DAFTAR PUSTAKA

Amwin, Aldhiyatika. 2021. "Deteksi Dan Klasifikasi Kendaraan Berbasis Algoritma You Only Look Once (YOLO)." *Universitas Islam Indonesia*.

B. S, Rekha. 2020. "Object Detection Using Region Based Convolutional Neural Network: A Survey." *International Journal for Research in Applied Science and Engineering Technology* 8(7):1927–32.

Bagian, Pengolahan Citra and I. F. Interpretasi. 2021. "Pengantar Interpretasi Dan." (Bagian 2).

Charli, Fino, Hadi Syaputra, Muhammad Akbar, Siti Sauda, and Febriyanti Panjaitan. 2020. "Implementasi Metode Faster Region Convolutional Neural Network (Faster R-CNN) Untuk Pengenalan Jenis Burung Lovebird." *Journal of Information Technology Ampera* 1(3):185–97.

Cholissodin, Imam and Arief Andy Soebroto. 2021. "AI , MACHINE LEARNING & DEEP LEARNING (Teori & Implementasi)." (December).

Czum, Julianna M. 2020. "Dive Into Deep Learning." *Journal of the American College of Radiology* 17(5):637–38.

Guenard, Rebecca. 2021. *Poisson from a Petri Dish*. Vol. 32.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2020. "Mask R-CNN." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42(2):386–97.

Munir, Rinaldi. 2013. "Pengantar Pengolahan Citra." *Pengolahan Citra Digital* (Bagian 1):1–10.

Panjaitan, Amyda Suryati. 2012. "Pemeliharaaan Larva Udang Vannamei (Litopenaeus Vannamei, Boone 1931) Dengan Pemberian Jenis Fitoplangkton Yang Berbeda." *Universitas Terbuka Jakarta 2012* 1–148.

PARDEDE, JASMAN and HENDRI HARDIANSAH. 2022. "Deteksi Objek Kereta Api Menggunakan Metode Faster R-CNN Dengan Arsitektur VGG 16." *MIND Journal* 7(1):21–36.

Pratama, Nando Rusrin. 2021. "Sistem Pengenalan Sayur-Sayuran Dengan Metode Image Processing."

Putra, Darma. 2010. "Pengolahan Citra Digital." (April):420.

Setiawan, Hironimus Hendra. 2018. "Klasifikasi Jenis Buah Pisang Dengan Image Processing Menggunakan Methode Classification Of Type Of Banana Fruits With Image Processing Using Backpropagation Method." *Universitas Sanata Dharma*.

Julianita. 2018. "Analisis Efesiensi Faktor Produksi Tambak Udang namei." *Skripsi FAKULTAS PERTANIAN UNIVERSITAS HAMMADIYAH SUMATERA UTARA* 1–61.

an, S. 2020. "Object Detection Using Deep Learning Algorithm CNN."

International Journal for Research in Applied Science and Engineering Technology 8(7):1578–84.

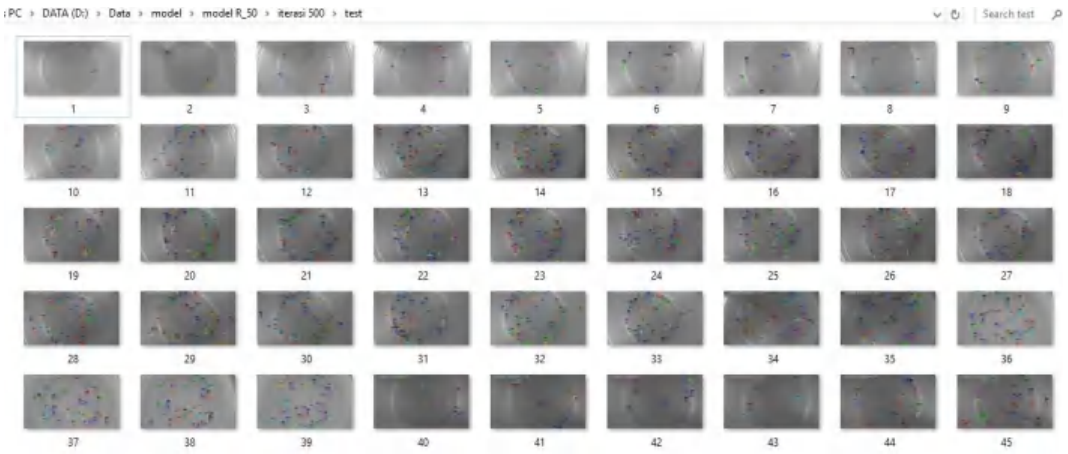
Syarifah. 2018. "Deep Learning Object Detection Pada Video." *Deep Learning Object Detection Pada Video Menggunakan Tensorflow Dan Convolutional Neural Network.*

Wulandari, A. 2020. *Estimasi Beban Limbah Nutrien Terhadap Daya Dukung Lingkungan Untuk Budidaya Udang Vannamei (Litopenaeus Vannamei) Semi Intensif Di Desa Banjar Kemuning.*

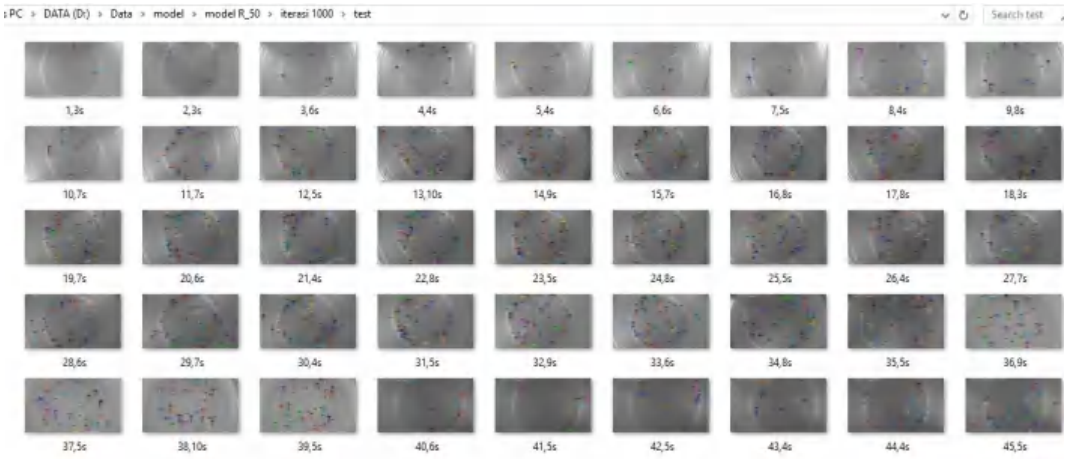
LAMPIRAN

Lampiran 1. Hasil Pengujian *Resnet*-50 dengan FPN

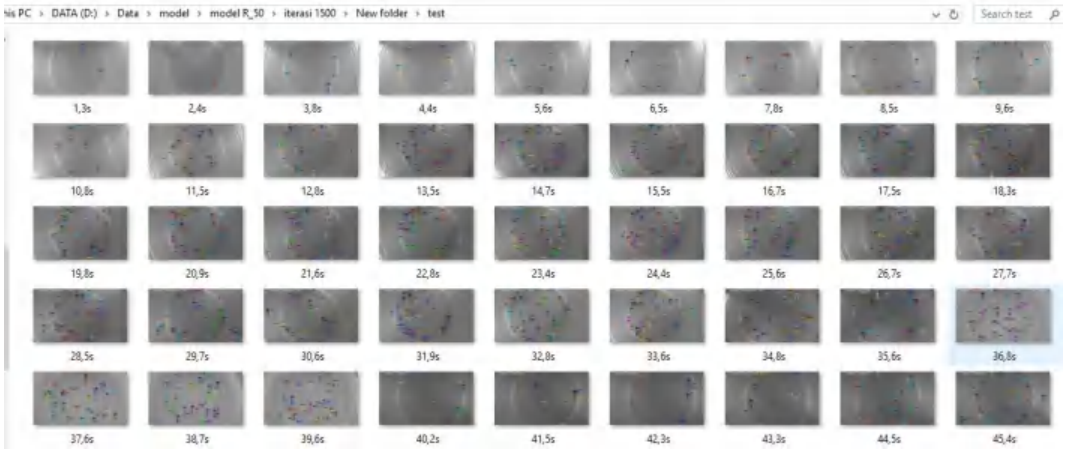
Hasil Pengujian untuk iterasi 500 *Resnet*-50 dengan FPN



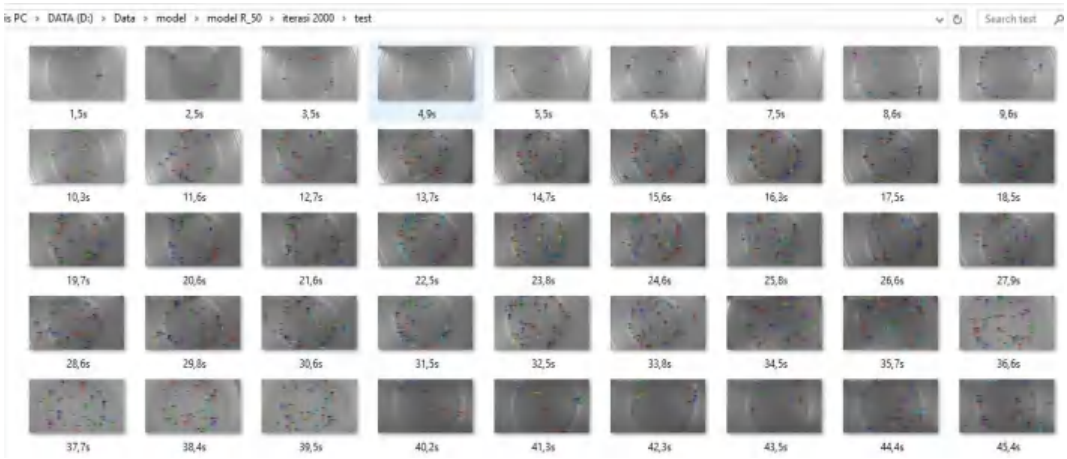
Hasil Pengujian untuk iterasi 1000 *Resnet*-50 dengan FPN



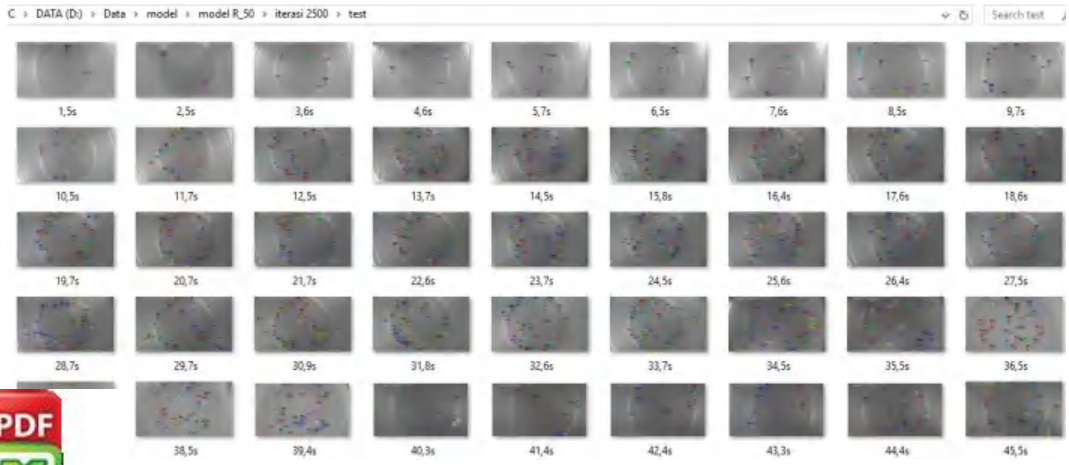
Hasil Pengujian untuk iterasi 1500 *Resnet*-50 dengan FPN



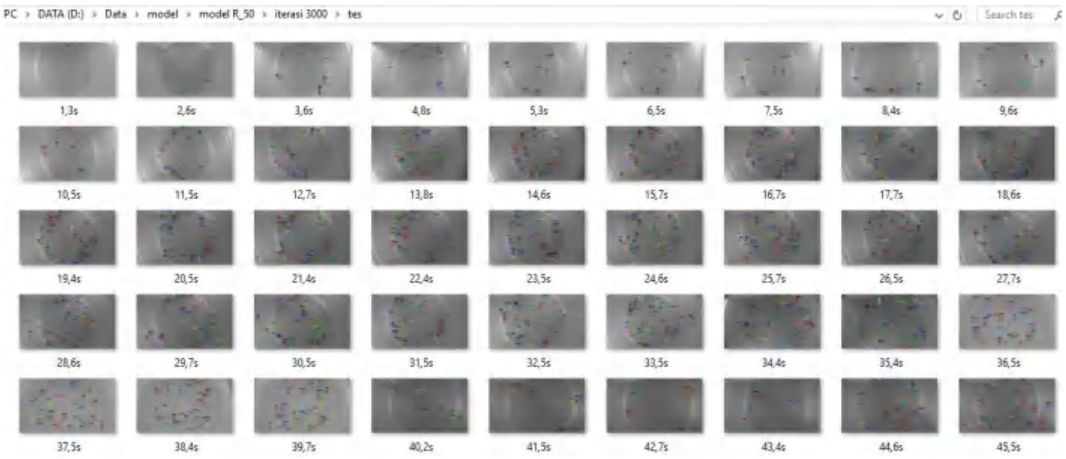
Hasil Pengujian untuk iterasi 2000 *Resnet*-50 dengan FPN



Hasil Pengujian untuk iterasi 2500 *Resnet*-50 dengan FPN

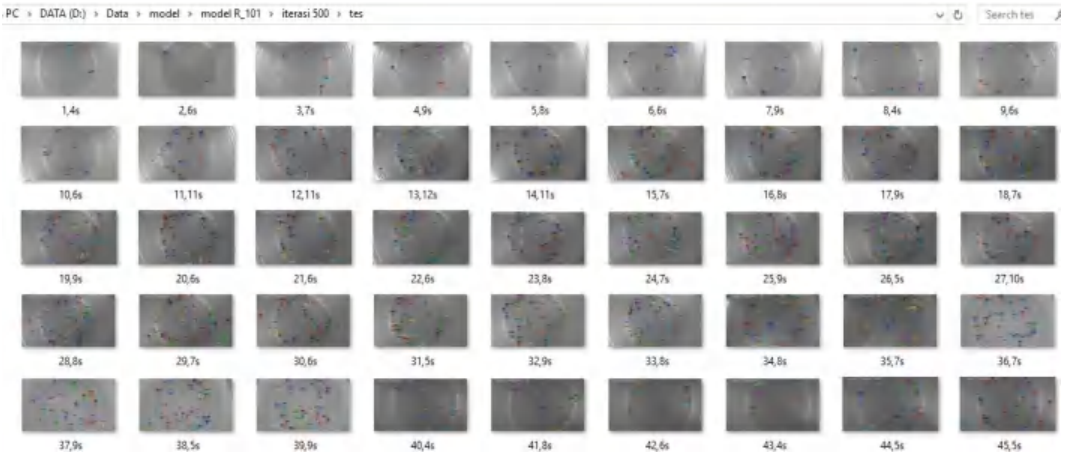


Hasil Pengujian untuk iterasi 3000 *Resnet*-50 dengan FPN

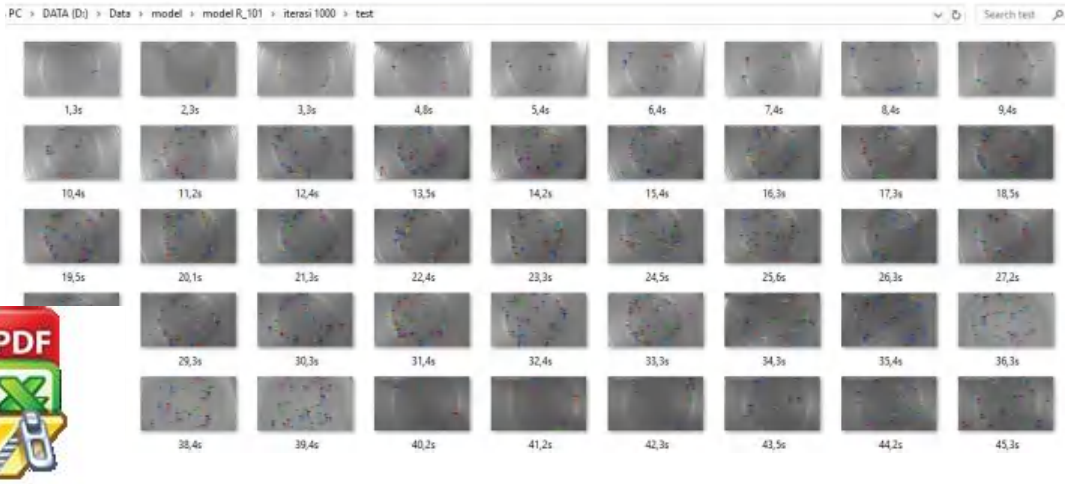


Lampiran 2. Hasil Pengujian *Resnet*-101 dengan FPN

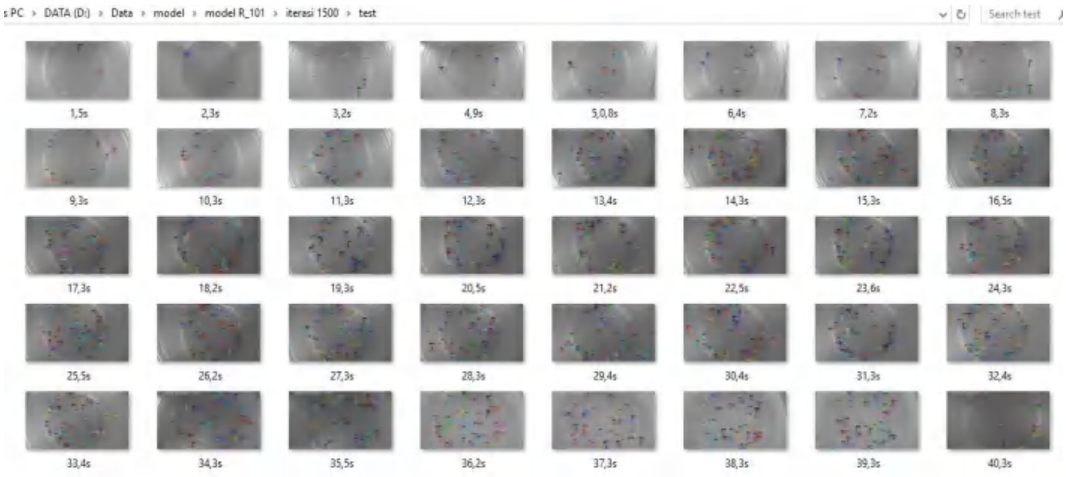
Hasil Pengujian untuk iterasi 500 *Resnet*-101 dengan FPN



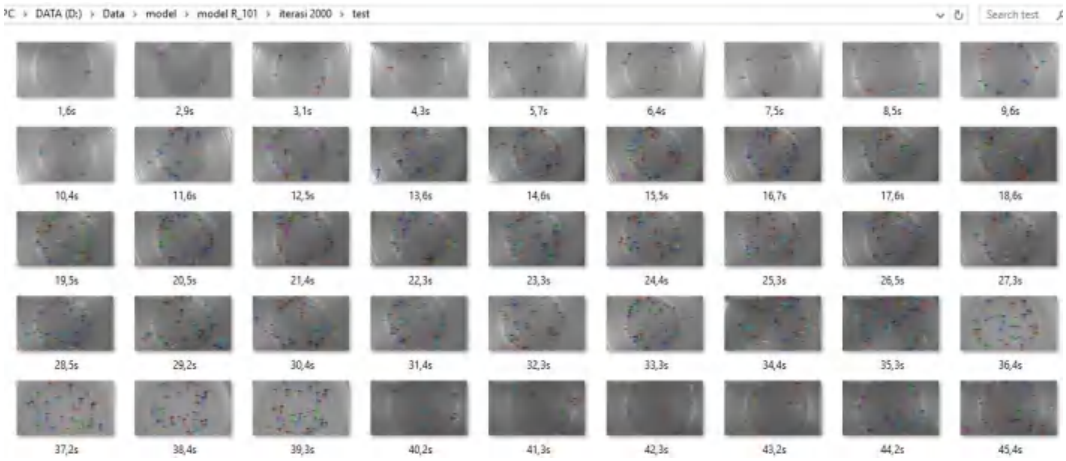
Hasil Pengujian untuk iterasi 1000 *Resnet*-101 dengan FPN



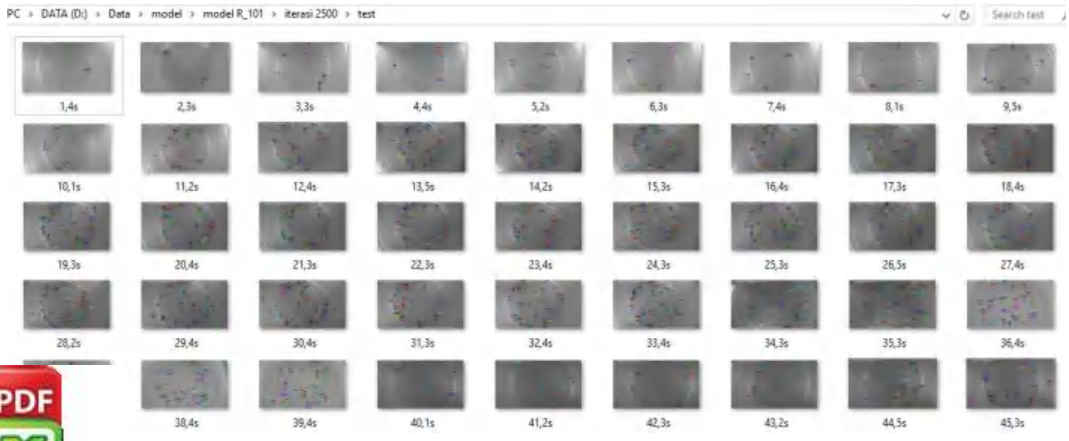
Hasil Pengujian untuk iterasi 1500 *Resnet*-101 dengan FPN



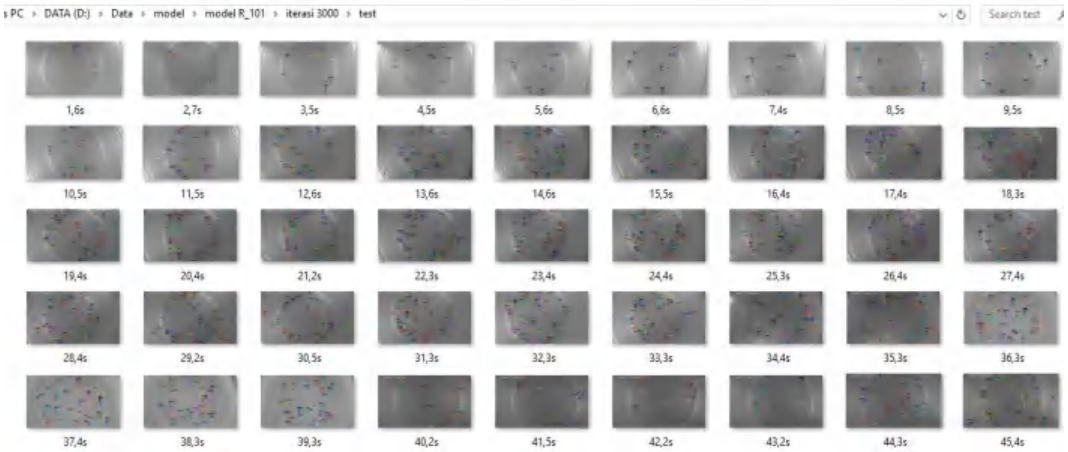
Hasil Pengujian untuk iterasi 2000 *Resnet*-101 dengan FPN



Hasil Pengujian untuk iterasi 2500 *Resnet*-101 dengan FPN



Hasil Pengujian untuk iterasi 3000 *Resnet*-101 dengan FPN



Lampiran 3. Source code

- Source code resnet50 dan resnet101

```
############################################################
#  Resnet Graph
############################################################

# Code adopted from:
# https://github.com/fchollet/deep-learning-
models/blob/master/resnet50.py

def identity_block(input_tensor, kernel_size, filters, stage, block,
                   use_bias=True, train_bn=True):
    """The identity_block is the block that has no conv layer at
shortcut
    # Arguments
        input_tensor: input tensor
        kernel_size: default 3, the kernel size of middle conv layer
at main path
        filters: list of integers, the nb_filters of 3 conv layer at
main path
        stage: integer, current stage label, used for generating
layer names
        block: 'a','b'..., current block label, used for generating
layer names
        use_bias: Boolean. To use or not use a bias in conv layers.
        train_bn: Boolean. Train or freeze Batch Norm layers

    ilter1, nb_filter2, nb_filter3 = filters
    _name_base = 'res' + str(stage) + block + '_branch'
    ame_base = 'bn' + str(stage) + block + '_branch'
```

```
    x = KL.Conv2D(nb_filter1, (1, 1), name=conv_name_base + '2a',
                  use_bias=use_bias)(input_tensor)
    x = BatchNorm(name=bn_name_base + '2a')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter2, (kernel_size, kernel_size),
padding='same',
                  name=conv_name_base + '2b', use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2b')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter3, (1, 1), name=conv_name_base + '2c',
                  use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2c')(x, training=train_bn)

    x = KL.Add()([x, input_tensor])
    x = KL.Activation('relu', name='res' + str(stage) + block +
'_out')(x)
    return x


def conv_block(input_tensor, kernel_size, filters, stage, block,
               strides=(2, 2), use_bias=True, train_bn=True):
    """conv_block is the block that has a conv layer at shortcut
    # Arguments
        input_tensor: input tensor
        kernel_size: default 3, the kernel size of middle conv layer
at main path
        filters: list of integers, the nb_filters of 3 conv layer at
main path
        stage: integer, current stage label, used for generating
layer names
        block: 'a','b'..., current block label, used for generating
layer names
        use_bias: Boolean. To use or not use a bias in conv layers.
        train_bn: Boolean. Train or freeze Batch Norm layers
    Note that from stage 3, the first conv layer at main path is
with subsample=(2,2)
    And the shortcut should have subsample=(2,2) as well
    """
    nb_filter1, nb_filter2, nb_filter3 = filters
    conv_name_base = 'res' + str(stage) + block + '_branch'
    ame_base = 'bn' + str(stage) + block + '_branch'

    KL.Conv2D(nb_filter1, (1, 1), strides=strides,
                  name=conv_name_base + '2a',
=use_bias)(input_tensor)
```

```
    x = BatchNorm(name=bn_name_base + '2a')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter2, (kernel_size, kernel_size),
padding='same',
                  name=conv_name_base + '2b', use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2b')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter3, (1, 1), name=conv_name_base +
                  '2c', use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2c')(x, training=train_bn)

    shortcut = KL.Conv2D(nb_filter3, (1, 1), strides=strides,
                         name=conv_name_base + '1',
use_bias=use_bias)(input_tensor)
    shortcut = BatchNorm(name=bn_name_base + '1')(shortcut,
training=train_bn)

    x = KL.Add()([x, shortcut])
    x = KL.Activation('relu', name='res' + str(stage) + block +
'_out')(x)
    return x
def resnet_graph(input_image, architecture, stage5=False,
train_bn=True):
    """Build a ResNet graph.
        architecture: Can be resnet50 or resnet101
        stage5: Boolean. If False, stage5 of the network is not
created
        train_bn: Boolean. Train or freeze Batch Norm layers
    """
    assert architecture in ["resnet50", "resnet101"]
    # Stage 1
    x = KL.ZeroPadding2D((3, 3))(input_image)
    x = KL.Conv2D(64, (7, 7), strides=(2, 2), name='conv1',
use_bias=True)(x)
    x = BatchNorm(name='bn_conv1')(x, training=train_bn)
    x = KL.Activation('relu')(x)
    C1 = x = KL.MaxPooling2D((3, 3), strides=(2, 2),
padding="same")(x)
    # Stage 2
    x = conv_block(x, 3, [64, 64, 256], stage=2, block='a',
(1, 1), train_bn=train_bn)
    identity_block(x, 3, [64, 64, 256], stage=2, block='b',
=train_bn)
    x = identity_block(x, 3, [64, 64, 256], stage=2, block='c',
=train_bn)
```

```
    # Stage 3
    x = conv_block(x, 3, [128, 128, 512], stage=3, block='a',
train_bn=train_bn)
    x = identity_block(x, 3, [128, 128, 512], stage=3, block='b',
train_bn=train_bn)
    x = identity_block(x, 3, [128, 128, 512], stage=3, block='c',
train_bn=train_bn)
    C3 = x = identity_block(x, 3, [128, 128, 512], stage=3,
block='d', train_bn=train_bn)
    # Stage 4
    x = conv_block(x, 3, [256, 256, 1024], stage=4, block='a',
train_bn=train_bn)
    block_count = {"resnet50": 5, "resnet101": 22}[architecture]
    for i in range(block_count):
        x = identity_block(x, 3, [256, 256, 1024], stage=4,
block=chr(98 + i), train_bn=train_bn)
    C4 = x
    # Stage 5
    if stage5:
        x = conv_block(x, 3, [512, 512, 2048], stage=5, block='a',
train_bn=train_bn)
        x = identity_block(x, 3, [512, 512, 2048], stage=5,
block='b', train_bn=train_bn)
        C5 = x = identity_block(x, 3, [512, 512, 2048], stage=5,
block='c', train_bn=train_bn)
    else:
        C5 = None
    return [C1, C2, C3, C4, C5]
```

- *Source code Counting*

```
#model R50
#larva udang vaname
#jumlah 50
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR,
"/content/drive/MyDrive/Model/model_final_R50(2500).pth")  #
path to the model we just trained
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set a custom
testing threshold
predictor = DefaultPredictor(cfg)

im = cv2.imread("/content/IMG_20220915_161959.jpg")
 = predictor(im)
ualizer(im[:, :, ::-1],
        scale=1,
        instance_mode=ColorMode.IMAGE_BW
```

```
)
out =
v.draw_instance_predictions(outputs["instances"].to("cpu"))
#counting
out_images = out.get_image()
h,w = out_images.shape[:2]
BLACK = (0,0,0)
WHITE = (255,255,255)
font = cv2.FONT_HERSHEY_SIMPLEX
font_size = int(4)
font_color = BLACK
font_thickness = 2
text = 'Jumlah Postlarva : ' + str(len(outputs["instances"]))


x,y = int(0.05 * w), int(0.05 * h)
img_text = cv2.putText(out_images, text, (x,y), font,
font_size, WHITE, font_thickness + 5, cv2.LINE_AA)
img_text = cv2.putText(out_images, text, (x,y), font,
font_size, font_color, font_thickness, cv2.LINE_AA)

im = Image.fromarray(img_text[:, :, ::-1])

im
```

- FPN *(Feature Pyramid Network)*

```
############################################################
#  Feature Pyramid Network Heads
############################################################
def fpn_classifier_graph(rois, feature_maps, image_meta,
                         pool_size, num_classes, train_bn=True,
                         fc_layers_size=1024):
    """Builds the computation graph of the feature pyramid network
classifier
    and regressor heads.

    rois: [batch, num_rois, (y1, x1, y2, x2)] Proposal boxes in
normalized
          coordinates.
    feature_maps: List of feature maps from different layers of the
pyramid.
                  [P2, P3, P4, P5]. Each has a different resolution.
    e_meta: [batch, (meta data)] Image details. See
    image_meta()
```

```
    pool_size: The width of the square feature map generated from
ROI Pooling.
    num_classes: number of classes, which determines the depth of
the results
    train_bn: Boolean. Train or freeze Batch Norm layers
    fc_layers_size: Size of the 2 FC layers

    Returns:
        logits: [batch, num_rois, NUM_CLASSES] classifier logits
(before softmax)
        probs: [batch, num_rois, NUM_CLASSES] classifier
probabilities
        bbox_deltas: [batch, num_rois, NUM_CLASSES, (dy, dx,
log(dh), log(dw))] Deltas to apply to
                     proposal boxes
    """
    # ROI Pooling
    # Shape: [batch, num_rois, POOL_SIZE, POOL_SIZE, channels]
    x = PyramidROIAlign([pool_size, pool_size],
                        name="roi_align_classifier")([rois,
image_meta] + feature_maps)
    # Two 1024 FC layers (implemented with Conv2D for consistency)
    x = KL.TimeDistributed(KL.Conv2D(fc_layers_size, (pool_size,
pool_size), padding="valid"),
                           name="mrcnn_class_conv1")(x)
    x = KL.TimeDistributed(BatchNorm(), name='mrcnn_class_bn1')(x,
training=train_bn)
    x = KL.Activation('relu')(x)
    x = KL.TimeDistributed(KL.Conv2D(fc_layers_size, (1, 1)),
                           name="mrcnn_class_conv2")(x)
    x = KL.TimeDistributed(BatchNorm(), name='mrcnn_class_bn2')(x,
training=train_bn)
    x = KL.Activation('relu')(x)

    shared = KL.Lambda(lambda x: K.squeeze(K.squeeze(x, 3), 2),
                       name="pool_squeeze")(x)

    # Classifier head
    mrcnn_class_logits = KL.TimeDistributed(KL.Dense(num_classes),
                                            name='mrcnn_class_logits
')(shared)
    mrcnn_probs = KL.TimeDistributed(KL.Activation("softmax"),
                                     name="mrcnn_class")(mrcnn_class


    ox head
    atch, num_rois, NUM_CLASSES * (dy, dx, log(dh), log(dw))]
```

```
    x = KL.TimeDistributed(KL.Dense(num_classes * 4,
activation='linear'),
                           name='mrcnn_bbox_fc')(shared)
    # Reshape to [batch, num_rois, NUM_CLASSES, (dy, dx, log(dh),
log(dw))]
    s = K.int_shape(x)
    mrcnn_bbox = KL.Reshape((s[1], num_classes, 4),
name="mrcnn_bbox")(x)

    return mrcnn_class_logits, mrcnn_probs, mrcnn_bbox


def build_fpn_mask_graph(rois, feature_maps, image_meta,
```

- *Mask* R-CNN

```
############################################################
#  MaskRCNN Class
############################################################

class MaskRCNN():
    """Encapsulates the Mask RCNN model functionality.

    The actual Keras model is in the keras_model property.
    """

    def __init__(self, mode, config, model_dir):
        """
        mode: Either "training" or "inference"
        config: A Sub-class of the Config class
        model_dir: Directory to save training logs and trained
weights
        """
        assert mode in ['training', 'inference']
        self.mode = mode
        self.config = config
        self.model_dir = model_dir
        self.set_log_dir()
        self.keras_model = self.build(mode=mode, config=config)

    def build(self, mode, config):
        """Build Mask R-CNN architecture.
            input_shape: The shape of the input image.
            mode: Either "training" or "inference". The inputs and
                outputs of the model differ accordingly.
        """
```

```
        assert mode in ['training', 'inference']

        # Image size must be dividable by 2 multiple times
        h, w = config.IMAGE_SHAPE[:2]
        if h / 2**6 != int(h / 2**6) or w / 2**6 != int(w / 2**6):
            raise Exception("Image size must be dividable by 2 at
least 6 times "
                            "to avoid fractions when downscaling and
upscaling."
                            "For example, use 256, 320, 384, 448,
512, ... etc. ")

        # Inputs
        input_image = KL.Input(
            shape=[None, None, config.IMAGE_SHAPE[2]],
name="input_image")
        input_image_meta = KL.Input(shape=[config.IMAGE_META_SIZE],
                                    name="input_image_meta")
        if mode == "training":
            # RPN GT
            input_rpn_match = KL.Input(
                shape=[None, 1], name="input_rpn_match",
dtype=tf.int32)
            input_rpn_bbox = KL.Input(
                shape=[None, 4], name="input_rpn_bbox",
dtype=tf.float32)

            # Detection GT (class IDs, bounding boxes, and masks)
            # 1. GT Class IDs (zero padded)
            input_gt_class_ids = KL.Input(
                shape=[None], name="input_gt_class_ids",
dtype=tf.int32)
            # 2. GT Boxes in pixels (zero padded)
            # [batch, MAX_GT_INSTANCES, (y1, x1, y2, x2)] in image
coordinates
            input_gt_boxes = KL.Input(
                shape=[None, 4], name="input_gt_boxes",
dtype=tf.float32)
            # Normalize coordinates
            gt_boxes = KL.Lambda(lambda x: norm_boxes_graph(
                x, K.shape(input_image)[1:3]))(input_gt_boxes)
            # 3. GT Masks (zero padded)
            # [batch, height, width, MAX_GT_INSTANCES]
            if config.USE_MINI_MASK:
                input_gt_masks = KL.Input(
                    shape=[config.MINI_MASK_SHAPE[0],
                           config.MINI_MASK_SHAPE[1], None],
```

```
                    name="input_gt_masks", dtype=bool)
            else:
                input_gt_masks = KL.Input(
                    shape=[config.IMAGE_SHAPE[0],
config.IMAGE_SHAPE[1], None],
                    name="input_gt_masks", dtype=bool)
        elif mode == "inference":
            # Anchors in normalized coordinates
            input_anchors = KL.Input(shape=[None, 4],
name="input_anchors")

        # Build the shared convolutional layers.
        # Bottom-up Layers
        # Returns a list of the last layers of each stage, 5 in
total.
        # Don't create the thead (stage 5), so we pick the 4th item
in the list.
        if callable(config.BACKBONE):
            _, C2, C3, C4, C5 = config.BACKBONE(input_image,
stage5=True,
                                                train_bn=config.TRAI
N_BN)
        else:
            _, C2, C3, C4, C5 = resnet_graph(input_image,
config.BACKBONE,
                                             stage5=True,
train_bn=config.TRAIN_BN)
        # Top-down Layers
        # TODO: add assert to varify feature map sizes match what's
in config
        P5 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1),
name='fpn_c5p5')(C5)
        P4 = KL.Add(name="fpn_p4add")([
            KL.UpSampling2D(size=(2, 2),
name="fpn_p5upsampled")(P5),
            KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1),
name='fpn_c4p4')(C4)])
        P3 = KL.Add(name="fpn_p3add")([
            KL.UpSampling2D(size=(2, 2),
name="fpn_p4upsampled")(P4),
            KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1),
name='fpn_c3p3')(C3)])
        P2 = KL.Add(name="fpn_p2add")([
            KL.UpSampling2D(size=(2, 2),
n_p3upsampled")(P3),
            KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (1, 1),
n_c2p2')(C2)])
```

```
        # Attach 3x3 conv to all P layers to get the final feature
maps.
        P2 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3),
padding="SAME", name="fpn_p2")(P2)
        P3 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3),
padding="SAME", name="fpn_p3")(P3)
        P4 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3),
padding="SAME", name="fpn_p4")(P4)
        P5 = KL.Conv2D(config.TOP_DOWN_PYRAMID_SIZE, (3, 3),
padding="SAME", name="fpn_p5")(P5)
        # P6 is used for the 5th anchor scale in RPN. Generated by
        # subsampling from P5 with stride of 2.
        P6 = KL.MaxPooling2D(pool_size=(1, 1), strides=2,
name="fpn_p6")(P5)

        # Note that P6 is used in RPN, but not in the classifier
heads.
        rpn_feature_maps = [P2, P3, P4, P5, P6]
        mrcnn_feature_maps = [P2, P3, P4, P5]

        # Anchors
        if mode == "training":
            anchors = self.get_anchors(config.IMAGE_SHAPE)
            # Duplicate across the batch dimension because Keras
requires it
            # TODO: can this be optimized to avoid duplicating the
anchors?
            anchors = np.broadcast_to(anchors, (config.BATCH_SIZE,)
+ anchors.shape)
            # A hack to get around Keras's bad support for constants
            anchors = KL.Lambda(lambda x: tf.Variable(anchors),
name="anchors")(input_image)
        else:
            anchors = input_anchors

        # RPN Model
        rpn = build_rpn_model(config.RPN_ANCHOR_STRIDE,
                              len(config.RPN_ANCHOR_RATIOS),
config.TOP_DOWN_PYRAMID_SIZE)
        # Loop through pyramid layers
        layer_outputs = []  # list of lists
        for p in rpn_feature_maps:
            layer_outputs.append(rpn([p]))
        # Concatenate layer outputs
        # Convert from list of lists of level outputs to list of

        # of outputs across levels.
```

```
        # e.g. [[a1, b1, c1], [a2, b2, c2]] => [[a1, a2], [b1, b2],
[c1, c2]]
        output_names = ["rpn_class_logits", "rpn_class", "rpn_bbox"]
        outputs = list(zip(*layer_outputs))
        outputs = [KL.Concatenate(axis=1, name=n)(list(o))
                   for o, n in zip(outputs, output_names)]

        rpn_class_logits, rpn_class, rpn_bbox = outputs

        # Generate proposals
        # Proposals are [batch, N, (y1, x1, y2, x2)] in normalized
coordinates
        # and zero padded.
        proposal_count = config.POST_NMS_ROIS_TRAINING if mode ==
"training"\
            else config.POST_NMS_ROIS_INFERENCE
        rpn_rois = ProposalLayer(
            proposal_count=proposal_count,
            nms_threshold=config.RPN_NMS_THRESHOLD,
            name="ROI",
            config=config)([rpn_class, rpn_bbox, anchors])

        if mode == "training":
            # Class ID mask to mark class IDs supported by the
dataset the image
            # came from.
            active_class_ids = KL.Lambda(
                lambda x:
parse_image_meta_graph(x)["active_class_ids"]
                )(input_image_meta)

            if not config.USE_RPN_ROIS:
                # Ignore predicted ROIs and use ROIs provided as an
input.
                input_rois =
KL.Input(shape=[config.POST_NMS_ROIS_TRAINING, 4],
                                         name="input_roi",
dtype=np.int32)
                # Normalize coordinates
                target_rois = KL.Lambda(lambda x: norm_boxes_graph(
                    x, K.shape(input_image)[1:3]))(input_rois)
            else:
                target_rois = rpn_rois

            # Generate detection targets
            # Subsamples proposals and generates target outputs for
```

```
            # Note that proposal class IDs, gt_boxes, and gt_masks
are zero
            # padded. Equally, returned rois and targets are zero
padded.
            rois, target_class_ids, target_bbox, target_mask =\
                DetectionTargetLayer(config,
name="proposal_targets")([
                    target_rois, input_gt_class_ids, gt_boxes,
input_gt_masks])

            # Network Heads
            # TODO: verify that this handles zero padded ROIs
            mrcnn_class_logits, mrcnn_class, mrcnn_bbox =\
                fpn_classifier_graph(rois, mrcnn_feature_maps,
input_image_meta,
                                     config.POOL_SIZE,
config.NUM_CLASSES,
                                     train_bn=config.TRAIN_BN,
                                     fc_layers_size=config.FPN_CLASS
IF_FC_LAYERS_SIZE)

            mrcnn_mask = build_fpn_mask_graph(rois,
mrcnn_feature_maps,
                                              input_image_meta,
                                              config.MASK_POOL_SIZE,
                                              config.NUM_CLASSES,
                                              train_bn=config.TRAIN_
BN)


    def find_trainable_layer(self, layer):
        """If a layer is encapsulated by another layer, this
function
        digs through the encapsulation and returns the layer that
holds
        the weights.
        """
        if layer.__class__.__name__ == 'TimeDistributed':
            return self.find_trainable_layer(layer.layer)
        return layer
dst....
```

LEMBAR PERBAIKAN SKRIPSI

"SISTEM PENGHITUNG JUMLAH POSTLARVA UDANG VANAME BERBASIS IMAGES MENGGUNAKAN MASK RCNN"

OLEH:

MUHAMMAD GHALIB S.M
D12117005

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 15 Juli 2024.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Dr. Ir. Zahir Zainuddin, M.Sc.	
Sekretaris	Dr. Eng. Ir. Muhammad Niswar, S.T., M.IT.	
Anggota	Dr. Amil Ahmad Ilham, S.T., M.IT.	
	Prof. Dr. Eng. Intan Sari Areni, S.T., M.T.	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Dr. Ir. Zahir Zainuddin, M.Sc.	
II	Dr. Eng. Ir. Muhammad Niswar, S.T., M.IT.	