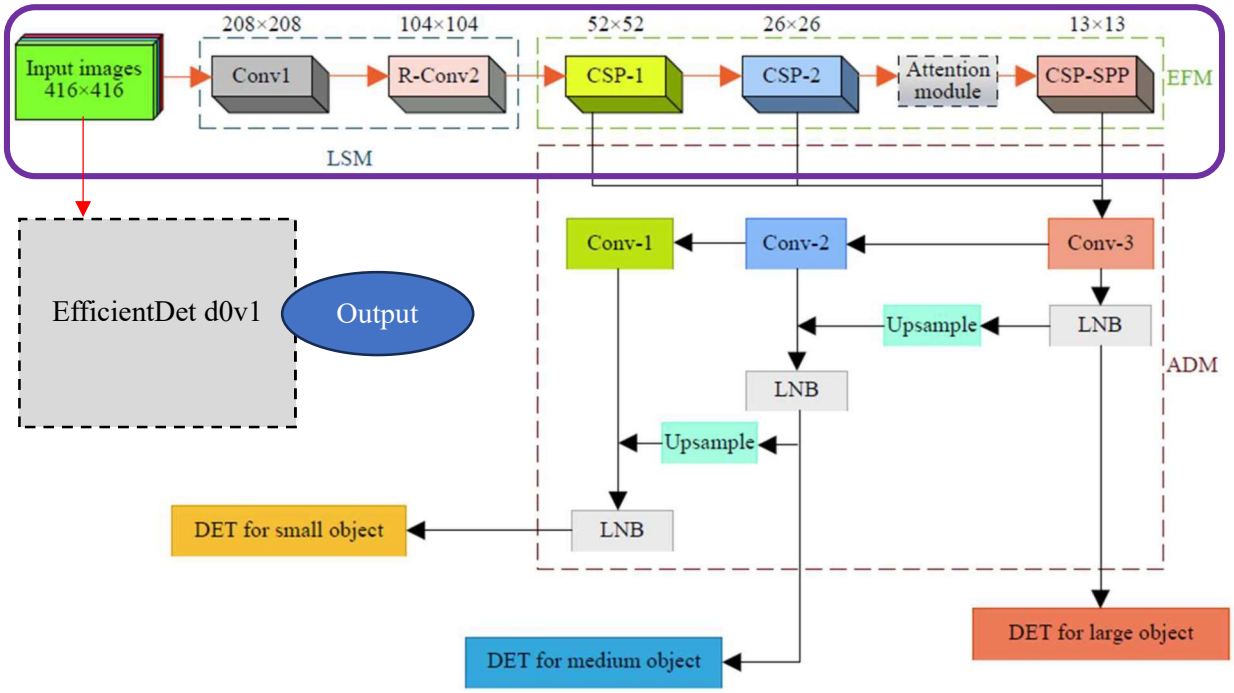

Figure 2. LSL-Net Architecture. Overview of the LSL-Net (taken from (1)). The section at the top outlined in purple (containing the LSM and EFM) is what was coded in this project. The red arrow pointing to EfficientDet d0v1 is what was used as a conceptual stand-in for the classification task the ADM would have finished.

[1] EfficientDet model d0 version v1 to be fully precise.

1. Sub-Nets

Lightweight and Stable feature extraction Module

```
Model: "LSM"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_1 (InputLayer)        [(None, 416, 416, 3)]     0         []

 conv2d (Conv2D)             (None, 207, 207, 32)      896       ['input_1[0][0]',
                                                                  'input_1[0][0]']

 conv2d_1 (Conv2D)           (None, 207, 207, 16)      528       ['conv2d[0][0]']

 conv2d_2 (Conv2D)           (None, 103, 103, 16)      2320      ['conv2d_1[0][0]']

 max_pooling2d (MaxPooling2  (None, 103, 103, 32)      0         ['conv2d[1][0]']
 D)

 conv2d_3 (Conv2D)           (None, 103, 103, 32)      544       ['conv2d_2[0][0]']

 conv2d_4 (Conv2D)           (None, 103, 103, 32)      1056      ['max_pooling2d[0][0]']

 concatenate (Concatenate)   (None, 103, 103, 64)      0         ['conv2d_3[0][0]',
                                                                  'conv2d_4[0][0]']

=================================================================================================
Total params: 5344 (20.88 KB)
Trainable params: 5344 (20.88 KB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 3. LSM Summary. Python console output summary of the LSM.

The Lightweight and Stable feature extraction Module (LSM) was created using an initial convolutional layer (after the input tensor layer) and branching into two paths. The first path contained three convolutional layers and the second a max pooling layer followed by a single convolutional layer. Finally the two branches were brought together via concatenation and a Keras model returned as the method's output. If an input was provided, the LSM then ran the input tensor through the model and returned a feature map ready for use with the EFM.

The most difficult part in programming the LSM was the branching of the module. Theoretically or conceptually it was easy to think of running two parallel CNN tasks, but in practice it took much more digging into Keras' capabilities.

Enhanced Feature processing Module

LSL-Net's Enhanced Feature processing Module (EFM) was even more complex as it contained two sub-sub-nets that performed feature extraction and had to be called around an attention layer. First, the EFM ingested the feature map from the LSM before feeding the feature map into a sub-sub-net called the "Cross-Stage Partial" module twice sequentially. The first CSP iteration – CSP-1 – produced a 52x52x128 feature map, while CSP-2 then produced a 26x26x256 feature map. This mid-stage feature map was then checked for tensor size agreement with what the following attention layer was expecting; if the two disagreed, the feature map was scaled up via zero padding. This allowed the feature map tensor to be passed between modules without losing any of the numerical features embedded.

Next, the attention layer's output moved into the final sub-sub-net called "CSP-SPP". CSP-SPP was designed to "greatly enhance the feature extraction ability [of the model]" (1). This sub-sub-net is made of several sub-branched convolutional layers and paths before finally being concatenated and passing the final feature map out of the EFM. The EFM's output feature map is therefore a 13x13x256 tensor that contains numerical representations of features within the image and starting points for object classification bounding boxes.

Training Data and Outputs

In this exercise training data ended up not being used given that the entire network was not implemented, and therefore training would be pointless. However the training dataset that would have been used would have been a variation of the MS COCO dataset, very likely downsized to a more manageable storage and sample size[2] like the MATLAB and Python "COCO Demo" dataset (4) (5). This dataset would have been split into a train, test, and validation split either manually or using a tool like SciKit Learn's Train-Test split module (6) to train the model on about 70% of the dataset, use 20% for true inference testing, and 10% for validation/hyperparameter tuning.

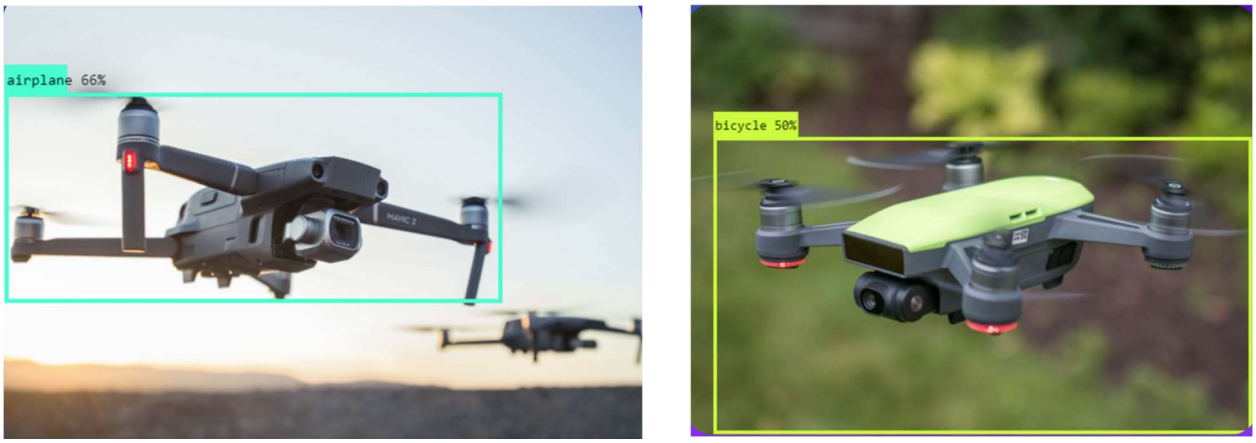

Figure 4. EfficientDet Object Detection Outputs. Drone images from (3) fed into EfficientDet – trained on MS COCO – for object detection. As evident, EfficientDet both generally misclassified the drones and had relatively low confidence; a problem that would have been overcome by a more task-representative dataset and/or transfer learning for the use case.

This would give three outputs. The first being the analyzed image with bounding boxes and object classes attached. The second and third outputs would be the model accuracy and loss graphs from training, which both indicate the model's accuracy and error, and if the model is overfitting, not converging, or encountering other learning problems. As seen in Figure 4, the ability of EfficientDet trained on MS COCO without fine-tuning is able leaves quite a bit to be desired in terms of classification and confidence.

[2] The full COCO dataset includes over 330,000 images and 1.5 million object instances, equating to 42.7GB of space (7).

Implementation Obstacles

The main problem with implementing the full LSL-Net was time and general complexity. Given more time and/or lesser outside responsibilities, LSL-Net could have likely been completed in its entirety. LSL-Net was extremely complex and taught me how to do many things I hadn't encountered with neural networks before – multi-branched networks with attention mechanisms, sub-networks within networks (not just pre-processing networks or modules), and handling padding or tensor-size issues between modules. So while the complexity was certainly there, the experience was by far a net positive.

References

1. *Low-altitude small-sized object detection using lightweight feature-enhanced convolutional neural network.* **Ye, Tao, et al.** 4, Beijing : Journal of Systems Engineering and Electronics, 1021, Vol. 21.

2. **Google (Tensorflow).** efficientdet. *Kaggle.* [Online] Google. [Cited: December 10, 2023.] https://www.kaggle.com/models/tensorflow/efficientdet.

3. **Ozel, Mehdi.** Drone Dataset (UAV). *Kaggle.* [Online] Google, 2019. [Cited: December 11, 2023.] https://www.kaggle.com/datasets/dasmehdixtr/drone-dataset-uav.

4. **Microsoft.** COCO API. *Github.* [Online] [Cited: December 11, 2023.] https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoDemo.ipynb.

5. —. Dataset. *Common Objects in Context.* [Online] 2021. [Cited: November 29, 2023.] https://cocodataset.org/#overview.

6. **SciKit.** sklearn.model_selection.train_test_split. *SciKit Learn.* [Online] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

7. **Mihajlovic, Ilija.** Everything You Ever Wanted to Know About Computer Vision. *Towards Data Science.* [Online] Medium, April 25, 2019. [Cited: November 04, 2023.] https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e.

8. **IBM.** What is computer vision? *IBM Think.* [Online] [Cited: November 04, 2023.] https://www.ibm.com/topics/computer-vision.

9. **Redmon, Joseph, et al.** *You Only Look Once: Unified, Real-Time Object Detection.* 2016. arXiv:1506.026405v5.

Appendix A - Main .py File

File - C:\Users\lbsch\OneDrive\School\CS 797O - NN and DL\Project\Project2_Schraeder.py

```
 1 # Project 2
 2 # CS797
 3 # Logan Schraeder
 4
 5 # Based on the LSL-Net
 6
 7 import modules
 8 import tensorflow as tf
 9 from tensorflow import keras
10 import numpy as np
11 from keras import layers
12 from keras.models import Sequential
13 import os
14 import pycocotools
15 import graphviz
16 import pydot
17
18 # Image loading and preprocessing
19
20 img_path = os.path.join(os.getcwd(), 'Sample COCO
   Images', 'helicopter.jpg')
21 img_height = 416
22 img_width = 416
23 # Load image and convert to a tf tensor
24 input_img = keras.utils.load_img(img_path,
   target_size=(img_height, img_width))
25 input_tensor = tf.convert_to_tensor(input_img)
26 # Add batch dimension to tensor to be compatible with
    Keras CNN input shapes
27 input_tensor = np.expand_dims(input_tensor, axis=0)
28
29
30 lsm = modules.lsm()
31 csp1 = modules.csp()
32 csp2 = modules.csp()
33 cspspp = modules.csp_spp()
34
35 lsm_out = lsm(input_tensor)
36 efm_out = modules.efm(lsm_out)
37 print(efm_out)
38 # Input into LNB and Accurate Detection Module goes
```

```
38 here
39 # Outputs small/med/large object detection and
   classification
```

Appendix B - LSL-Net Modules .py File

File - C:\Users\lbsch\OneDrive\School\CS 797O - NN and DL\Project\modules.py

```python
import tensorflow as tf
from tensorflow import keras
from keras import layers

def spp_net(img_in):
    # SPP Net within CSP-SPP
    # Input: 26x26x32; Output: 26x26x128
    # Returns the SPP-Net model ready for use
    if img_in is None:
        inputs = keras.Input(shape=(26, 26, 32))
    else:
        inputs = img_in
    # 3-headed feature extraction max pool layers
    branch1 = layers.MaxPooling2D(pool_size=(5, 5), strides=(1, 1))(inputs)
    branch2 = layers.MaxPooling2D(pool_size=(9, 9), strides=(1, 1))(inputs)
    branch3 = layers.MaxPooling2D(pool_size=(13, 13), strides=(1, 1))(inputs)

    # Check and resize each branch to match expected input size

    if img_in.shape[1:2] != 26:
        img_in = tf.image.resize_with_pad(img_in, 26, 26)
    if branch1.shape[1:2] != 26:
        branch1 = tf.image.resize_with_pad(branch1, 26, 26)
    if branch2.shape[1:2] != 26:
        branch2 = tf.image.resize_with_pad(branch2, 26, 26)
    if branch3.shape[1:2] != 26:
        branch3 = tf.image.resize_with_pad(branch3, 26, 26)

    # Bring the branches together
    output = layers.concatenate([img_in, branch1, branch2, branch3])
    spp_net_model = keras.Model(inputs=inputs, outputs=output, name="SPP-Net")
```

```
32      spp_net_model.summary()
33
34      return spp_net_model
35
36  def lsm():
37      # Build LSM model and module
38      # Input: 416x416x3 image; Output: 104x104x3
    feature map
39      # Returns the LSM model ready for use
40      # Keras non-sequential model API
41      # FYI the model can be shown as a graphic plot
    using:
42      # keras.utils.plot_model(model, "<something.png
    >")
43      inputs = keras.Input(shape=(416, 416, 3))
44      conv = layers.Conv2D(32, kernel_size=(3, 3),
    strides=(2, 2))
45      conv_b1 = conv(inputs)
46      # Input layer into first convolutional layer. Now
     branch:
47      # Branch 1
48      conv_b1 = layers.Conv2D(16, kernel_size=(1, 1),
    strides=(1, 1))(conv_b1)
49      conv_b1 = layers.Conv2D(16, kernel_size=(3, 3),
    strides=(2, 2))(conv_b1)
50      output_b1 = layers.Conv2D(32, kernel_size=(1, 1
    ), strides=(1, 1))(conv_b1)
51
52      # Branch 2
53      conv_b2 = conv(inputs)
54      conv_b2 = layers.MaxPooling2D((2, 2), strides=(2
    , 2))(conv_b2)
55      output_b2 = layers.Conv2D(32, kernel_size=(1, 1
    ), strides=(1, 1))(conv_b2)
56
57      # Concacenation
58      output = layers.concatenate([output_b1, output_b2
    ])
59
60      lsm_model = keras.Model(inputs=inputs, outputs=
    output, name="LSM")
```

```
61      lsm_model.summary()
62
63
64      # tf.keras.utils.plot_model(model, "lslnet_trial
    .png")
65      # TODO: Not sure why plot_model doesn't want to
    work. Try in Colab
66      # TODO: Layer output shapes are 1 off in both
    dimensions; problem?
67
68      return lsm_model
69
70  def csp():
71      # Build CSP model and module
72      # Input: 104x104x64 feature map from LSM
73      # Output: Reduced feature maps
74      # Returns the CSP model ready for use
75      inputs = keras.Input(shape=(104, 104, 64),
    batch_size=1)
76
77      csp_filters = 32
78      # Initial convolutional layer
79      conv = layers.Conv2D(csp_filters, kernel_size=(3
    , 3), strides=(1, 1))
80      conv1 = conv(inputs)
81
82      # Additional feature extraction branch(es)
83      conv_b1 = layers.Conv2D(csp_filters, kernel_size
    =(3, 3), strides=(1, 1))(conv1)
84      conv_b1_1 = layers.Conv2D(csp_filters,
    kernel_size=(3, 3), strides=(1, 1), padding="same")(
    conv_b1)
85      output_b1_1 = layers.concatenate([conv_b1,
    conv_b1_1])
86      output_b1 = layers.Conv2D(csp_filters,
    kernel_size=(1, 1), strides=(1, 1), padding="same")(
    output_b1_1)
87      output_b1 = layers.MaxPooling2D(pool_size=(1, 1
    ), padding="same")(output_b1)
88      output_b1 = tf.image.resize_with_pad(output_b1,
    102, 102)
```

```
 89     # Final concatenation and feature map output
 90     output = layers.concatenate([conv1, output_b1])
 91 
 92     csp_model = keras.Model(inputs=inputs, outputs=
    output, name="CSP")
 93     csp_model.summary()
 94 
 95     return csp_model
 96 
 97 def csp_spp():
 98     # CSP-SPP submodule within the EFM
 99     # Input: Image of shape 26x26x256; Output:
    13x13x256 feature map
100     # Returns the CSP-SPP model ready for use
101     inputs = keras.Input(shape=(26, 26, 256))
102     # First convolution
103     conv = layers.Conv2D(256, kernel_size=(3, 3),
    strides=(1, 1))
104     conv1 = conv(inputs)
105 
106     # Branch B
107     strides = (1, 1)
108     conv_b = layers.Conv2D(128, kernel_size=(3, 3),
    strides=strides)(conv1)
109     conv_b = layers.Conv2D(64, kernel_size=(1, 1),
    strides=strides)(conv_b)
110     # Continues Branch B into B1 (b is used later
    for a recurrent connection)
111     conv_b1 = layers.Conv2D(64, kernel_size=(3, 3),
    strides=strides)(conv_b)
112     conv_b1 = layers.Conv2D(32, kernel_size=(1, 1),
    strides=strides)(conv_b1)
113     conv_b1_spp = spp_net(conv_b1)
114     conv_b1 = conv_b1_spp(conv_b1)
115     conv_b = tf.image.resize_with_pad(conv_b1, 26,
    26)
116     conc1 = layers.concatenate([conv_b, conv_b1])
117     conv_B = layers.Conv2D(256, kernel_size=(3, 3),
    strides=strides)(conc1)
118 
119     # Final concatenation
```

```
120     conc2 = layers.concatenate([conv1, conv_B])
121 
122     # Max pooling layer
123     endpool = layers.MaxPooling2D(pool_size=(2, 2),
    strides=(2, 2))(conc2)
124     output = layers.Conv2D(256, kernel_size=(1, 1),
    strides=strides)(endpool)
125 
126     csp_spp_model = keras.Model(inputs=inputs,
    outputs=output, name="CSP-SPP")
127     csp_spp_model.summary()
128 
129     return csp_spp_model
130 
131 def efm(input_tensor):
132     if input_tensor.shape[1:2] != 104:
133         input_tensor = tf.image.resize_with_pad(
    input_tensor, 104, 104)
134     efm_output = csp()(input_tensor)
135     if efm_output.shape[1:2] != 104:
136         efm_output = tf.image.resize_with_pad(
    efm_output, 104, 104)
137     efm_output = csp()(efm_output)
138     if efm_output.shape[1:2] != 104:
139         efm_output = tf.image.resize_with_pad(
    efm_output, 104, 104)
140     efm_output_attn = keras.layers.Attention()([
    efm_output, input_tensor])
141     efm_output_attn = tf.reshape(efm_output_attn, (-
    1, 26, 26, 256))
142 
143     efm_output = csp_spp()(efm_output_attn)
144 
145     return efm_output
146 
147 
```

Appendix C - Main .py Console Output

File - Project2_Schraeder

```
C:\Users\lbsch\anaconda3\envs\CS898_py310\python.exe "C:\Users\lbsch\OneDrive\School\CS 7970 - NN and DL\Project\Project2_Schraeder.py"
2023-12-12 15:34:22.776427: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "LSM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to

=================================================================
 input_1 (InputLayer)        [(None, 416, 416, 3)]     0         []

 conv2d (Conv2D)             (None, 207, 207, 32)      896       ['input_1[0][0]',
                                                                  'input_1[0][0]']

 conv2d_1 (Conv2D)           (None, 207, 207, 16)      528       ['conv2d[0][0]']

 conv2d_2 (Conv2D)           (None, 103, 103, 16)      2320      ['conv2d_1[0][0]']

 max_pooling2d (MaxPooling2  (None, 103, 103, 32)      0         ['conv2d[1][0]']
 D
)
```

```
18
19
20  conv2d_3 (Conv2D)           (None, 103, 103, 32
   )         544       ['conv2d_2[0][0]']
21
22  conv2d_4 (Conv2D)           (None, 103, 103, 32
   )         1056      ['max_pooling2d[0][0]']
23
24  concatenate (Concatenate)   (None, 103, 103, 64
   )         0         ['conv2d_3[0][0]',
25
                       'conv2d_4[0][0]']
26
27 ==================================================
   ================================================
28 Total params: 5344 (20.88 KB)
29 Trainable params: 5344 (20.88 KB)
30 Non-trainable params: 0 (0.00 Byte)
31 _________________________________________________
   ________________________________________________
32 Model: "CSP"
33 _________________________________________________
   ________________________________________________
34  Layer (type)                Output Shape
                    Param #   Connected to

35 ==================================================
   ================================================
36  input_2 (InputLayer)        [(1, 104, 104, 64
   )]          0         []
37
38  conv2d_5 (Conv2D)           (1, 102, 102, 32
   )            18464     ['input_2[0][0]']
39
40  conv2d_6 (Conv2D)           (1, 100, 100, 32
```

```
 )             9248      ['conv2d_5[0][0]']
 

 conv2d_7 (Conv2D)           (1, 100, 100, 32
)             9248      ['conv2d_6[0][0]']
 

 concatenate_1 (Concatenate  (1, 100, 100, 64
)             0         ['conv2d_6[0][0]',
 )
                  'conv2d_7[0][0]']
 

 conv2d_8 (Conv2D)           (1, 100, 100, 32
)             2080      ['concatenate_1[0][0]']
 

 max_pooling2d_1 (MaxPoolin  (1, 100, 100, 32
)             0         ['conv2d_8[0][0]']
 g2D
)


 

 tf.image.resize_with_pad (  (1, 102, 102, 32
)             0         ['max_pooling2d_1[0][0]']
 TFOpLambda
)


 

 concatenate_2 (Concatenate  (1, 102, 102, 64
)             0         ['conv2d_5[0][0]',
 )
                   'tf.image.resize_with_pad[0][
 
                  0]']
 

 ==============================================
 ==============================================
```

```
Total params: 39040 (152.50 KB)
Trainable params: 39040 (152.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "CSP"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to

=================================================================
 input_3 (InputLayer)        [(1, 104, 104, 64)]          0         []

 conv2d_9 (Conv2D)           (1, 102, 102, 32)            18464     ['input_3[0][0]']

 conv2d_10 (Conv2D)          (1, 100, 100, 32)            9248      ['conv2d_9[0][0]']

 conv2d_11 (Conv2D)          (1, 100, 100, 32)            9248      ['conv2d_10[0][0]']

 concatenate_3 (Concatenate  (1, 100, 100, 64)            0         ['conv2d_10[0][0]',
 )                                                                   'conv2d_11[0][0]']

 conv2d_12 (Conv2D)          (1, 100, 100, 32)            2080      ['concatenate_3[0][0
```

```
79 ]']
80
81  max_pooling2d_2 (MaxPoolin  (1, 100, 100, 32
   )            0         ['conv2d_12[0][0
   ]']
82  g2D
   )
83
84  tf.image.resize_with_pad_1  (1, 102, 102, 32
   )            0         ['max_pooling2d_2[0][0
   ]']
85   (TFOpLambda
   )
86
87  concatenate_4 (Concatenate  (1, 102, 102, 64
   )            0         ['conv2d_9[0][0
   ]',
88  )
                    'tf.image.resize_with_pad_1[0
89
                   ][0]']
90
91 ==================================================
   ================================================
92 Total params: 39040 (152.50 KB)
93 Trainable params: 39040 (152.50 KB)
94 Non-trainable params: 0 (0.00 Byte)
95 __________________________________________________
   ________________________________________________
96 Model: "SPP-Net"
97 __________________________________________________
   ________________________________________________
98  Layer (type)                Output Shape
                    Param #   Connected to
```

```
 99 ==================================================
    ================================================
100  input_5 (InputLayer)        [(None, 20, 20, 32
    )]          0         []
101
102  max_pooling2d_3 (MaxPoolin  (None, 16, 16, 32
    )           0         ['input_5[0][0]']
103  g2D
    )
104
105  max_pooling2d_4 (MaxPoolin  (None, 12, 12, 32
    )           0         ['input_5[0][0]']
106  g2D
    )
107
108  max_pooling2d_5 (MaxPoolin  (None, 8, 8, 32
    )             0         ['input_5[0][0
    ]']
109  g2D
    )
110
111  tf.image.resize_with_pad_2  (None, 26, 26, 32
    )           0         ['input_5[0][0]']
112   (TFOpLambda
    )
113
114  tf.image.resize_with_pad_3  (None, 26, 26, 32
    )           0         ['max_pooling2d_3[1][0]']
115   (TFOpLambda
    )
116
```

```
116
117  tf.image.resize_with_pad_4  (None, 26, 26, 32
    )           0         ['max_pooling2d_4[1][0]']
118   (TFOpLambda
    )

119

120  tf.image.resize_with_pad_5  (None, 26, 26, 32
    )           0         ['max_pooling2d_5[1][0]']
121   (TFOpLambda
    )

122

123  concatenate_5 (Concatenate  (None, 26, 26, 128
    )          0         ['tf.image.resize_with_pad_2[1
124  )
                    ][0]',
125
                     'tf.image.resize_with_pad_3[1
126
                    ][0]',
127
                     'tf.image.resize_with_pad_4[1
128
                    ][0]',
129
                     'tf.image.resize_with_pad_5[1
130
                    ][0]']
131

132 ==================================================
    ================================================
133 Total params: 0 (0.00 Byte)
134 Trainable params: 0 (0.00 Byte)
135 Non-trainable params: 0 (0.00 Byte)
136 _________________________________________________
    _________________________________________________
137 Model: "CSP-SPP"
```

```
138 _________________________________________________
    _________________________________________________
139  Layer (type)                Output Shape
                    Param #   Connected to

140 =================================================
    =================================================
141  input_4 (InputLayer)        [(None, 26, 26, 256
    )]          0         []
142
143  conv2d_13 (Conv2D)          (None, 24, 24, 256
    )          590080    ['input_4[0][0]']
144
145  conv2d_14 (Conv2D)          (None, 22, 22, 128
    )          295040    ['conv2d_13[0][0]']
146
147  conv2d_15 (Conv2D)          (None, 22, 22, 64
    )           8256      ['conv2d_14[0][0]']
148
149  conv2d_16 (Conv2D)          (None, 20, 20, 64
    )           36928     ['conv2d_15[0][0]']
150
151  conv2d_17 (Conv2D)          (None, 20, 20, 32
    )           2080      ['conv2d_16[0][0]']
152
153  SPP-Net (Functional)        (None, 26, 26, 128
    )          0         ['conv2d_17[0][0]']
154
155  tf.image.resize_with_pad_6  (None, 26, 26, 128
    )          0         ['SPP-Net[0][0]']
156   (TFOpLambda
    )
157
```

```
157
158  concatenate_6 (Concatenate  (None, 26, 26, 256
    )          0         ['tf.image.resize_with_pad_6[0
159  )
                    ][0]',
160
                     'SPP-Net[0][0]']
161

162  conv2d_18 (Conv2D)          (None, 24, 24, 256
    )          590080    ['concatenate_6[0][0]']
163

164  concatenate_7 (Concatenate  (None, 24, 24, 512
    )          0         ['conv2d_13[0][0]',
165  )
                     'conv2d_18[0][0]']
166

167  max_pooling2d_6 (MaxPoolin  (None, 12, 12, 512
    )          0         ['concatenate_7[0][0]']
168  g2D
    )

169


170  conv2d_19 (Conv2D)          (None, 12, 12, 256
    )          131328    ['max_pooling2d_6[0][0]']
171

172 ================================================
    ================================================
173 Total params: 1653792 (6.31 MB)
174 Trainable params: 1653792 (6.31 MB)
175 Non-trainable params: 0 (0.00 Byte)
176 ________________________________________________
    ________________________________________________
177 Model: "CSP"
178 ________________________________________________
    ________________________________________________
179  Layer (type)                Output Shape
```

```
                 Param #   Connected to

==================================================
==================================================
 input_6 (InputLayer)        [(1, 104, 104, 64
)]          0         []

 conv2d_20 (Conv2D)          (1, 102, 102, 32
)            18464     ['input_6[0][0
]']

 conv2d_21 (Conv2D)          (1, 100, 100, 32
)            9248      ['conv2d_20[0][0
]']

 conv2d_22 (Conv2D)          (1, 100, 100, 32
)            9248      ['conv2d_21[0][0
]']

 concatenate_8 (Concatenate  (1, 100, 100, 64
)            0         ['conv2d_21[0][0
]',
 )
                    'conv2d_22[0][0]']

 conv2d_23 (Conv2D)          (1, 100, 100, 32
)            2080      ['concatenate_8[0][0
]']

 max_pooling2d_7 (MaxPoolin  (1, 100, 100, 32
)            0         ['conv2d_23[0][0
]']
 g2D
)
```

```
196
197  tf.image.resize_with_pad_7  (1, 102, 102, 32
    )             0          ['max_pooling2d_7[0][0
    ]']
198   (TFOpLambda
    )

199

200  concatenate_9 (Concatenate  (1, 102, 102, 64
    )             0          ['conv2d_20[0][0
    ]',
201  )
                     'tf.image.resize_with_pad_7[0
202
                    ][0]']
203

204 ==================================================
    ================================================
205 Total params: 39040 (152.50 KB)
206 Trainable params: 39040 (152.50 KB)
207 Non-trainable params: 0 (0.00 Byte)
208 __________________________________________________
    ________________________________________________
209 Model: "CSP"
210 __________________________________________________
    ________________________________________________
211  Layer (type)                Output Shape
                     Param #   Connected to

212 ==================================================
    ================================================
213  input_7 (InputLayer)        [(1, 104, 104, 64
    )]           0          []
214

215  conv2d_24 (Conv2D)          (1, 102, 102, 32
    )             18464      ['input_7[0][0
    ']
```

```
216
217  conv2d_25 (Conv2D)          (1, 100, 100, 32
    )            9248      ['conv2d_24[0][0
    ]']
218
219  conv2d_26 (Conv2D)          (1, 100, 100, 32
    )            9248      ['conv2d_25[0][0
    ]']
220
221  concatenate_10 (Concatenat  (1, 100, 100, 64
    )            0         ['conv2d_25[0][0
    ]',
222  e
    )
                   'conv2d_26[0][0]']
223
224  conv2d_27 (Conv2D)          (1, 100, 100, 32
    )            2080      ['concatenate_10[0][0
    ]']
225
226  max_pooling2d_8 (MaxPoolin  (1, 100, 100, 32
    )            0         ['conv2d_27[0][0
    ]']
227  g2D
    )

228

229  tf.image.resize_with_pad_8  (1, 102, 102, 32
    )            0         ['max_pooling2d_8[0][0
    ]']
230   (TFOpLambda
    )

231
```

```
232  concatenate_11 (Concatenat  (1, 102, 102, 64
    )            0         ['conv2d_24[0][0
    ]',
233  e
    )
                  'tf.image.resize_with_pad_8[0
234
                   ][0]']
235

236 ================================================
    ================================================
237 Total params: 39040 (152.50 KB)
238 Trainable params: 39040 (152.50 KB)
239 Non-trainable params: 0 (0.00 Byte)
240 ________________________________________________
    ________________________________________________
241 Model: "SPP-Net"
242 ________________________________________________
    ________________________________________________
243  Layer (type)                Output Shape
                    Param #   Connected to

244 ================================================
    ================================================
245  input_9 (InputLayer)        [(None, 20, 20, 32
    )]          0         []
246

247  max_pooling2d_9 (MaxPoolin  (None, 16, 16, 32
    )            0         ['input_9[0][0]']
248  g2D
    )

249

250  max_pooling2d_10 (MaxPooli  (None, 12, 12, 32
    )            0         ['input_9[0][0]']
251  ng2D
    )
```

```
252
253  max_pooling2d_11 (MaxPooli  (None, 8, 8, 32
    )              0         ['input_9[0][0
    ]']
254  ng2D
    )
255
256  tf.image.resize_with_pad_9  (None, 26, 26, 32
    )            0         ['input_9[0][0]']
257   (TFOpLambda
    )
258
259  tf.image.resize_with_pad_1  (None, 26, 26, 32
    )            0         ['max_pooling2d_9[1][0]']
260  0 (TFOpLambda
    )
261
262  tf.image.resize_with_pad_1  (None, 26, 26, 32
    )            0         ['max_pooling2d_10[1][0]']
263  1 (TFOpLambda
    )
264
265  tf.image.resize_with_pad_1  (None, 26, 26, 32
    )            0         ['max_pooling2d_11[1][0]']
266  2 (TFOpLambda
    )
267
268  concatenate_12 (Concatenat  (None, 26, 26, 128
    )           0         ['tf.image.resize_with_pad_9[1
269  e
```

```
269
    )
                ][0]',
270
                   'tf.image.resize_with_pad_10[
271
                  1][0]',
272
                   'tf.image.resize_with_pad_11[
273
                  1][0]',
274
                   'tf.image.resize_with_pad_12[
275
                  1][0]']
276

277 ==================================================
    ================================================
278 Total params: 0 (0.00 Byte)
279 Trainable params: 0 (0.00 Byte)
280 Non-trainable params: 0 (0.00 Byte)
281 __________________________________________________
    ________________________________________________
282 Model: "CSP-SPP"
283 __________________________________________________
    ________________________________________________
284  Layer (type)                Output Shape
                    Param #   Connected to

285 ==================================================
    ================================================
286  input_8 (InputLayer)        [(None, 26, 26, 256
    )]        0         []
287

288  conv2d_28 (Conv2D)          (None, 24, 24, 256
    )          590080    ['input_8[0][0]']
289

290  conv2d_29 (Conv2D)          (None, 22, 22, 128
```

```
290 )           295040    ['conv2d_28[0][0]']
291
292  conv2d_30 (Conv2D)          (None, 22, 22, 64
    )           8256      ['conv2d_29[0][0]']
293
294  conv2d_31 (Conv2D)          (None, 20, 20, 64
    )           36928     ['conv2d_30[0][0]']
295
296  conv2d_32 (Conv2D)          (None, 20, 20, 32
    )           2080      ['conv2d_31[0][0]']
297
298  SPP-Net (Functional)        (None, 26, 26, 128
    )           0         ['conv2d_32[0][0]']
299
300  tf.image.resize_with_pad_1  (None, 26, 26, 128
    )           0         ['SPP-Net[0][0]']
301  3 (TFOpLambda
    )
302
303  concatenate_13 (Concatenat  (None, 26, 26, 256
    )           0         ['tf.image.resize_with_pad_13[
304  e
    )
                  0][0]',
305
                     'SPP-Net[0][0]']
306
307  conv2d_33 (Conv2D)          (None, 24, 24, 256
    )           590080    ['concatenate_13[0][0]']
308
309  concatenate_14 (Concatenat  (None, 24, 24, 512
    )           0         ['conv2d_28[0][0]',
```

```
310  e
    )
                  'conv2d_33[0][0]']
311
312  max_pooling2d_12 (MaxPooli  (None, 12, 12, 512
    )          0         ['concatenate_14[0][0]']
313  ng2D
    )
314
315  conv2d_34 (Conv2D)          (None, 12, 12, 256
    )          131328    ['max_pooling2d_12[0][0]']
316
317 ================================================
    ================================================
318 Total params: 1653792 (6.31 MB)
319 Trainable params: 1653792 (6.31 MB)
320 Non-trainable params: 0 (0.00 Byte)
321 ________________________________________________
    ________________________________________________
322 tf.Tensor(
323 [[[[-1.56929455e+01 -1.45838104e+02 -3.73503418e+01
     ...  2.57777832e+02
324      5.75415535e+01  1.22790497e+02]
325    [-1.55190964e+01 -1.45223724e+02 -3.75998459e+01
     ...  2.57540314e+02
326      5.70895920e+01  1.23243019e+02]
327    [-1.53152065e+01 -1.45136093e+02 -3.78853111e+01
     ...  2.57352905e+02
328      5.73750763e+01  1.22634552e+02]
329    ...
330    [-1.55697832e+01 -1.45823502e+02 -3.91250229e+01
     ...  2.56965546e+02
331      5.66590157e+01  1.22351921e+02]
332    [-1.56300831e+01 -1.45858017e+02 -3.90726967e+01
     ...  2.56981873e+02
333      5.66440392e+01  1.22317871e+02]
334    [-2.67140198e+00 -1.12416092e+02 -2.04495773e+01
```

```
334  ...  2.79150879e+02
335      2.14270210e+01  1.09178223e+02]]
336
337   [[-1.57448044e+01 -1.47523743e+02 -3.54778366e+01
     ...  2.58632324e+02
338      5.68170357e+01  1.24219711e+02]
339    [-1.61108780e+01 -1.47441269e+02 -3.62826462e+01
     ...  2.58346649e+02
340      5.65394821e+01  1.24793388e+02]
341    [-1.59570827e+01 -1.47873352e+02 -3.65797195e+01
     ...  2.58787933e+02
342      5.67991409e+01  1.24554199e+02]
343    ...
344    [-1.61551113e+01 -1.47992798e+02 -3.74805908e+01
     ...  2.58241669e+02
345      5.57171593e+01  1.24311676e+02]
346    [-1.62208500e+01 -1.47927048e+02 -3.76281509e+01
     ...  2.58228638e+02
347      5.56272850e+01  1.24547241e+02]
348    [-3.17031455e+00 -1.14424362e+02 -1.89657230e+01
     ...  2.80383392e+02
349      2.03734646e+01  1.11238266e+02]]
350
351   [[-1.62698593e+01 -1.46546844e+02 -3.92792358e+01
     ...  2.60200317e+02
352      5.52994766e+01  1.26255699e+02]
353    [-1.70323944e+01 -1.47496307e+02 -3.76200447e+01
     ...  2.59938080e+02
354      5.62990074e+01  1.25492844e+02]
355    [-1.64378929e+01 -1.46899094e+02 -3.88800087e+01
     ...  2.59954102e+02
356      5.55704308e+01  1.26467018e+02]
357    ...
358    [-1.66494732e+01 -1.47459793e+02 -3.94060249e+01
     ...  2.59360901e+02
359      5.56270447e+01  1.25794472e+02]
360    [-1.66603203e+01 -1.47453979e+02 -3.94736557e+01
     ...  2.59272949e+02
361      5.56148643e+01  1.25851067e+02]
362    [-3.15233302e+00 -1.14094795e+02 -2.04333248e+01
     ...  2.81527161e+02
```

```
363      2.03090725e+01  1.12331825e+02]]
364
365   ...
366
367   [[-1.79445953e+01 -1.49471619e+02 -4.29768410e+01
     ...  2.68092743e+02
368      6.08303070e+01  1.22887573e+02]
369    [-1.80488243e+01 -1.49983185e+02 -4.32275734e+01
     ...  2.68389526e+02
370      5.98050385e+01  1.25062538e+02]
371    [-1.79633503e+01 -1.49940399e+02 -4.31403503e+01
     ...  2.68272034e+02
372      5.98300591e+01  1.25163834e+02]
373    ...
374    [-1.76574593e+01 -1.50137878e+02 -4.22367249e+01
     ...  2.68249756e+02
375      6.13142166e+01  1.24322533e+02]
376    [-1.79492760e+01 -1.48640961e+02 -3.71679955e+01
     ...  2.70488251e+02
377      6.43893204e+01  1.23584007e+02]
378    [-1.22663498e+00 -1.12699394e+02 -1.46529064e+01
     ...  2.96014343e+02
379      2.92451382e+01  1.09681572e+02]]
380
381   [[-1.52627001e+01 -1.42419022e+02 -2.71748772e+01
     ...  2.75936157e+02
382      5.37684784e+01  1.16295815e+02]
383    [-1.43975182e+01 -1.41922974e+02 -2.92492065e+01
     ...  2.73340179e+02
384      5.49794464e+01  1.18129150e+02]
385    [-1.43688574e+01 -1.41893280e+02 -2.91668587e+01
     ...  2.73251801e+02
386      5.50084877e+01  1.18239975e+02]
387    ...
388    [-1.42164593e+01 -1.42251434e+02 -2.82880402e+01
     ...  2.73365234e+02
389      5.68056145e+01  1.16940338e+02]
390    [-1.47394962e+01 -1.45032913e+02 -2.46122704e+01
     ...  2.72726562e+02
391      5.72558327e+01  1.12796272e+02]
392    [ 1.00223446e+00 -1.13487885e+02 -9.48472023e-01
```

```
392  ...  2.98199524e+02
393      1.58505135e+01  1.00964340e+02]]
394
395   [[ 5.38356876e+00 -9.83788071e+01  1.29720898e+01
     ...  2.63472748e+02
396      7.28235626e+01  9.27972717e+01]
397    [ 5.10420895e+00 -9.86941986e+01  1.26432943e+01
     ...  2.63671509e+02
398      7.26128769e+01  9.27546997e+01]
399    [ 5.06598854e+00 -9.86676559e+01  1.26525288e+01
     ...  2.63642059e+02
400      7.26437683e+01  9.27699966e+01]
401    ...
402    [ 5.45350456e+00 -9.90622482e+01  1.33138914e+01
     ...  2.63305023e+02
403      7.49315262e+01  9.20810776e+01]
404    [ 6.44116879e+00 -9.90521469e+01  1.28082190e+01
     ...  2.62318695e+02
405      7.62105331e+01  9.12174072e+01]
406    [ 2.06980057e+01 -8.69503250e+01  2.78244686e+01
     ...  2.81459534e+02
407      4.75627632e+01  9.08115082e+01]]]
408
409
410  [[[-7.77276516e+00 -1.45266891e+02 -3.30775528e+01
     ...  2.26809937e+02
411      1.67667297e+02  8.01402435e+01]
412    [ 1.28974915e-01 -1.47342789e+02 -3.39539795e+01
     ...  2.27198410e+02
413      1.70299255e+02  7.92634811e+01]
414    [ 4.59408379e+00 -1.42857498e+02 -3.39532013e+01
     ...  2.26267502e+02
415      1.74250824e+02  7.88953629e+01]
416    ...
417    [-6.00317478e+00 -1.46585083e+02 -4.03367424e+01
     ...  2.28689178e+02
418      1.75541092e+02  8.48242950e+01]
419    [-5.00833797e+00 -1.52255356e+02 -4.42947426e+01
     ...  2.31118988e+02
420      1.73722321e+02  8.93592834e+01]
421    [ 7.16781425e+00 -1.14167488e+02 -5.05965328e+00
```

```
421  ...  2.48024841e+02
422      1.39573303e+02  9.93358841e+01]]
423
424   [[-9.86407185e+00 -1.40717163e+02 -3.68114777e+01
     ...  2.18622101e+02
425      1.85125244e+02  9.16327057e+01]
426    [-4.51356888e-01 -1.42358093e+02 -3.83698463e+01
     ...  2.17315552e+02
427      1.89567230e+02  9.05620346e+01]
428    [ 4.02861977e+00 -1.35624664e+02 -3.82169609e+01
     ...  2.12575867e+02
429      1.93018219e+02  8.64588623e+01]
430    ...
431    [-4.70001125e+00 -1.39730057e+02 -3.95479965e+01
     ...  2.28209396e+02
432      1.89289429e+02  9.67723236e+01]
433    [-6.95774078e+00 -1.42870453e+02 -4.39668655e+01
     ...  2.33011978e+02
434      1.85527710e+02  1.02996864e+02]
435    [ 3.87606621e+00 -1.09597198e+02  6.85658216e-01
     ...  2.49936279e+02
436      1.48703049e+02  1.12897644e+02]]
437
438   [[-1.04099808e+01 -1.18791382e+02 -2.92150097e+01
     ...  2.08765472e+02
439      1.89066040e+02  1.11582222e+02]
440    [ 4.01763916e-01 -1.20215752e+02 -2.54948406e+01
     ...  2.04726746e+02
441      1.92186081e+02  1.07527260e+02]
442    [ 8.63836288e+00 -1.12697647e+02 -2.77309456e+01
     ...  1.99991089e+02
443      1.97767090e+02  1.04455643e+02]
444    ...
445    [ 3.69476891e+00 -1.24329094e+02 -3.08795891e+01
     ...  2.22692291e+02
446      1.96975739e+02  1.15045227e+02]
447    [ 6.92811966e-01 -1.23519508e+02 -3.57058716e+01
     ...  2.29852631e+02
448      1.95849869e+02  1.23392097e+02]
449    [ 1.31779251e+01 -8.96851196e+01  1.20309668e+01
     ...  2.45937393e+02
```

```
      1.61314377e+02  1.32863174e+02]]

  ...

  [[-1.19477673e+01 -7.57368011e+01 -6.49359703e+00
   ...  1.13716194e+02
     1.32391144e+02  2.75767860e+01]
    [ 3.63336563e-01 -6.93417664e+01 -4.81119156e+00
   ...  1.16829140e+02
     1.33221069e+02  2.89380169e+01]
    [ 8.93341160e+00 -6.32773209e+01 -1.02450695e+01
   ...  1.10139130e+02
     1.38553345e+02  1.85357647e+01]
    ...
    [ 3.18777046e+01 -6.46021042e+01  9.21191025e+00
   ...  1.25134079e+02
     1.50132950e+02  2.29754887e+01]
    [ 4.21240616e+01 -7.00091400e+01  2.03177452e+01
   ...  1.07343323e+02
     1.82214539e+02  1.10666695e+01]
    [ 4.73202400e+01 -4.63797264e+01  3.81014862e+01
   ...  1.13332565e+02
     1.19008125e+02  2.49565620e+01]]

  [[ 5.02523041e+00 -6.42826462e+01 -1.35352440e+01
   ...  1.04692780e+02
     1.14977150e+02  1.54575024e+01]
    [ 6.27570724e+00 -5.98975792e+01 -1.32244473e+01
   ...  1.03839569e+02
     1.17100189e+02  1.38987427e+01]
    [ 1.01602478e+01 -5.29236679e+01 -3.04066448e+01
   ...  9.59239273e+01
     1.21055992e+02  6.88895512e+00]
    ...
    [ 4.42937851e+01 -5.44629173e+01  4.11935234e+00
   ...  1.14258011e+02
     1.35020782e+02 -1.19480639e+01]
    [ 3.94115257e+01 -5.10350189e+01  2.49451542e+00
   ...  8.52706146e+01
     1.33156250e+02 -2.65145340e+01]
    [ 3.97256851e+01 -1.61756134e+01  2.87010994e+01
```

```
479  ...  1.01509254e+02
480      9.34929123e+01 -1.19092369e+01]]
481
482   [[ 2.56304073e+01 -4.22183609e+01  1.67694149e+01
     ...  8.38313370e+01
483      1.24535706e+02  3.01584129e+01]
484    [ 3.08996868e+01 -3.74895058e+01  1.63559704e+01
     ...  8.16885757e+01
485      1.20170975e+02  2.14794846e+01]
486    [ 3.09757328e+01 -4.10826492e+01  1.32568836e+01
     ...  8.28426819e+01
487      1.24009140e+02  1.46005363e+01]
488    ...
489    [ 6.74696121e+01 -2.44844246e+01  2.66566811e+01
     ...  9.73637695e+01
490      1.50597122e+02  2.10994415e+01]
491    [ 7.78200912e+01 -9.12467289e+00  2.87763786e+01
     ...  8.95169373e+01
492      1.46511414e+02  1.41282148e+01]
493    [ 5.22855034e+01  2.23506145e+01  4.22154274e+01
     ...  9.54710312e+01
494      1.08981766e+02  1.79139633e+01]]]
495
496
497  [[[ 4.56510468e+01 -6.64648438e+01 -4.02868652e+01
     ...  8.15001678e+01
498      1.27490150e+02  4.59910393e+00]
499    [ 3.47577667e+01 -5.52226677e+01 -3.29086380e+01
     ...  7.81284943e+01
500      1.19778160e+02 -1.12998772e+01]
501    [ 2.76691551e+01 -4.83053741e+01 -2.54994888e+01
     ...  7.78047943e+01
502      1.19777267e+02 -6.42221069e+00]
503    ...
504    [ 3.02774525e+01 -3.85850449e+01 -3.65788078e+00
     ...  7.76444092e+01
505      1.32288223e+02  1.30905418e+01]
506    [ 3.51393547e+01 -3.18811188e+01 -1.07973728e+01
     ...  6.85270996e+01
507      1.29670624e+02 -1.99608383e+01]
508    [ 3.28532791e+01 -1.87812920e+01  1.49854603e+01
```

```
508  ...  7.77924500e+01
509      8.77274628e+01  1.36141300e+01]]
510
511   [[ 4.58666039e+01 -6.13223724e+01 -1.43890877e+01
     ...  9.92140121e+01
512      1.54273987e+02  3.08552132e+01]
513    [ 1.64360657e+01 -5.04548492e+01 -1.85559578e+01
     ...  8.78983536e+01
514      1.33297043e+02  1.04917908e+00]
515    [ 1.92477798e+01 -4.68036652e+01 -1.46129093e+01
     ...  8.91899872e+01
516      1.38267914e+02  1.13622990e+01]
517    ...
518    [ 2.30615826e+01 -4.38088722e+01  3.59917259e+00
     ...  6.57004395e+01
519      1.21032227e+02  8.80097771e+00]
520    [ 2.55457993e+01 -2.86765251e+01  3.93417263e+00
     ...  7.86969147e+01
521      1.23758698e+02  1.63020515e+00]
522    [ 2.24067535e+01 -1.90332546e+01  1.77276173e+01
     ...  7.81783447e+01
523      9.54161606e+01  2.28526573e+01]]
524
525   [[ 5.71092796e+01 -5.25240250e+01 -4.50183868e-01
     ...  1.22250320e+02
526      1.66168243e+02  1.81501675e+01]
527    [ 4.31627922e+01 -3.55214386e+01 -6.92947197e+00
     ...  9.83889618e+01
528      1.35480209e+02 -3.67641220e+01]
529    [ 2.79002094e+01 -3.80818481e+01 -2.30904007e+01
     ...  8.66417313e+01
530      1.36525192e+02 -1.43190250e+01]
531    ...
532    [ 3.96372604e+01 -3.09868050e+01 -9.47643661e+00
     ...  7.39594803e+01
533      1.11200729e+02 -2.46972504e+01]
534    [ 3.44223289e+01 -3.39592667e+01  2.08325768e+00
     ...  7.16585541e+01
535      1.09694603e+02 -1.33400497e+01]
536    [ 2.87442970e+01 -1.92709656e+01  1.68183041e+01
     ...  7.84680634e+01
```

```
537      7.78918228e+01  3.00018024e+00]]
538
539   ...
540
541   [[ 6.60685883e+01 -2.43389664e+01  3.72660866e+01
     ...  1.23062012e+02
542      1.77372330e+02 -3.07973709e+01]
543    [ 6.98238144e+01 -5.64447861e+01  5.44798546e+01
     ...  1.20064034e+02
544      1.97064804e+02  1.13360538e+01]
545    [ 4.01266327e+01 -3.31984138e+01  3.65982971e+01
     ...  1.21533356e+02
546      1.62223480e+02 -6.68830109e+00]
547    ...
548    [ 3.85629539e+01 -2.07217979e+01  1.58701706e+01
     ...  8.10864563e+01
549      1.13560051e+02  5.43896294e+00]
550    [ 3.52058220e+01 -3.01295509e+01  2.42377968e+01
     ...  8.40931396e+01
551      1.07246780e+02  5.09218597e+00]
552    [ 3.47771454e+01 -1.56075821e+01  5.81925507e+01
     ...  9.29611969e+01
553      7.66302872e+01  1.30641155e+01]]
554
555   [[ 7.94225769e+01 -4.12645531e+01  5.37882729e+01
     ...  1.03713676e+02
556      1.85863632e+02 -2.31781349e+01]
557    [ 8.32941208e+01 -6.43834839e+01  6.00974350e+01
     ...  1.06587616e+02
558      1.87583725e+02 -3.15146961e+01]
559    [ 3.64095917e+01 -3.76358528e+01  2.82759991e+01
     ...  7.08139038e+01
560      1.57932495e+02 -2.14483509e+01]
561    ...
562    [ 4.11954117e+00 -4.68157425e+01  3.14174385e+01
     ...  8.16881409e+01
563      1.41840027e+02 -1.49136906e+01]
564    [ 8.91360092e+00 -6.48427124e+01  3.24141388e+01
     ...  8.63964539e+01
565      1.25318817e+02 -5.04369354e+00]
566    [ 2.02910004e+01 -4.41121025e+01  6.52419891e+01
```

```
566  ...  1.01603828e+02
567       9.48143387e+01  3.76517487e+00]]
568
569   [[ 8.44889679e+01  6.74751663e+00  5.72393875e+01
     ...  1.07616890e+02
570       1.92575745e+02  2.06408043e+01]
571    [ 1.05223907e+02 -1.02524853e+01  3.62286377e+01
     ...  1.17152634e+02
572       1.89660675e+02  1.41586838e+01]
573    [ 8.34436646e+01 -1.95510674e+01  4.95339432e+01
     ...  9.69127350e+01
574       1.76449280e+02  2.05368271e+01]
575    ...
576    [ 5.07246819e+01 -2.58532887e+01  6.70583420e+01
     ...  1.11921555e+02
577       1.37542740e+02  1.43398800e+01]
578    [ 5.54584389e+01 -2.53865376e+01  6.05946541e+01
     ...  1.22246933e+02
579       1.22130669e+02  8.02160645e+00]
580    [ 5.80940132e+01 -1.97236481e+01  8.06735382e+01
     ...  1.26024307e+02
581       8.75464020e+01  1.39942589e+01]]]
582
583
584  [[[ 1.55794306e+01 -1.20596085e+02 -4.00692291e+01
     ...  2.32198669e+02
585       9.71308975e+01  6.59402313e+01]
586    [ 1.01585941e+01 -1.13952370e+02 -3.82268448e+01
     ...  2.31525970e+02
587       7.90950851e+01  6.18730011e+01]
588    [-1.86883926e+00 -1.20556427e+02 -3.13798866e+01
     ...  2.29012451e+02
589       8.44035492e+01  6.10188293e+01]
590    ...
591    [-1.15864697e+01 -1.15904404e+02 -3.70721588e+01
     ...  2.23351227e+02
592       8.60104599e+01  5.34120598e+01]
593    [-3.25832367e-02 -1.18317184e+02 -3.40178070e+01
     ...  2.29817627e+02
594       8.15039062e+01  5.27868919e+01]
595    [ 1.51830854e+01 -8.51049271e+01 -1.05678749e+01
```

```
595  ...  2.50404297e+02
596      4.96485748e+01  6.17064247e+01]]
597
598   [[ 2.37123528e+01 -1.19848175e+02 -3.71137276e+01
     ...  2.10589935e+02
599      1.08272049e+02  9.82524490e+01]
600    [ 1.29594536e+01 -1.15140923e+02 -4.26018562e+01
     ...  2.16350220e+02
601      1.03995834e+02  8.43472443e+01]
602    [ 2.08965607e+01 -1.11849792e+02 -3.63373070e+01
     ...  2.10356674e+02
603      9.84346161e+01  8.91858139e+01]
604    ...
605    [ 1.76834564e+01 -1.12230698e+02 -4.04638824e+01
     ...  2.13946457e+02
606      9.98451385e+01  9.31028595e+01]
607    [ 1.78657303e+01 -1.12156158e+02 -4.01290627e+01
     ...  2.14416718e+02
608      1.00386551e+02  9.32159195e+01]
609    [ 3.33971825e+01 -7.85904236e+01 -1.48477459e+01
     ...  2.33559875e+02
610      6.75966721e+01  9.84411087e+01]]
611
612   [[-7.53475046e+00 -1.20476471e+02 -3.08294525e+01
     ...  2.16912537e+02
613      1.21759598e+02  9.60631943e+01]
614    [-6.01978493e+00 -1.20039253e+02 -3.16922874e+01
     ...  2.16208496e+02
615      1.21128227e+02  9.47741547e+01]
616    [-7.26607704e+00 -1.19987999e+02 -3.19296818e+01
     ...  2.16365646e+02
617      1.20192101e+02  9.39404907e+01]
618    ...
619    [-8.07579327e+00 -1.17255539e+02 -3.14913750e+01
     ...  2.17255951e+02
620      1.20930168e+02  9.09562378e+01]
621    [-7.22683096e+00 -1.17501060e+02 -3.17039623e+01
     ...  2.16840210e+02
622      1.20444664e+02  9.03542862e+01]
623    [ 7.58897591e+00 -8.38924866e+01 -6.44559765e+00
     ...  2.35437256e+02
```

```
624      8.76988678e+01  9.55626221e+01]]
625
626   ...
627
628   [[-1.25322542e+01 -6.42006683e+01 -1.40774498e+01
     ...  1.16778046e+02
629      1.07827538e+02  2.21646881e+01]
630    [-1.24252186e+01 -7.41463928e+01 -9.14113235e+00
     ...  1.18466255e+02
631      1.04125801e+02  3.19609642e+01]
632    [-9.97055817e+00 -6.50760345e+01 -2.59757347e+01
     ...  1.17637497e+02
633      1.08721619e+02  2.92295475e+01]
634    ...
635    [-2.22588577e+01 -6.83969803e+01 -3.80954285e+01
     ...  1.13238098e+02
636      1.09263306e+02  2.94615059e+01]
637    [-2.21621685e+01 -6.81864090e+01 -3.89499474e+01
     ...  1.14809021e+02
638      1.08799042e+02  2.94781132e+01]
639    [-7.39255333e+00 -5.19122200e+01 -1.49155264e+01
     ...  1.28311920e+02
640      8.40810318e+01  3.26702728e+01]]
641
642   [[-4.80719376e+00 -5.57491646e+01 -1.61030464e+01
     ...  1.09368210e+02
643      1.03156151e+02  4.08184624e+01]
644    [-3.91500401e+00 -5.37600861e+01 -1.82542038e+01
     ...  1.03463936e+02
645      1.02072083e+02  4.04013023e+01]
646    [-1.05789957e+01 -5.35933990e+01 -2.71335621e+01
     ...  1.10029556e+02
647      9.99990616e+01  3.91138115e+01]
648    ...
649    [-1.70042191e+01 -5.89774704e+01 -3.46740570e+01
     ...  1.10786400e+02
650      9.92662125e+01  4.03513260e+01]
651    [-1.93580704e+01 -6.02990112e+01 -3.51268692e+01
     ...  1.11162178e+02
652      9.85922928e+01  3.99723625e+01]
653    [-6.82684517e+00 -4.43143578e+01 -1.46604967e+01
```

```
653  ...  1.25487213e+02
654      7.63794403e+01  4.28984985e+01]]
655
656   [[ 1.56289978e+01 -3.31070251e+01 -5.56753349e+00
     ...  1.10888191e+02
657      9.51877594e+01  4.04949951e+01]
658    [ 1.52317076e+01 -3.60503311e+01 -7.89433146e+00
     ...  1.09215309e+02
659      9.58218231e+01  4.31459236e+01]
660    [ 1.04541674e+01 -3.67780037e+01 -1.24266062e+01
     ...  1.09736839e+02
661      9.48748169e+01  4.29733810e+01]
662    ...
663    [ 5.91470528e+00 -4.10750351e+01 -2.01880569e+01
     ...  1.12132828e+02
664      8.74105148e+01  4.35122719e+01]
665    [ 5.66353035e+00 -4.21802368e+01 -2.00067310e+01
     ...  1.13032860e+02
666      8.60828476e+01  4.35412140e+01]
667    [ 1.65546417e+01 -2.91806393e+01 -5.63043118e+00
     ...  1.23953453e+02
668      6.68862000e+01  4.59240799e+01]]]], shape=(4,
    12, 12, 256), dtype=float32)
669
670 Process finished with exit code 0
671
```

Appendix D - EfficientDet .py File

File - C:\Users\lbsch\OneDrive\School\CS 797O - NN and DL\Project\efficientdet.py

```
# EfficientDet Inference

# References:
# https://www.kaggle.com/models/tensorflow/efficientdet
# https://www.kaggle.com/datasets/dasmehdixtr/drone-dataset-uav/

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow import keras
import os
import numpy as np

# Load image dataset
img_path = os.path.join(os.getcwd(), 'drone_dataset_xml_format', 'drone_jpgs')
files = os.listdir(img_path)
img_height = 416
img_width = 416

# Load EfficientDet d0v1
detector = hub.load("https://kaggle.com/models/tensorflow/efficientdet/frameworks/TensorFlow2/variations/d0/versions/1")

for img in files:
    # Load image and convert to a tf tensor
    path = os.path.join(img_path, img)
    input_img = keras.utils.load_img(path, target_size=(img_height, img_width))
    input_tensor = tf.convert_to_tensor(input_img)
    # Add batch dimension to tensor to be compatible with Keras CNN input shapes
    input_tensor = np.expand_dims(input_tensor, axis=0)
    output = detector(input_tensor)
    output_class = output["detection_classes"]
    print(output_class)

```

Appendix E - EfficientDet Console Output

File - efficientdet

```
 1 C:\Users\lbsch\anaconda3\envs\CS898_py310\python.exe
    "C:\Users\lbsch\OneDrive\School\CS 7970 - NN and DL\
   Project\efficientdet.py"
 2 2023-12-12 15:36:57.959684: I tensorflow/core/
   platform/cpu_feature_guard.cc:182] This TensorFlow
   binary is optimized to use available CPU instructions
    in performance-critical operations.
 3 To enable the following instructions: SSE SSE2 SSE3
   SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in
   other operations, rebuild TensorFlow with the
   appropriate compiler flags.
 4 WARNING:absl:Importing a function (
   __inference___call___32344) with ops with unsaved
   custom gradients. Will likely fail if a gradient is
   requested.
 5 WARNING:absl:Importing a function (
   __inference_EfficientDet-
   D0_layer_call_and_return_conditional_losses_97451)
   with ops with unsaved custom gradients. Will likely
   fail if a gradient is requested.
 6 WARNING:absl:Importing a function (
   __inference_bifpn_layer_call_and_return_conditional_l
   osses_77595) with ops with unsaved custom gradients.
   Will likely fail if a gradient is requested.
 7 WARNING:absl:Importing a function (
   __inference_EfficientDet-
   D0_layer_call_and_return_conditional_losses_103456)
   with ops with unsaved custom gradients. Will likely
   fail if a gradient is requested.
 8 WARNING:absl:Importing a function (
   __inference_EfficientDet-
   D0_layer_call_and_return_conditional_losses_93843)
   with ops with unsaved custom gradients. Will likely
   fail if a gradient is requested.
 9 WARNING:absl:Importing a function (
   __inference_EfficientDet-
   D0_layer_call_and_return_conditional_losses_107064)
   with ops with unsaved custom gradients. Will likely
   fail if a gradient is requested.
10 WARNING:absl:Importing a function (
   __inference_bifpn_layer_call_and_return_conditional_l
```

```
osses_75975) with ops with unsaved custom gradients. Will likely fail if a gradient is requested.
tf.Tensor(
[[ 5.  5.  5.  5.  5. 35.  5. 33.  5.  1.  5. 28.  5.  5.  3. 38. 42.  3.
  36.  3.  5.  5.  5. 33.  1.  5. 33. 35.  5.  5. 28.  9. 33.  1. 31. 33.
   1.  5.  5.  5.  1.  3. 37. 28.  8.  1.  8. 28. 40.  5.  1.  5. 21.  5.
  38.  8.  3. 32.  5.  3.  5. 31.  1.  3. 72.  5.  5. 36. 27.  5.  1. 35.
  27. 36. 33. 16. 27.  1. 27. 44.  1. 33.  3. 77.  3. 16.  5. 16. 65.  1.
   5. 65. 27. 34. 36.  1. 38.  1.  5.  1.]], shape=(1, 100), dtype=float32)
tf.Tensor(
[[ 5.  5. 13. 16.  5.  1.  1.  5.  5.  7. 33.  1.  1.  1.  1. 76. 75.
  24. 28. 85.  1.  5.  1.  1. 74. 77. 44.  1.  1.  1. 34. 77.  1.  5. 38.
   5.  1. 74.  1. 16.  1. 31.  1.  1.  1. 37.  1. 65. 27.  5.  1. 27.  5.
  37.  1. 75.  1. 76. 16.  5. 84.  5. 16.  1. 44.  1. 37.  3. 90.  5.  1.
  28.  1. 27. 77.  1. 21.  3. 10. 16.  5. 31.  1.  1. 16.  1.  1. 16. 37.
  90. 14. 16.  5.  1. 37.  5. 44.  1. 13.]], shape=(1, 100), dtype=float32)
tf.Tensor(
[[ 5.  5.  5.  1.  5.  5.  5.  5. 16.  3. 16.  5. 13.  5.  1. 33.  5. 28.
   5. 38.  5.  5. 36. 42. 24. 42. 16.  5.  1. 16.  8.  9.  5.  1. 13.  5.
   5.  5. 36. 38.  5. 16. 10.  1. 33. 35. 21.  5. 28. 28. 21.  5.  5. 16.
   5. 16. 16. 27.  5.  5. 16. 16.  5.  5. 27. 10. 13. 13.  5.  5. 38.  1.
   5.  5.  1.  7. 16. 37. 13.  5.  5. 13. 36.  5.  5.  5. 34.  5. 14.  5.
  16.  5. 14.  5. 38.  5.  1. 16.  5. 10.]], shape=(1, 100), dtype=float32)
```

```
32 tf.Tensor(
33 [[ 5. 16. 24. 13.  5.  1. 17.  5. 33. 14.  5.  1.  1
   .  3.  7. 85. 21. 74.
34    5. 75. 34.  1.  5.  5.  5. 10. 31.  5. 16. 89.  1
   . 87. 28. 33. 16.  1.
35   85. 11. 38. 77. 44. 27.  1. 74. 44. 37. 10. 10. 31
   . 14. 37. 18. 13. 31.
36   76.  5. 13.  5.  1.  1.  4. 76. 65.  5. 10.  5.  1
   .  1. 36.  5. 77. 23.
37    5.  1. 31. 15. 16. 28.  1. 33. 37.  3. 75. 16. 16
   .  1. 16.  1. 74. 10.
38    1. 44. 22. 51. 13. 10. 77.  1.  1.  5.]], shape=(1
   , 100), dtype=float32)
39 tf.Tensor(
40 [[ 5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  5.  1. 33
   .  5. 10.  1. 77.  1.
41   85. 31.  5. 65. 74.  5.  5.  5.  5.  1. 84.  5.  1
   .  1.  5. 37.  5. 77.
42   27.  3. 33.  5.  5.  1.  5.  5.  5.  1.  5.  5.  1
   . 44. 37.  5. 33. 85.
43   37. 72.  5.  5. 44.  5.  5.  5. 31. 13. 10.  5.  5
   .  1.  5.  5. 31. 27.
44    1.  5. 36.  5.  5. 16.  1. 16. 16. 16. 28.  1. 27
   . 72.  5.  5.  3. 16.
45    5.  5.  1. 75.  5. 38.  5.  3. 14.  1.]], shape=(1
   , 100), dtype=float32)
46 tf.Tensor(
47 [[ 5.  5.  5.  5.  5.  5.  5.  1.  5.  5.  5.  5.  3
   .  5. 16. 38.  1.  3.
48    8.  5.  1.  3.  8.  5.  1. 24.  5.  5. 21.  5.  5
   .  5. 36.  8.  5.  9.
49    4.  1.  5. 36. 13.  5.  5.  1. 21.  3. 28.  5.  5
   .  1.  1.  7.  1.  5.
50    5. 21.  1.  5. 42.  5.  5.  5.  1.  5.  5.  5.  1
   . 38.  1.  1.  1.  5.
51   33.  1. 16.  8.  5.  5.  1.  9. 36.  5. 38.  5.  5
   .  3.  9.  5.  1.  1.
52   15.  5.  6. 38. 35.  5. 35.  5.  1.  6.]], shape=(1
   , 100), dtype=float32)
53 tf.Tensor(
54 [[ 5.  5.  5.  1.  5.  1.  1.  5.  5.  1.  1.  5.  5
```

```
54 .  5.  1.  5. 27.  1.
55   10.  5. 14.  5.  3. 27.  5.  5.  1.  5.  1. 10.  1
   .  1.  3. 27. 10.  1.
56   38. 31. 13.  1. 10. 16. 28.  5.  1.  1.  1.  5.  5
   . 33. 31.  5.  1.  5.
57   37. 10.  1.  5. 37.  1.  1.  5.  1.  5.  5.  1. 16
   .  1.  5.  5.  1. 10.
58   37. 65.  1. 31.  1.  1.  1. 28. 16. 27. 37.  5. 13
   .  1.  5.  1.  5.  1.
59    5.  5. 10.  1.  1. 16.  5.  5. 17.  5.]], shape=(1
   , 100), dtype=float32)
60 tf.Tensor(
61 [[ 2. 52. 86.  4. 44. 11. 64. 15. 70. 51. 31. 44. 33
   . 19. 85. 41. 62. 20.
62   44. 67. 18.  1. 44. 44. 27. 88. 40. 87. 61. 86. 28
   . 24. 44. 17.  3. 84.
63   44. 64. 34. 86. 55. 79. 81. 31.  7. 16. 21. 31.  8
   .  9. 15.  2.  1. 15.
64   50. 67. 64. 11. 32.  1. 63. 65.  6.  2. 22.  1. 47
   .  2. 47. 52.  2. 15.
65   31. 47. 15. 52. 15. 15. 14. 44. 64. 15.  1. 64. 44
   . 38. 39. 42. 67. 15.
66   86. 62. 81.  1. 11. 90. 31. 78. 62. 86.]], shape=(1
   , 100), dtype=float32)
67 tf.Tensor(
68 [[ 5.  5.  1. 10. 10.  1. 10.  5.  5. 10. 10.  1.  5
   . 13.  1.  1.  5.  5.
69    1.  1.  5.  8.  1.  1. 14.  5. 10.  5. 10.  5.  1
   . 33. 21.  1. 21. 21.
70    1. 13.  5.  3.  5. 27. 10. 13.  5. 13.  5.  1.  1
   . 14. 44. 77.  5.  5.
71    5.  5.  5.  1.  5.  5.  1.  8. 10.  5.  1.  5. 85
   .  5.  5.  5.  5.  1.
72    7.  5.  5. 16.  1. 10.  1.  1. 10.  5. 21.  5.  5
   .  5.  5. 27.  1.  1.
73    3.  5.  5.  5.  8.  5. 10.  1. 85.  5.]], shape=(1
   , 100), dtype=float32)
74 tf.Tensor(
75 [[38.  5. 77. 75. 75. 31. 28. 84. 75. 16.  3. 77. 34
   . 73. 87.  4. 77. 43.
76   27. 33. 31. 31. 41. 28. 77. 76. 84. 75. 74. 24. 36
```

```
 76 . 85. 34.  5. 27. 77.
 77   85.  9. 89. 34. 28.  3. 28. 28. 87. 42. 75. 35. 85
    . 43. 73. 34. 31. 33.
 78   76. 87.  1. 31.  1.  5. 89. 85. 37. 32. 27. 37. 65
    . 87. 74.  2. 33. 33.
 79    3. 87. 39. 27. 75. 28. 28.  1. 43. 15. 72. 84. 87
    . 52. 47. 40.  1. 33.
 80   16. 37. 27.  1. 37.  1. 89. 75. 34. 74.]], shape=(
    1, 100), dtype=float32)
 81 tf.Tensor(
 82 [[ 5.  5.  5.  5.  5.  1.  6. 10.  1.  1.  5.  3.  1
    .  1.  1.  1.  5.  8.
 83   10.  1.  1. 28.  1.  1.  3.  1.  1. 10.  1.  1.  1
    .  1.  5.  1.  3.  5.
 84    5. 64.  1.  5.  3. 28.  1. 85.  1. 10.  1.  1.  1
    .  8.  1.  1.  2. 10.
 85   10.  1.  9.  7.  1.  5.  1.  5. 13.  5.  1. 57. 65
    . 33. 13. 16.  1.  3.
 86   13.  1.  1.  5.  7.  1. 33.  1.  3.  7. 55. 13.  8
    .  5. 38.  3. 35.  5.
 87   13.  2. 44.  5. 38.  1. 16.  3.  5. 22.]], shape=(
    1, 100), dtype=float32)
 88 tf.Tensor(
 89 [[ 5.  5.  5.  5.  5.  5.  5.  3.  5.  8.  3.  5.  5
    .  8.  5.  9. 33. 10.
 90   21.  3.  3.  3.  1.  7. 13.  1.  1.  5. 38.  5.  8
    .  9. 28. 35. 28. 38.
 91   10.  5. 42.  5.  5.  5.  5.  5. 16. 36.  8. 36.  5
    . 13.  1.  7.  7. 10.
 92   33. 35.  5.  3. 13.  5.  1.  5.  5.  5.  5.  1.  8
    .  5.  1.  8.  5.  5.
 93   28.  5. 16. 15. 65.  6. 13. 38.  5.  5. 38. 16.  5
    .  5.  1.  7. 24.  5.
 94    3. 21.  5.  4. 42. 36.  5.  3. 10.  5.]], shape=(
    1, 100), dtype=float32)
 95 tf.Tensor(
 96 [[ 2.  2. 87.  2. 41. 87. 28.  4. 31.  1. 87. 41. 31
    . 28. 52. 31. 15. 67.
 97   62. 87.  2. 11. 28. 52. 87. 28. 39.  4. 31. 90. 40
    . 62. 34. 15. 31.  1.
 98   40. 39. 85. 84.  4. 87. 32. 90. 41. 67.  2. 16. 38
```

```
 98 . 44. 86. 89. 49. 31.
 99   27.  1. 44. 32. 39. 50. 52. 44. 38. 49.  2.  4. 32
    . 64. 85. 28. 44. 62.
100   84. 11. 84. 27. 84. 33. 85. 41. 49. 84. 87. 15. 87
    . 27. 87. 67. 44.  2.
101   62.  3. 86. 84.  1. 51. 52. 89. 42. 44.]], shape=(
    1, 100), dtype=float32)
102 tf.Tensor(
103 [[ 5.  5.  5.  5.  1.  1.  5.  3. 38.  5.  5.  1.  7
    . 35.  5.  5. 28. 33.
104    3. 36.  5.  5.  5.  8. 21. 16. 13.  5.  5.  1. 38
    . 36. 16.  8.  1.  1.
105    9.  1.  1.  5.  5. 28.  1.  5.  1.  5.  1.  9. 38
    .  1.  5.  1.  1.  3.
106    5. 24. 42.  5.  1. 65. 35. 42. 75.  3.  5.  5. 27
    . 35.  1.  1.  5.  3.
107   33. 27. 85. 28. 35.  1. 38.  1.  1. 16.  1.  1.  5
    .  1.  1.  5.  3. 10.
108   36.  1.  3.  5.  1.  5.  5.  1.  5. 31.]], shape=(
    1, 100), dtype=float32)
109 tf.Tensor(
110 [[ 5. 38. 24.  5. 10.  3.  5. 10. 13.  5. 10.  7.  6
    .  5. 10. 10.  8.  1.
111   10. 85.  5. 10. 10.  5.  1.  5.  1. 13.  1.  3.  5
    . 28. 28. 13. 10.  5.
112    5. 28. 10. 10.  3. 10.  1. 13.  9. 38. 38.  3. 28
    . 13. 13.  4. 25. 14.
113    5. 10. 10.  1. 10.  1.  3.  5.  8. 85. 10.  5. 10
    . 28. 10.  3. 16.  3.
114   33. 24.  8. 38.  3. 13.  1. 10.  1. 10. 38. 10.  5
    . 16. 10.  1. 27.  1.
115   16. 10.  3. 10. 36. 38. 13. 31.  8.  1.]], shape=(
    1, 100), dtype=float32)
116 tf.Tensor(
117 [[ 5.  5. 44. 10.  5.  1. 10.  1.  1. 14. 11. 10.  5
    .  7. 33.  5. 31. 10.
118   14.  5.  5.  1.  3. 13.  1.  1. 85. 10.  8. 16. 13
    . 65. 44.  1. 10. 77.
119   33. 14. 21.  3. 16.  1.  1. 13. 14. 44. 31.  5.  1
    .  5. 77. 16. 10. 24.
120   13. 11. 44.  3. 10.  1. 13.  3. 37.  1.  5. 44. 16
```

```
120 . 16. 44. 85.  1.  1.
121   47. 10. 27. 23. 28. 28. 44.  5.  1. 72.  8. 10. 38
    . 72. 14. 16.  5. 64.
122   31. 31.  5. 85.  1.  1. 85. 62.  1.  7.]], shape=(
    1, 100), dtype=float32)
123 Traceback (most recent call last):
124   File "C:\Users\lbsch\OneDrive\School\CS 7970 - NN
    and DL\Project\efficientdet.py", line 29, in <module
    >
125     output = detector(input_tensor)
126              ^^^^^^^^^^^^^^^^^^^^^^
127   File "C:\Users\lbsch\anaconda3\envs\CS898_py310\
    Lib\site-packages\tensorflow\python\saved_model\load
    .py", line 762, in _call_attribute
128     return instance.__call__(*args, **kwargs)
129            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
130   File "C:\Users\lbsch\anaconda3\envs\CS898_py310\
    Lib\site-packages\tensorflow\python\util\
    traceback_utils.py", line 150, in error_handler
131     return fn(*args, **kwargs)
132            ^^^^^^^^^^^^^^^^^^
133   File "C:\Users\lbsch\anaconda3\envs\CS898_py310\
    Lib\site-packages\tensorflow\python\eager\
    polymorphic_function\polymorphic_function.py", line
    825, in __call__
134     result = self._call(*args, **kwds)
135              ^^^^^^^^^^^^^^^^^^^^^^^^^
136   File "C:\Users\lbsch\anaconda3\envs\CS898_py310\
    Lib\site-packages\tensorflow\python\eager\
    polymorphic_function\polymorphic_function.py", line
    864, in _call
137     results = self._variable_creation_fn(*args, **
    kwds)
138               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ^^^
139   File "C:\Users\lbsch\anaconda3\envs\CS898_py310\
    Lib\site-packages\tensorflow\python\eager\
    polymorphic_function\tracing_compiler.py", line 148
    , in __call__
140     return concrete_function._call_flat(
141            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
  File "C:\Users\lbsch\anaconda3\envs\CS898_py310\Lib\site-packages\tensorflow\python\eager\polymorphic_function\monomorphic_function.py", line 1349, in _call_flat
    return self._build_call_outputs(self._inference_function(*args))
                                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\lbsch\anaconda3\envs\CS898_py310\Lib\site-packages\tensorflow\python\eager\polymorphic_function\atomic_function.py", line 196, in __call__
    outputs = self._bound_context.call_function(
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\lbsch\anaconda3\envs\CS898_py310\Lib\site-packages\tensorflow\python\eager\context.py", line 1457, in call_function
    outputs = execute.execute(
              ^^^^^^^^^^^^^^^^
  File "C:\Users\lbsch\anaconda3\envs\CS898_py310\Lib\site-packages\tensorflow\python\eager\execute.py", line 53, in quick_execute
    tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
KeyboardInterrupt

Process finished with exit code -1073741510 (0xC000013A: interrupted by Ctrl+C)

```