BAB IV
HASIL DAN PEMBAHASAN

4.1 Implementasi

Semua tahapan perancangan menggunakan *flowchart* seperti pada Gambar 3. 4. diimplementasikan dalam kode program dengan Bahasa pemrograman Python versi 3.7. langkah-langkah dan kode program untuk setiap proses yang ada pada sistem akan dijelaskan pada hasil dan pembahasan ini.

4.1.1 Support vector machine (SVM)

Semua tahapan penerapan metode SVM pada program yang telah dirancangan dengan *flowchart* seperti pada **Error! Reference source not found.** diimplementasikan dalam kode rogram dengan Bahasa pemrograman Python versi 3.7. Berikut adalah tahapan implementasi pada kode program:

A. *Import packages* yang dibutuhkan

```
    import os
    import math
    import glob
    import time
    import cv2
    from collections import deque
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg

    from skimage.feature import hog
    from   skimage.color.adapt_rgb   import   adapt_rgb,   each_channel,
hsv_value
    from skimage import filters

    from sklearn.model_selection import train_test_split
    from sklearn import svm
    from sklearn.preprocessing import StandardScaler
    from sklearn import tree

    from scipy.ndimage.measurements import label

    %matplotlib inline
```

Gambar 4. 1. *Import requi*red *packages*

Untuk menggunakan modul-modul yang ada dalam *package* atau *library* pada Python, terlebih dahulu modul dan *package* harus diimpor di awal program seperti pada Gambar 4. 1.

Library Scikit-*Image* (Skimage) menyediakan modul HOG yang akan digunakan untuk mengekstrasi fitur di tahap selanjutnya. *Library* Sklearn menyediakan model yang digunakan untuk *training* data dengan metode SVM.

B. Fungsi untuk ekstrak fitur

Gambar 4. 2. merupakan pendefinisian fungsi untuk tiga jenis ektraksi fitur yang digunakan, yaitu 1) raw color dengan fungsi bins_spatial, 2) color histogram dengan fungsi color_hist, 3) dan histogram of gradients (HOG) dengan fungsi get_hog_features yang telah tersedia di Scikit-Image. Karena citra yang digunakan adalah citra warna atau RGB maka pada perhitungan dilakukan dengan memisahkan warna pada 3 *channel* berbeda.

```
    def bin_spatial(img, size=(32, 32)):
        """
            Return the image color bins
        """
        color1 = cv2.resize(img[:,:,0], size).ravel()
        color2 = cv2.resize(img[:,:,1], size).ravel()
        color3 = cv2.resize(img[:,:,2], size).ravel()
        return np.hstack((color1, color2, color3))

    def color_hist(img, nbins=32):
        """
            Return all channel histogram.
        """
        # Compute the histogram of the color channels separately
        channel1_hist = np.histogram(img[:,:,0], bins=nbins)
        channel2_hist = np.histogram(img[:,:,1], bins=nbins)
        channel3_hist = np.histogram(img[:,:,2], bins=nbins)

        return       np.concatenate((channel1_hist[0],       channel2_hist[0],
channel3_hist[0]))


    def          get_hog_features(img,          orient,          pix_per_cell,
cell_per_block,feature_vector=True):
        """
            Return a histogram of oriented gradients using skimage.
        """
        return  hog(img,  orientations=orient,  pixels_per_cell=(pix_per_cell,
pix_per_cell),
                          cells_per_block=(cell_per_block,   cell_per_block),
transform_sqrt=True,
                          visualize=False, feature_vector=feature_vector)
```

Gambar 4. 2. Fungsi untuk ekstrak fitur

C. Fungsi wrapper

Seperti namanya wrapper function digunakan untuk membungkus, dengan kata lain 3 fitur yang berbeda dibungkus atau dikombinasikan dengan hanya memanggil satu fitur list yaitu features.append seperti pada Gambar 4. 3.

```
    def extract_features(imgs, colorConv, orient=9,
         pix_per_cell=8, cell_per_block=2, hog_channel=0):
        """
         Wrapper function to return a bag of features by combining
different features extracted with above functions.
        """
        # Create a list to append feature vectors to
        features = []
        # Iterate through the list of images and extract features
        for file in imgs:
            image = mpimg.imread(file)
            feature_image = convert_color(image, colorConv)
            spatial_features = bin_spatial(feature_image)
            hist_features = color_hist(feature_image)
            if hog_channel == 'ALL':
                hog_features = []
                for channel in range(feature_image.shape[2]):

hog_features.append(get_hog_features(feature_image[:,:,channel],orient,
pix_per_cell, cell_per_block))
            hog_features = np.ravel(hog_features)
            else:
                hog_features                                          =
get_hog_features(feature_image[:,:,hog_channel],  orient,  pix_per_cell,
cell_per_block)
            # Append the new feature vector to the features list
            features.append(np.concatenate((spatial_features,
hist_features, hog_features)))
    return features
```

Gambar 4. 3. fungsi untuk mengkombinasikan 3 fitur berbeda

D. Fungsi untuk train dan test *classifier*

Untuk *training* dan *testing classifier* maka dapat didefinisikan 3 fungsi yang berbeda seperti pada Gambar 4. 4. Fungsi time digunakan untuk menampilkan waktu yang dibutuhkan selama proses berlangsung, baik pada *training* maupun *testing*.

```
    def train_SVC(X_train, y_train):
        """
            Function to train an svm.
        """
        svc = svm.LinearSVC()
        # Check the training time for the SVC
        t=time.time()
        svc.fit(X_train, y_train)
        t2 = time.time()
        print(round(t2-t, 2), 'Seconds to train SVC...')
        return svc


    def train_dtree(X_train, y_train):
        """
            Function to train a decision tree.
        """
        clf = tree.DecisionTreeClassifier()
        t=time.time()
        clf = clf.fit(X_train, y_train)
        t=time.time()
        print(round(t2-t, 2), 'Seconds to train dtree...')
        return clf


    def test_classifier(svc, X_test, y_test):
        """
           Funtion to test the classifier.
        """
        print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
        # Check the prediction time for a single sample
        t=time.time()
        n_predict = 10
        pred = svc.predict(X_test[0:n_predict])
        actual = y_test[0:n_predict]
        print('My SVC predicts: ', pred)
        print('For these',n_predict, 'labels: ', actual)
        t2 = time.time()
        print(round(t2-t,  5),  'Seconds  to  predict',  n_predict,'labels  with
SVC')
```

Gambar 4. 4. Fungsi untuk *training* dan *testing classifier*

E. Fungsi untuk menghilangkan duplikasi hasil deteksi

Fungsi pada Gambar 4. 5. merupakan iterasi yang yang mendeteksi objek kapal dan menambahnya dengan satu hasil deteksi yang paling tinggi. Deteksi yang dipertahankan adalah yang memiliki piksel paling kecil.

```
    def add_heat(heatmap, bbox_list):

        """
            Iterate the windows with detected ships and enhance the once
with highest detections.
        """
        for box in bbox_list:
            # Add += 1 for all pixels inside each bbox
            heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1
        return heatmap


    def apply_threshold(heatmap, threshold):
        """
            Only keep the detections that have a minimum number of pixels.
        """
        # Zero out pixels below the threshold
        heatmap[heatmap < threshold] = 0
        return heatmap


    def draw_labeled_bboxes(img, labels):
        """
            Draw the boxes on the detected ships
        """
        for i in range(1, labels[1]+1):
            # Find pixels with each ship label value
            nonzero = (labels[0] == i).nonzero()
            # Identify x and y values of those pixels
            nonzeroy = np.array(nonzero[0])
            nonzerox = np.array(nonzero[1])
            # Define a bounding box based on min/max x and y
            bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox),
np.max(nonzeroy)))
            cv2.rectangle(img, bbox[0], bbox[1], (255,255,255), 6)
        return img
```

Gambar 4. 5. Fungsi untuk menghilangkan duplikasi hasil deteksi

F. Fungsi untuk mendeteksi kapal menggunakan *classifier*

Fungsi pada Gambar 4. 6 mengambil gambar, mengekstrak fitur dari wilayah yang diinginkan dan menjalankan prediksi, lalu mengembalikan ke koordinat tempat kapal terdeteksi. Ukuran *window* sama dengan ukuran citra pada train dan test.

```
    def find_ships(img, colorConv, svc, X_scaler, orient, pix_per_cell,
cell_per_block):
        img = img.astype(np.float32)/255
        img_shape = img.shape
            img = convert_color(img, colorConv)
        nxblocks = (img.shape[1] // pix_per_cell)-1
        nyblocks = (img.shape[0] // pix_per_cell)-1
        nfeat_per_block = orient*cell_per_block**2
        window = 80
        nblocks_per_window = (window // pix_per_cell)-1
        cells_per_step = 2
        nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
        nysteps = (nyblocks - nblocks_per_window) // cells_per_step
        hog_ch1   =   get_hog_features(img[:,:,0],   orient,   pix_per_cell,
cell_per_block, feature_vector=False)
        hog_ch2   =   get_hog_features(img[:,:,1],   orient,   pix_per_cell,
cell_per_block, feature_vector=False)
        hog_ch3   =   get_hog_features(img[:,:,2],   orient,   pix_per_cell,
cell_per_block, feature_vector=False)
        on_windows = []
        for xb in range(nxsteps):
            for yb in range(nysteps):
                ypos = yb*cells_per_step
                xpos = xb*cells_per_step
                # Extract HOG for this patch
                hog_feat1      =      hog_ch1[ypos:ypos+nblocks_per_window,
xpos:xpos+nblocks_per_window].ravel()
                hog_feat2      =      hog_ch2[ypos:ypos+nblocks_per_window,
xpos:xpos+nblocks_per_window].ravel()
                hog_feat3      =      hog_ch3[ypos:ypos+nblocks_per_window,
xpos:xpos+nblocks_per_window].ravel()
                hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))
                xleft = xpos*pix_per_cell
                ytop = ypos*pix_per_cell
                # Extract the image patch
       subimg=cv2.resize(img[ytop:ytop+window,left:xleft+window], (80,80))
                # Get color features
                spatial_features = bin_spatial(subimg)
                # Get histogram feature
                hist_features = color_hist(subimg)
                # add all features and Scale them
                test_features                                            =
X_scaler.transform(np.hstack((spatial_features,              hist_features,
hog_features)).reshape(1, -1))
                # make a prediction
                test_prediction = svc.predict(test_features)
                # Add to list of windows if ship predicted
                if test_prediction == 1:
                    xbox_left = np.int(xleft)
                    ytop_draw = np.int(ytop)
                    win_draw = np.int(window)
                    on_windows.append(((xbox_left,
ytop_draw),(xbox_left+win_draw,ytop_draw+win_draw)))
        return on_windows
```

Gambar 4. 6. Kode untuk mendeteksi kapal menggunakan *classifier*

G. Set-up *training* dan *validation*

Menyiapkan data untuk *training classifier*. Data akan diacak dan dibagi ke dalam *training* set dan *validation* set. Perbandingan pembagian data adalah 80% atau 3200 citra untuk *training* dan 20% atau 800 citra untuk *validation*. Jika kode pada Gambar 4. 7 dijalankan maka akan muncul output Gambar 4. 8.

```
    def setup_train_data(colorConv, orient, pix_per_cell, cell_per_block,
hog_channel):

        ships = []
        images = glob.glob('C:/Users/16523150/Desktop/object-detection-
with-svm-and-opencv-master/positive/*.png', recursive=True)
        for image in images:
            ships.append(image)

        images = glob.glob('C:/Users/16523150/Desktop/object-detection-
with-svm-and-opencv-master/negative/*.png', recursive=True)
        notships = []
        for image in images:
            notships.append(image)

        ship_features = extract_features(ships, colorConv, orient=orient,
                                         pix_per_cell=pix_per_cell,
cell_per_block=cell_per_block,
                                         hog_channel=hog_channel)
        notship_features = extract_features(notships, colorConv,
orient=orient,
                                    pix_per_cell=pix_per_cell,
cell_per_block=cell_per_block,
                                    hog_channel=hog_channel)
        # Create an array stack of feature vectors
        X = np.vstack((ship_features, notship_features)).astype(np.float64)
        # Fit a per-column scaler
        X_scaler = StandardScaler().fit(X)
        # Apply the scaler to X
        scaled_X = X_scaler.transform(X)
        # Define the labels vector
        y_train = np.hstack((np.ones(len(ship_features)),
np.zeros(len(notship_features))))
        # shuffle the data
        #X_train, y_train = shuffle(scaled_X, y_train)
        X_train, X_test, y_train, y_test = train_test_split(scaled_X,
y_train, test_size=0.2, random_state=2)
        return X_train, X_test, y_train, y_test, X_scaler

    print('Preparing training data...')
    X_train, X_test, y_train, y_test, X_scaler =
setup_train_data(colorConv, orient,

pix_per_cell, cell_per_block, hog_channel)
    print("Number of training examples =", len(X_train))
    print("Number of testing examples =", len(X_test))
```

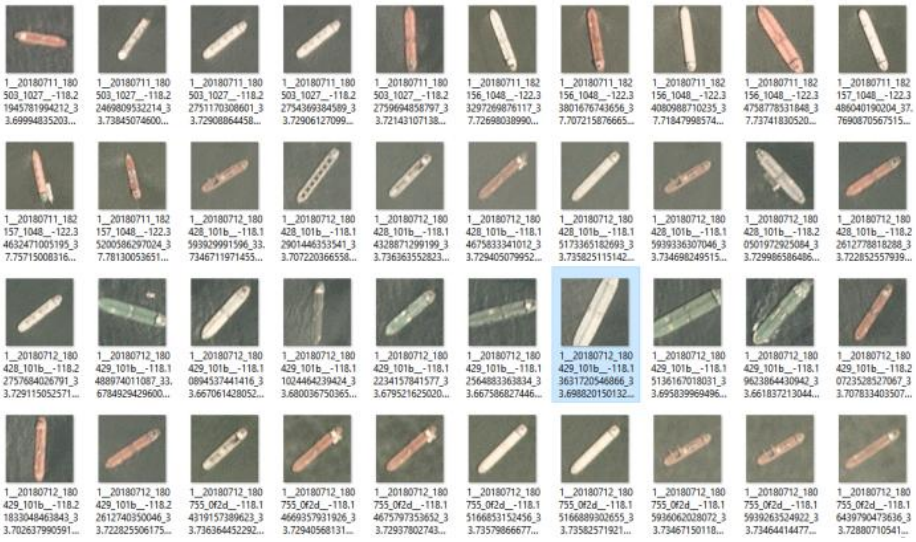
Gambar 4. 7. Set-up data

```
Preparing training data...
Number of training examples = 3200
Number of testing examples = 800
```

Gambar 4. 8. menyiapkan data

H. *Training* dan *Testing classifier*

Kode program pada Gambar 4. 10 digunakan untuk *training classifier* yang sebelumnya telah didefiniskan fungsinya. Pada Gambar 4. 11 dapat diketahui bahwa waktu yang dibutuhkan untuk melatih *classifier* adalah 5.22 detik. Kemudian *classifier* diuji dengan citra sebanyak 800 seperti pada gambar Gambar 4. 9 menggunakan kode program seperti pada Gambar 4. 12 dan hasilnya kita bisa lihat pada Gambar 4. 13. Berdasarkan Gambar 4. 13 model memiliki akurasi 94,38% untuk data valiation.



Gambar 4. 9. Contoh citra yang digunakan untuk validasi

```
print('Training Classifier...')
svc = train_SVC(X_train, y_train)
#clf = train_dtree(X_train, y_train)
```

Gambar 4. 10. *Training classifier* code

```
In [16]: print('Training Classifier...')
         svc = train_SVC(X_train, y_train)
         #clf = train_dtree(X_train, y_train)

         Training Classifier...
         5.22 Seconds to train SVC...
```

Gambar 4. 11. waktu yang dibutuhkan untuk *training classifier*

```
print('Testing Classifier...')
test_classifier(svc, X_test, y_test)
#test_classifier(clf, X_test, y_test)
```

Gambar 4. 12. *Testing classifier* pada data *validation*

```
#test_classifier(clf, X_test, y_test)

Testing Classifier...
Test Accuracy of SVC =  0.9438
My SVC predicts:  [0. 1. 0. 1. 0. 0. 1. 0. 1. 1.]
For these 10 labels:  [0. 1. 0. 0. 0. 0. 1. 0. 0. 1.]
0.00401 Seconds to predict 10 labels with SVC
```

Gambar 4. 13. Hasil pengujian pada data *validation*

I. Fungsi wrapper untuk memproses hasil

Wrapper function pada Gambar 4. 14 digunakan untuk menjalankan semua fungsi yang telah didefinisikan sebelumnya.

```
def process_img(img, threshhold=1.5, show_stages=False):
    # get the windows where the classifier predicts ship
    hot_windows = find_ships(img, colorConv, svc, X_scaler, orient,
pix_per_cell, cell_per_block)
    if show_stages:
        img1 = np.copy(img)
        for bbox in hot_windows:
            cv2.rectangle(img1, bbox[0], bbox[1], (0,0,255), 6)
        plot_img(img1, show_stages, "All detections")

    # Highlight the windows
    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    heat = add_heat(heat,hot_windows)
    plot_img(heat, show_stages, "After Applying heat")

    # Append the detections to detections from last n frames
    recent_heatmaps.append(heat)

    # Take the mean of last n frames as discard the windows that are
below the threshold
    heatmap = apply_threshold(np.mean(recent_heatmaps,
axis=0),threshhold)
    plot_img(heatmap, show_stages, "After threshold")

    # Add labels to remaning detections
    labels = label(heatmap)
    # Draw boxes on the ships and return the image
    return draw_labeled_bboxes(np.copy(img), labels)
```

Gambar 4. 14. Kode untuk menjalankan semua fungsi.

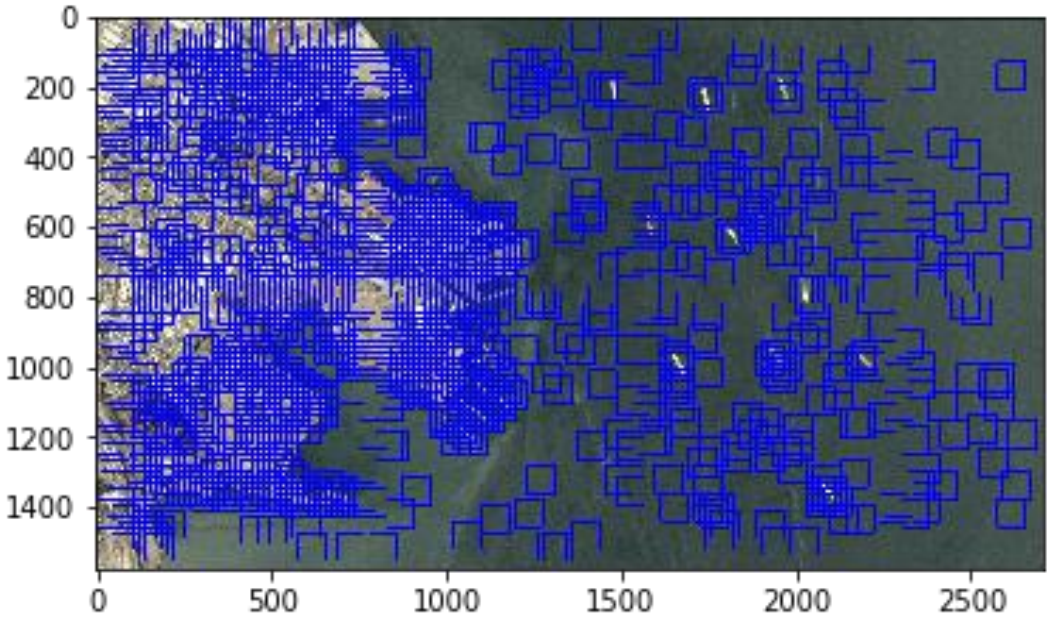
J. Pengujian pada gambar beresolusi tinggi

Kode pada Gambar 4. 15 digunakan untuk menguji *classifier* pada citra satelit beresolusi tinggi. Hasil deteksi pertama seperti pada Gambar 4. 16 merupakan hasil deteksi dengan banyak duplikasi, oleh karena itu duplikasi dihilangkan dengan kode pada Gambar 4. 5 sehingga menghasilkan deteksi seperti pada

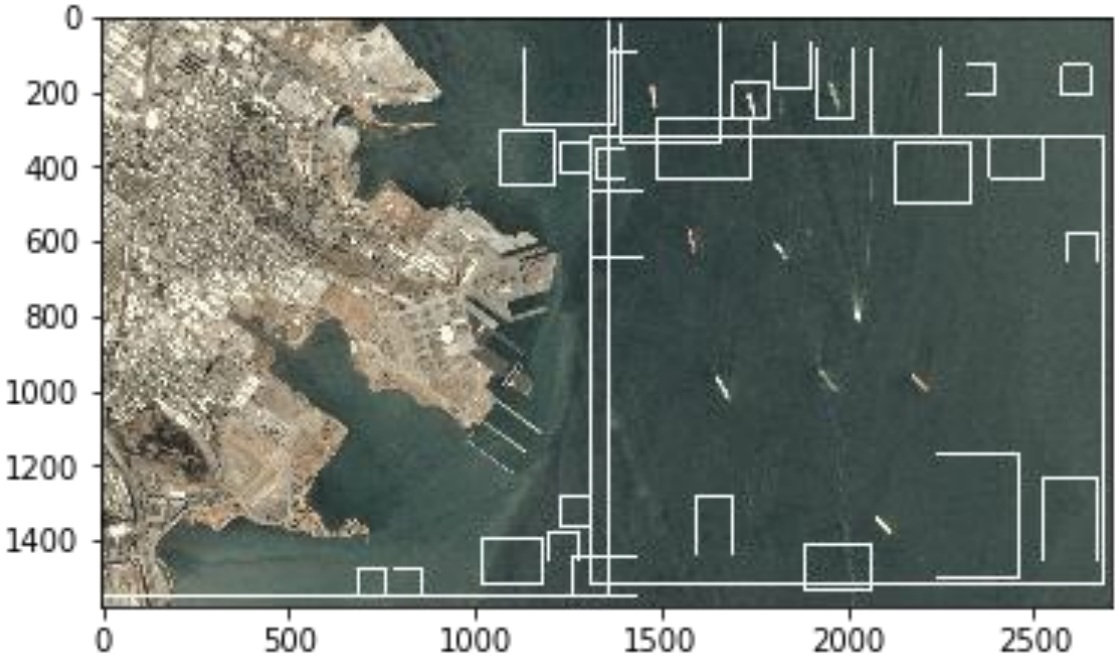
Gambar 4. 17. Karena menggabungkan lebih dari satu windows seperti dijelaskan pada tahap penghilangan duplikasi sebelumnya, menyebabkan hasil deteksi atau hasil penandaan lokasi objek menjadi meluas.

```
test_dir = "scenes/"
test_images = glob.glob(test_dir+'sfbay_3.png')
for test_image in test_images:
    print()
    print("-----------------"+test_image+"----------------------")
    recent_heatmaps = deque(maxlen=10)
    img = mpimg.imread(test_image)
    out_img = process_img(img, 1, True)
    plot_img(out_img, True, "Final Result")
    print()
```

Gambar 4. 15. Kode untuk menguji pada citra beresolusi tinggi



Gambar 4. 16. Hasil deteksi awal



Gambar 4. 17. Hasil deteksi dengan menghilangkan duplikasi

4.1.2 Convolution neural networks

Tahapan perancangan menggunakan *flowchart* seperti pada Gambar 3. 5 dan Gambar 3. 6 diimplementasikan dalam kode program dengan Bahasa pemrograman Python versi 3.7. Berikut adalah tahapan implementasi pada kode program :

A. *Import packages* yang dibutuhkan

Untuk menggunakan modul-modul yang ada dalam *package* atau *library* pada Python, terlebih dahulu modul dan *package* harus diimpor di awal program seperti pada Gambar 4. 18. *Library* TensorFlow2 menyediakan modul Keras yang akan digunakan untuk membangun model Jaringan pada CNN.

```
import json, sys, random
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Flatten, Activation
from keras.layers import Dropout
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.optimizers import SGD
import keras.callbacks
from PIL import Image, ImageDraw
from matplotlib import pyplot as plt
```

Gambar 4. 18. *Import* modul dan *packages*

B. Mengakses data set

Kode pada Gambar 4. 19 digunakan untuk mengakses data set berupa *file* berbentuk JSON yang berisi label dan lokasi pada data set citra yang sama seperti yang digunakan pada metode SVM.

```
f = open(r'../ships-in-satellite-imagery/shipsnet.json')
dataset = json.load(f)
f.close()

input_data = np.array(dataset['data']).astype('uint8')
output_data = np.array(dataset['labels']).astype('uint8')
```

Gambar 4. 19. Kode untuk membuka data set

C. *Reshape* data

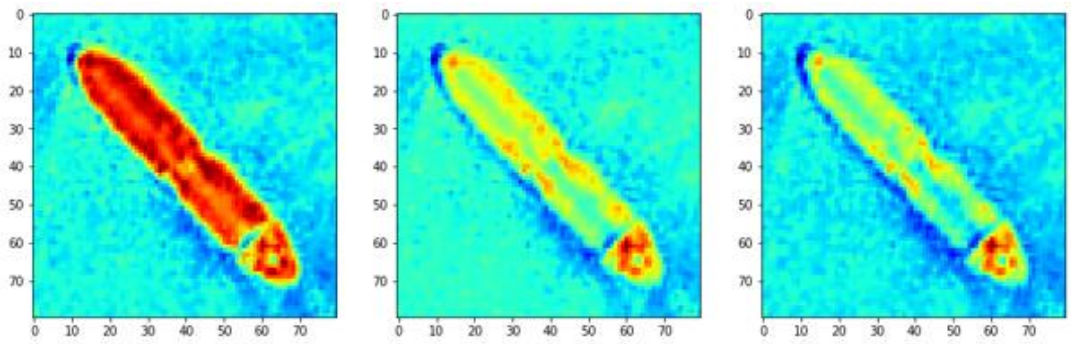
Karena input data berupa array dalam satu *channel* maka data perlu diubah bentuknya menjadi 3 *channel*, yaitu *red*, *green*, dan *blue* seperti pada Gambar 4. 20. Dengan kode pada Gambar 4. 21 maka dapat dilihat Gambar 4. 22 citra yang telah diubah dalam 3 *channel* berbeda karena citra yang digunakan merupakan citra warna RGB.

```
n_spectrum = 3
weight = 80
height = 80
X = input_data.reshape([-1, n_spectrum, weight, height])
X[0].shape

pic = X[3]

red_spectrum = pic[0]
green_spectrum = pic[1]
blue_spectrum = pic[2]
```

Gambar 4. 20. Kode untuk *reshape* data

```
plt.figure(2, figsize = (5*3, 5*1))
plt.set_cmap('jet')
plt.subplot(1, 3, 1)
plt.imshow(red_spectrum)
plt.subplot(1, 3, 2)
plt.imshow(green_spectrum)
plt.subplot(1, 3, 3)
plt.imshow(blue_spectrum)
plt.show()
```

Gambar 4. 21. Kode menampilkan data yang telah diubah menjadi 3 *channel*



Gambar 4. 22. Data yang telah diubah menjadi 3 *channel*

D. Membangun jaringan

Pada Gambar 4. 23 jaringan syaraf tiruan dibangun dengan menggunakan data random. Karena citra yang digunakan adalah tipe uint8 dengan nilai dalam interval [0, 255] dan kita ingin memuat nilai antara 0 dan 1, maka perlu dinormalisasi terlebih dahulu.

```
    indexes = np.arange(4000)
    np.random.shuffle(indexes)

    X_train = X[indexes].transpose([0,2,3,1])
    y_train = y[indexes]

    # normalization
    X_train = X_train / 255 # images are type uint8 with values in the
[0, 255] interval and we would like to contain values between 0 and 1


    np.random.seed(42)

    # network design
    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same', input_shape=(80, 80, 3),
activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2))) #40x40
    model.add(Dropout(0.25))

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2))) #20x20
    model.add(Dropout(0.25))

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2))) #10x10
    model.add(Dropout(0.25))

    model.add(Conv2D(32, (10, 10), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2))) #5x5
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(2, activation='softmax'))
```

Gambar 4. 23. Kode untuk membangun jaringan

E. *Training* model

```
# optimization setup
sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)
model.compile(
    loss='categorical_crossentropy',
    optimizer=sgd,
    metrics=['accuracy'])

# training
model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=16,
    validation_split=0.2,
    shuffle=True,
verbose=2)
```

Gambar 4. 24. Kode untuk *training* model

Pada Gambar 4. 24 model dilatih dalam 16 *epoch* dengan hasil loss dan accuracy untuk *training*, serta val_loss dan val_accuracy untuk validasi. Nilai loss dan accuracy pada *training* dan validasi dengan 16 *epoch* adalah sebagai berikut:

1. Epoch 1 - 25s - loss: 0.4136 - accuracy: 0.8178 - val_loss: 0.2033 - val_accuracy: 0.9087
2. Epoch 2 - 24s - loss: 0.2341 - accuracy: 0.8988 - val_loss: 0.2133 - val_accuracy: 0.9350
3. Epoch 3- 24s - loss: 0.1980 - accuracy: 0.9178 - val_loss: 0.1524 - val_accuracy: 0.9538
4. Epoch 4 - 24s - loss: 0.1621 - accuracy: 0.9312 - val_loss: 0.1404 - val_accuracy: 0.9513
5. Epoch 5 - 24s - loss: 0.1375 - accuracy: 0.9469 - val_loss: 0.1012 - val_accuracy: 0.9613
6. Epoch 6 - 24s - loss: 0.1372 - accuracy: 0.9519 - val_loss: 0.1247 - val_accuracy: 0.9663
7. Epoch 7 - 24s - loss: 0.1124 - accuracy: 0.9588 - val_loss: 0.0860 - val_accuracy: 0.9750
8. Epoch 8 - 24s - loss: 0.0989 - accuracy: 0.9650 - val_loss: 0.0783 - val_accuracy: 0.9800

9. Epoch 9 - 24s - loss: 0.0977 - accuracy: 0.9663 - val_loss: 0.0707 - val_accuracy: 0.9800
10. Epoch 10 - 26s - loss: 0.0796 - accuracy: 0.9712 - val_loss: 0.0497 - val_accuracy: 0.9787
11. Epoch 11 - 24s - loss: 0.0742 - accuracy: 0.9719 - val_loss: 0.0482 - val_accuracy: 0.9812
12. Epoch 12 - 24s - loss: 0.0693 - accuracy: 0.9744 - val_loss: 0.0448 - val_accuracy: 0.9862
13. Epoch 13 - 24s - loss: 0.0505 - accuracy: 0.9819 - val_loss: 0.0419 - val_accuracy: 0.9837
14. Epoch 14 - 24s - loss: 0.0495 - accuracy: 0.9837 - val_loss: 0.0463 - val_accuracy: 0.9837
15. Epoch 15 - 24s - loss: 0.0611 - accuracy: 0.9803 - val_loss: 0.0417 - val_accuracy: 0.9875
16. Epoch 16 - 24s - loss: 0.0536 - accuracy: 0.9816 - val_loss: 0.0525 - val_accuracy: 0.9800

Berdasarkan traning di atas dapat dilihat bahwa waktu yang dibutuhkan untuk melatih data dengan 16 *epoch* adalah 387 detik atau rata-rata satu *epoch* membutuhkan waktu sekitar 24,18 detik. Akurasi pada *training* mengalami peningkatan yang cukup stabil dengan rata-rata nilai akurasi 0.9499 atau 94.99%. Akurasi pada validasi juga cukup stabil dengan perubahan kecil di setiap *epoch*nya, rata-rata nilai akurasi pada validasi 0.9657 atau 96,57%.

F. Pengujian pada citra satelit beresolusi tinggi

Jaringan yang telah dilatih dalam 16 *epoch* selanjutnya diuji pada citra satelit beresolusi tinggi. Citra satelit yang akan diuji dibuka lalu disimpan dengan variable bernama *image* menggunakan kode seperti pada Gambar 4. 25. Lalu citra tersebut diubah dulu bentuknya ke dalam 3 *channel* RGB dengan kodeGambar 4. 26. Tahap selanjutnya dalah mencari objek yang akan dideteksi sebagai kapal dengan menggunakan *window* berukuran 80x80 dengan pergerakan 10 pixel di setiap langkahnya. Lalu objek kapal yang ditemukan akan ditampilkan dengan kode pada Gambar 4. 27 dan Gambar 4. 28.

```
    image = Image.open(r'../ships-in-satellite-imagery/scenes/sfbay_1.png')
    pix = image.load()
```

Gambar 4. 25. Kode untuk memanggil citra yang diuji

```
    n_spectrum = 3
    width = image.size[0]
    height = image.size[1]


    # creat vector
    picture_vector = []
    for chanel in range(n_spectrum):
        for y in range(height):
            for x in range(width):
                picture_vector.append(pix[x, y][chanel])


    picture_vector = np.array(picture_vector).astype('uint8')
    picture_tensor     =     picture_vector.reshape([n_spectrum,     height,
width]).transpose(1, 2, 0)
```

Gambar 4. 26. Kode untuk mengubah bentuk citra yang diuji ke dalam 3 *channel* RGB.

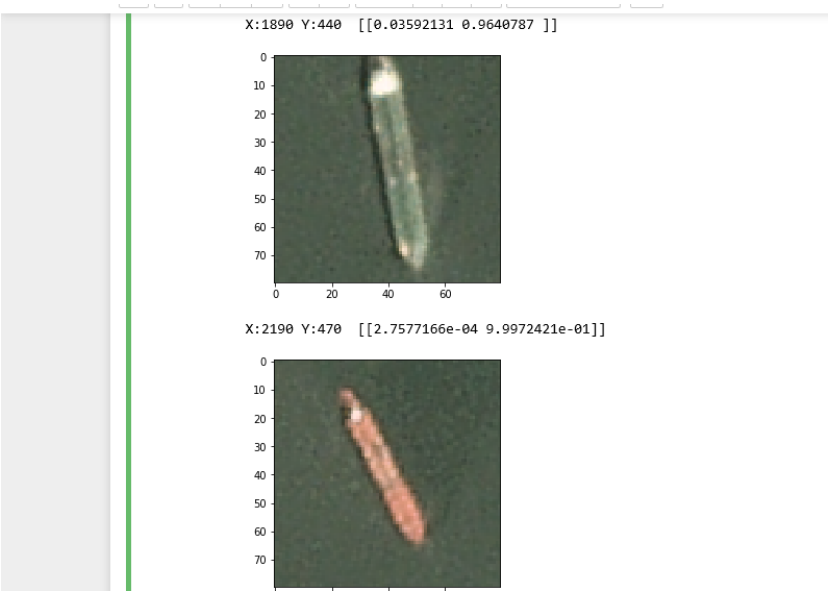
```
    def cutting(x, y):
        area_study = np.arange(3*80*80).reshape(3, 80, 80)
        for i in range(80):
            for j in range(80):
                area_study[0][i][j] = picture_tensor[0][y+i][x+j]
                area_study[1][i][j] = picture_tensor[1][y+i][x+j]
                area_study[2][i][j] = picture_tensor[2][y+i][x+j]
        area_study = area_study.reshape([-1, 3, 80, 80])
        area_study = area_study.transpose([0,2,3,1])
        area_study = area_study / 255
        sys.stdout.write('\rX:{0} Y:{1}  '.format(x, y))
        return area_study

    def not_near(x, y, s, coordinates):
        result = True
        for e in coordinates:
        if x+s > e[0][0] and x-s < e[0][0] and y+s > e[0][1] and y-s <
e[0][1]:
                result = False
```

Gambar 4. 27. Kode untuk mencari objek kapal

```
    def show_ship(x, y, acc, thickness=5):
        for i in range(80):
            for ch in range(3):
                for th in range(thickness):
                    picture_tensor[ch][y+i][x-th] = -1

        for i in range(80):
            for ch in range(3):
                for th in range(thickness):
                    picture_tensor[ch][y+i][x+th+80] = -1

        for i in range(80):
            for ch in range(3):
                for th in range(thickness):
                    picture_tensor[ch][y-th][x+i] = -1

        for i in range(80):
            for ch in range(3):
                for th in range(thickness):
                    picture_tensor[ch][y+th+80][x+i] = -1

    step = 10; coordinates = []

    for y in range(int((height-(80-step))/step)):
        for x in range(int((width-(80-step))/step) ):
            area = cutting(x*step, y*step)
            result = model.predict(area)
            if  result[0][1]  >  0.90  and  not_near(x*step,y*step,  88,
coordinates):
                coordinates.append([[x*step, y*step], result])
                print(result)

                plt.imshow(area[0])
                plt.show()
```

Gambar 4. 28. Mencari objek kapal dan menampilkannya.

Pixel yang dideteksi sebagai objek kapal akan ditampilkan seperti pada Gambar 4. 29



Gambar 4. 29. Hasil pencarian objek kapal dengan menggerakan *window*

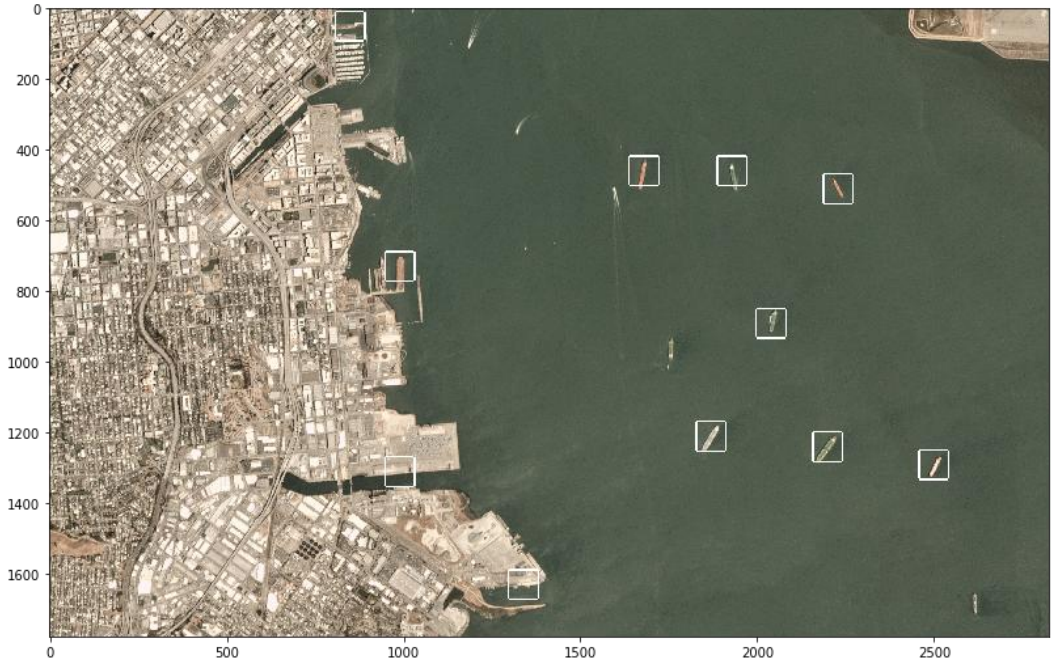
```
for e in coordinates:
    show_ship(e[0][0], e[0][1], e[1][0][1])
picture_tensor = picture_tensor.transpose(1,2,0)
picture_tensor.shape

plt.figure(1, figsize = (15, 30))
plt.subplot(3,1,1)
plt.imshow(picture_tensor)
plt.show()
```

Gambar 4. 30. Kode untuk memberikan *bounding box* pada citra yang diuji

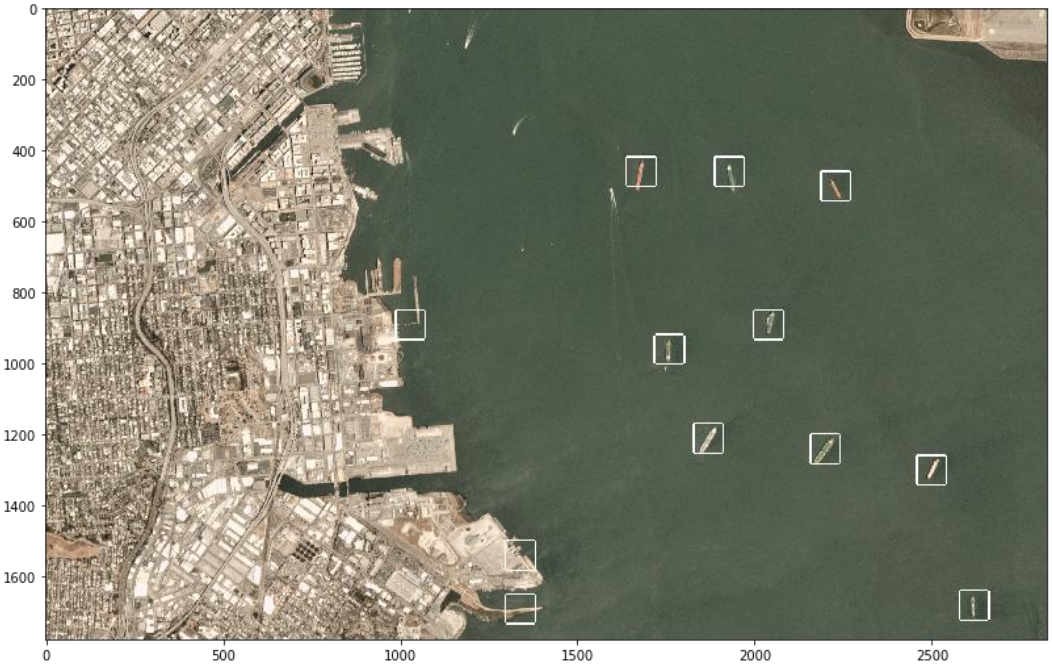
Agar lebih mudah dipahami, proses deteksi akan dilanjutkan dengan menandai lokasi objek berdasarkan koordinat objek yang dideteksi dengan *bounding box* menggunakan kode pada Gambar 4. 30. Berikut

Gambar 4. 31,

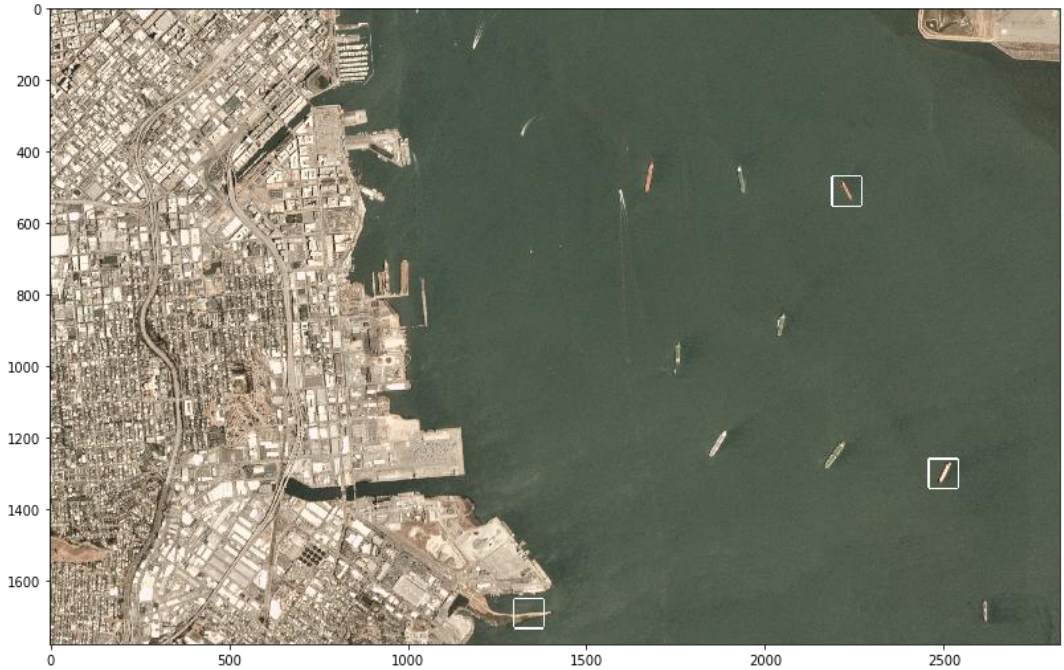
Gambar 4. 32, Gambar 4. 33, Gambar 4. 34, Gambar 4. 35, dan Gambar 4. 36 yang merupakan hasil akhir deteksi dengan jumlah *epoch* yang berbeda:



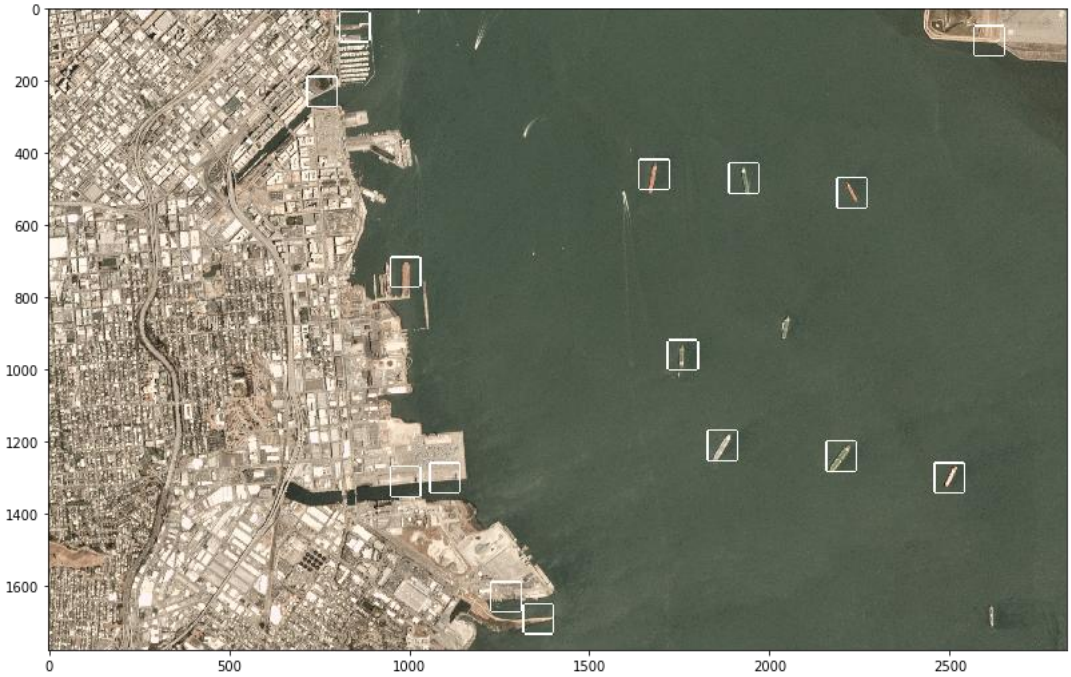
Gambar 4. 31. Hasil akhir deteksi dengan 15 *epoch*



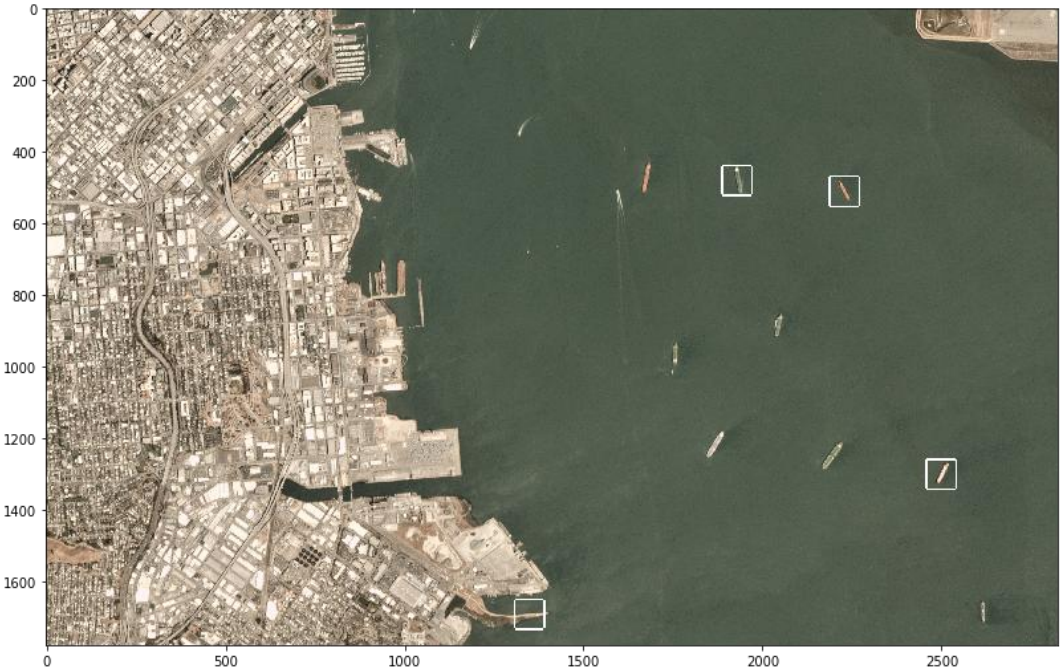
Gambar 4. 32. Hasil akhir deteksi dengan 16 *epoch*



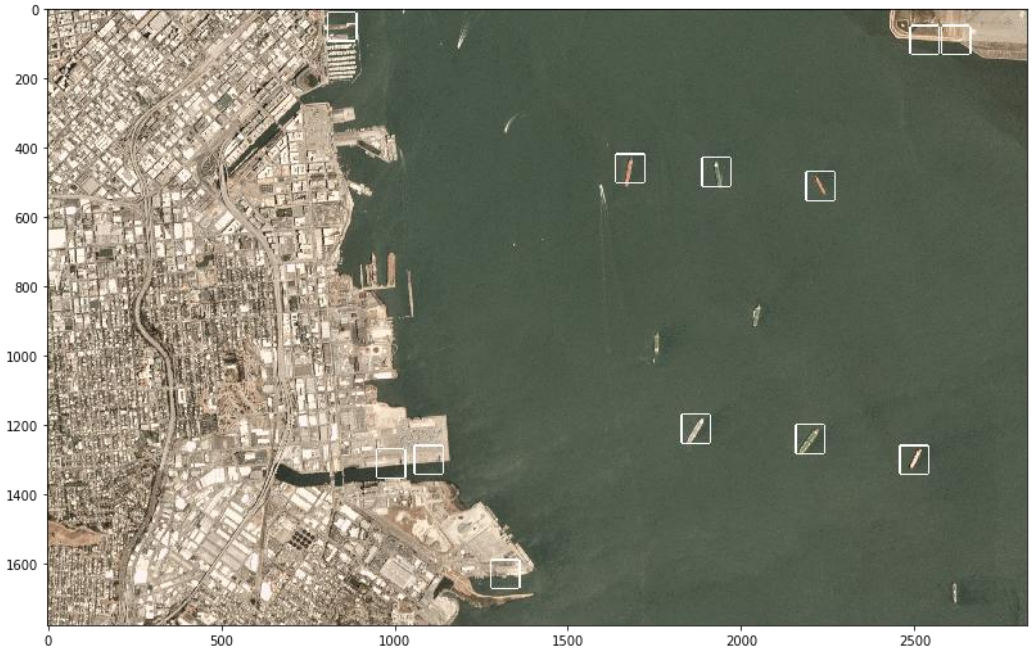
Gambar 4. 33. Hasil akhir deteksi dengan 17 *epoch*



Gambar 4. 34. Hasil akhir deteksi dengan 18 *epoch*



Gambar 4. 35. Hasil akhir deteksi dengan 19 *epoch*



Gambar 4. 36. Hasil akhir deteksi dengan 20 *epoch*

Hasil akhir deteksi objek dapat dilihat berdasarakan *training* dengan masing-masing jumlah *epoch* yang berbeda, yaitu 15-20 *epoch*. Keenam percobaan dengan *epoch* berbeda tersebut dilatih dan diuji dengan jumlah data yang sama. Hasil deteksi dengan *epoch* 15

Gambar 4. 31, dengan 16 *epoch*

Gambar 4. 32, 17 *epoch* Gambar 4. 33, 18 *epoch* Gambar 4. 34, 19 *epoch* Gambar 4. 35, dan 20 *epoch* Gambar 4. 36. Berdasarkan hasil tersebut dapat dilihat bahwa deteksi dengan model yang dilatih sebanyak 16 siklus memiliki hasil deteksi yang terbaik.

4.2 Perbandingan hasil implementasi

Pada penelitian ini metode SVM dan CNN diimplementasikan dan dibandingkan dengan data yang sama, baik ukuran, jumlah, dan pembagian train/val sama yaitu 80% dan 20%. Kedua program hasil implementasi juga dijalankan pada satu device yang sama dengan Bahasa pemrograman yang sama pula, yaitu Python versi 3.7.

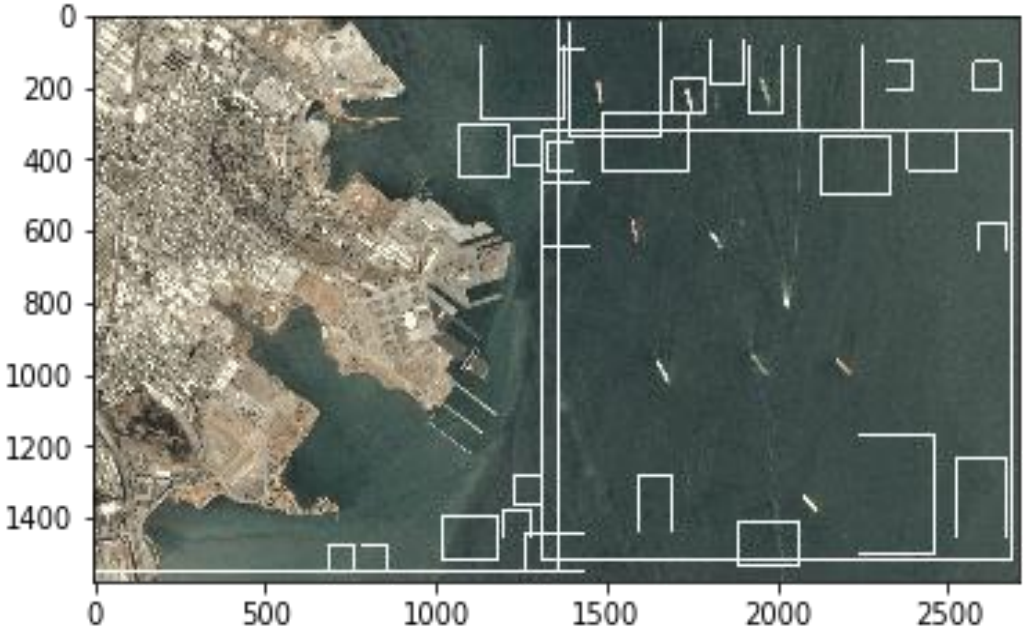
Waktu yang dibutuhkan CNN dan SVM untuk melatih model juga berbeda. SVM hanya membutuhkan waktu 5,22 detik, sedangkan CNN membutuhkan waktu 387 detik. CNN memiliki selisih 382 detik lebih lama dari SVM untuk melatih model. Untuk waktu pengujian pada citra satelilt beresolusi tinggi dengan citra yang sama yaitu 2709x1577 pixel, SVM membutuhkan waktu 57,76 detik dan CNN membutuhkan waktu 2095,63 detik.

Baik CNN maupun SVM memiliki akurasi yang tinggi pada saat diuji dengan data *validation* CNN 96,57% dan SVM 94,38%. Walaupun pada *testing* dengan data *validation* SVM memiliki akurasi yang tinggi tetapi saat diuji pada citra satelit beresolusi tinggi hasilnya tidak cukup baik. Banyak terjadi kekeliruan deteksi baik pada kapal maupun non-kapal.

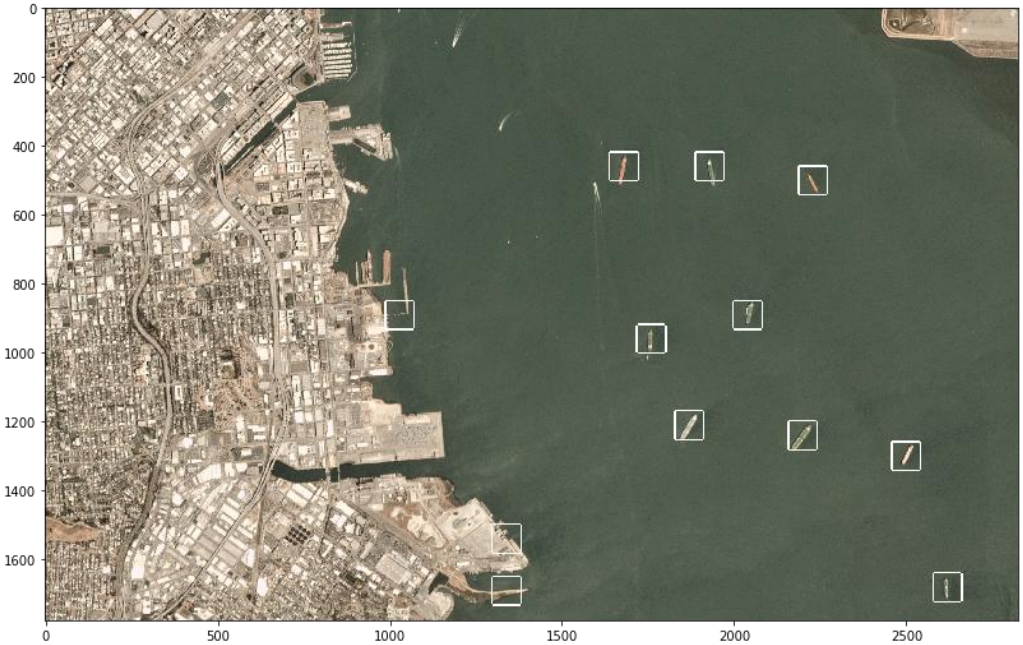
Seperti dapat dilihat pada, CNN dapat mendeteksi objek kapal lebih baik daripada SVM ketika diuji dengan tangkapan pelabuhan dari gambar satelit resolusi tinggi. Hampir semua objek kapal dalam gambar dapat dideteksi dengan baik oleh CNN. Sedangkan dalam SVM, ada banyak deteksi kesalahan. Kita dapat membandingkan perbedaan hasil deteksi CNN dan SVM pada

Gambar 4. 37 dengan Gambar 4. 38 dan Gambar 4. 39 dengan Gambar 4. 40.

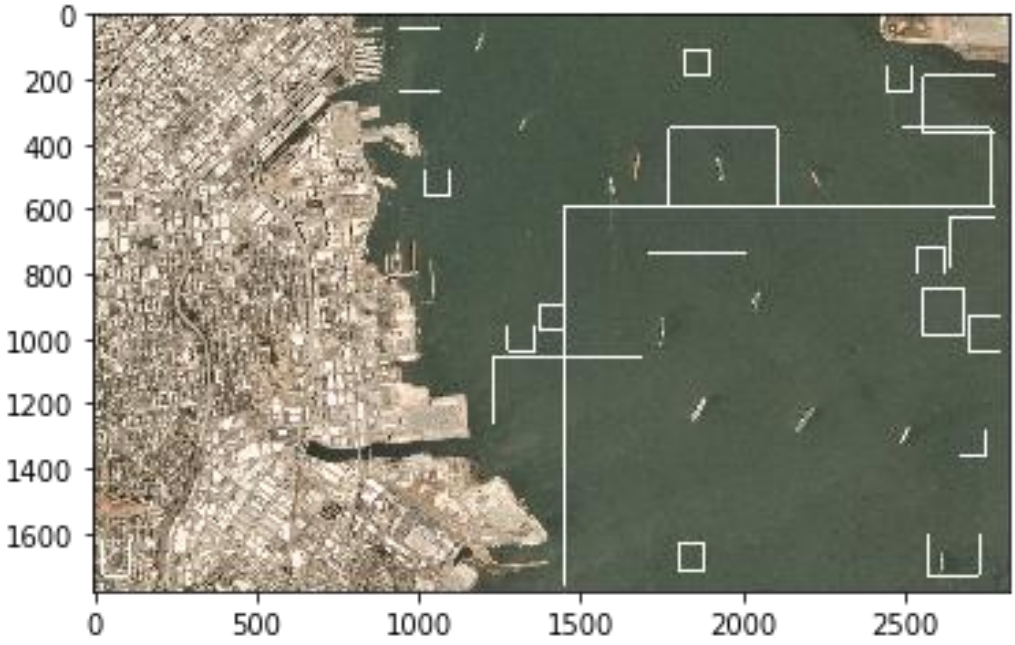
Gambar 4. 37. Hasil deteksi objek dengan CNN



Gambar 4. 38. Hasil deteksi objek dengan SVM



Gambar 4. 39. Hasil deteksi objek dengan CNN



Gambar 4. 40. Hasil deteksi objek dengan SVM