diff = diff/1000

absDiff = np.abs(diff)

MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE400 SUMMER PRACTICE REPORT

Student Name: Berk Alperen Bener
Student ID:2165991
SP Company Name: ESEN System Integration
SP Date: 08.02.2021 - 05.03.2021
Submission Date: 13.04.2021

TABLE OF CONTENT

1.Introduction

I have performed my EE400 Internship in ESEN System Integration, which is aerodynamics company which is subsidary of My practice lasted totally 4 weeks, starting from 08.02.2019 and ending in 05.03.2019. The division that I work was dealing with UAV(Unmanned Aerial Vehicle) systems. One of the subgroup was trying to make end to end, scalable module that can be used to determine different kind of flight data without conventional hardware sensors.

The report starts with brief history of company then introduction to main problem of determinstion different kind of flight data and current solutions to solve them. After that I proposed new approach to determination of flight data using machine learning algorithms. These part includes gathering data, building models,training and testing them. Afer conclusion there exists references that I used to make design decision and appendices that includes all the codes that I wrote throughout the internship which I used Python.

2.Description of the Company

2.1 Company Name

Esen System Integration

2.2 Company Location

Üniversiteler Mahallesi, Titanyum C Blok, ODTÜ Teknokent, 06800 Çankaya/Ankara

2.3 Organizational Structure of the Company

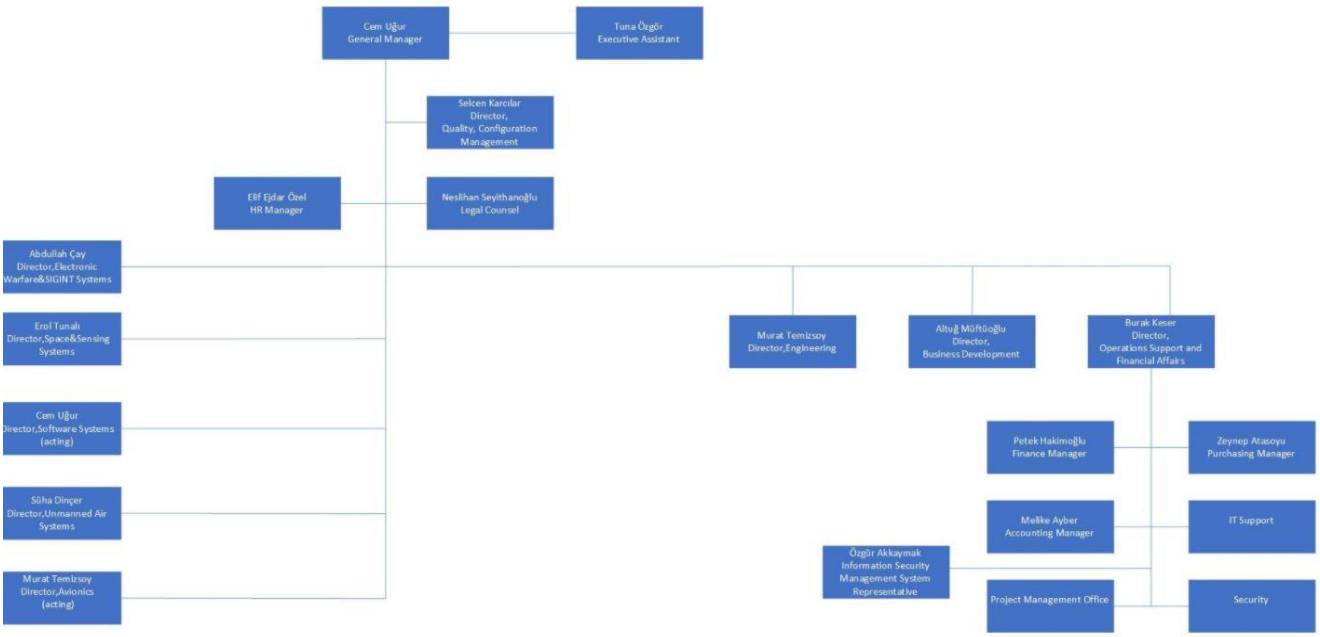

Figure 1: Organizational structure of the company in managerial level

2.4 Main Area of Business

ESEN has wide range of products which includes in the fields of aviation, space, defense and security. Specifically it has capabilities in four areas big data fusion, analysis and decision support system,aircraft systems, Electronic Warfare and Signal Intelligence Systems, Satellite and Space Systems.

2.5 Brief History of the Company

Esen System Integration established in February 2012 as an affiliated company of US-based of Sierra Nevada Corporation. The company moved in its current office in ODTU Teknokent in July 2012, then completed its first flight in March 2013. In the past 8 years it consistenly increased its headcount currently to 180 people.

3. Project Descripton

The project that I involved was designing and implementng end to end system that deals with low altitude multimodal UAV data that provided by AUAir Dataset and from that image predicting altitude of the UAV using only image. Under the supervision of (BS. Hacettepe University Electrical and Electronics Engineering, MS. METU EE) Erol Tunalı. I delivered working model in 20 days of my compulsory internship.

The spesific UAV that I worked with was low altitude quadcopter that is used for traffic surveillance. These type of quadcopters generally fly in the range of 50 meters. The project requires to create an altitude estimation model that can be used spesifically for this range to eliminate the need of external sensors which increase cost, weight and battery consumption. This model requires to make estimation at most 5 percent error margin that is comparable with other methods.

3.1 Current Solutions

The UAVs mainly use devices that their purpose is to determine the altitude of the with respect to certain fixed point. These family of devices are called as altimeter. The altimeters calculate the height between sea level or ground level. Different type of altimeters use different type of physical principles to calculate height such as pressure density, electromagnetic wave reflection etc. But mainly there exist four type of altimeters barometric, radio, GNSS and laser altimeters. Lets review them.

3.1.1 Barometric Altimeters

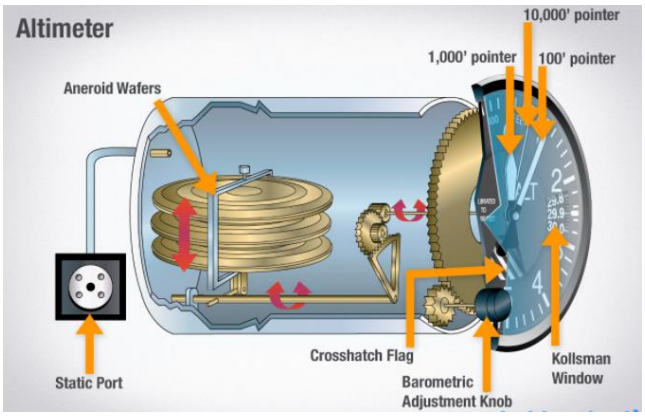

Figure 2: Internal Structure of the barometric altimeter

Barometric altimeters**[1]** use the basic principle asserts that as the observant's altitude increase atmospheric pressure decrease. Of course for most accurate measurement device have to reckon other variables that can effect the measured pressure that depends on environmental conditions. Such as temperature, molar mass of the air. Basically barometric altimeter uses the following formula to determine altitude.

$$z = c\,T\,\log(P_o/P)$$

Figure 3: Barometer formula

In this formula c represents gravity and molar mass, T represents temprature and while P represents measured pressure, P0 represents atmospheric pressure in the sea level. The altimeter is mechanical and drive mechanical pointer to show altitude data.

3.1.2 Radio Altimeters

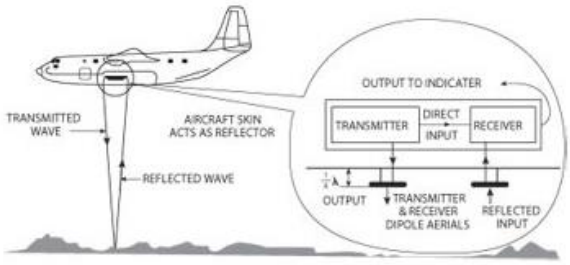

Figure 4: Radio Altimeter working principle

Radio altimeter**[2]** use the same principle of radar. There exists mounted bacon on the vehicle than this bacon creates electromagnetic waves. These waves travel through air and hits the ground that directly below the vehicle and reflects back. The time between initial signal and reflected signal is measured, using this data the altitude is found. Since electromagnetic waves are in the speed of light the altitude measurement occurs very fast. They can be affected my surface irregularities but nevertheless it is reliable method for measurement.

3.1.3 Global Navigation Satellite System(GNSS) Altimeter

Figure 5: GNSS Structure that helps altitude determination using different bacons

GNSS**[3]** system use multilateration to determine the altitude of the vehicle by using three or more satellite. The system use radio waves to determine relative position of the vehicle with respect to satellites. But eventhough it can be useful for high altitude vehicles, due to error margin of nearly 5 meters it is not useful for low altitude aerial vehicles especially for the vehicles that requires precise movements.

3.1.4 Laser Altimeter

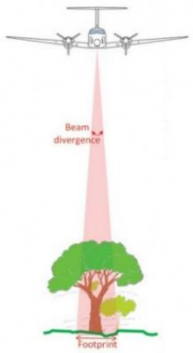

Figure 6: Laser Altimeter

Laser altimeters**[4]** use the same principle that is used in radio altimeters but the difference is it sends electromagnetic waves that in the range of visible light not radiowaves. When these waves are sent to ground they reflects back and these reflected lights are collimated by using lenses and mirrors resulting collimated lights are directed to photocells. Resulting trip time is used to determine the altitude.

3.2 Proposed Solutions

To get rid of the external sensors that can increase initial and maintanince cost, we propose image processing based solution to find altitude data using machine learning algorithms. The first is based on solely image that and uses CNN(Convolutional Neural Network) algorithm.

3.2.1 Finding Altitude Data Using CNN and Only Image Data

CNN is a class of deep learning algorithms that mainly is used as analyzing visual data. The architectural layer that distinguish it comparing MLP(Multi Layer Perceptron) is it has convolutional layer. Since image data is 2d array with huge number of pixels and pixel values, using MLP would create exploding number of weights that would result intractable calculation of weights due to curse of dimensionality. The convolutional layers overcome this problem and make calculation feasable. CNNs are usually used for classification problem but in this project we are deploying them for regression problem.

3.2.1.1 Dataset

To train our algorithms to make prediction about UAVs altitude we need multimodal dataset. This dataset should include beside images at least altitude that are matched to corresponding image data. In 2020 from the researchers at Aarhus University Erdal Kayacan and Ilker bozcan published paper in IEEE International Conference on Robotics and Automation (ICRA) 2020**[5]** that gives comprehensive explanation for such low altitude UAV dataset. The image dataset and corresponding annotations for external sensors can be accesssed through https://github.com/bozcani/auairdataset. Lets review the dataset.

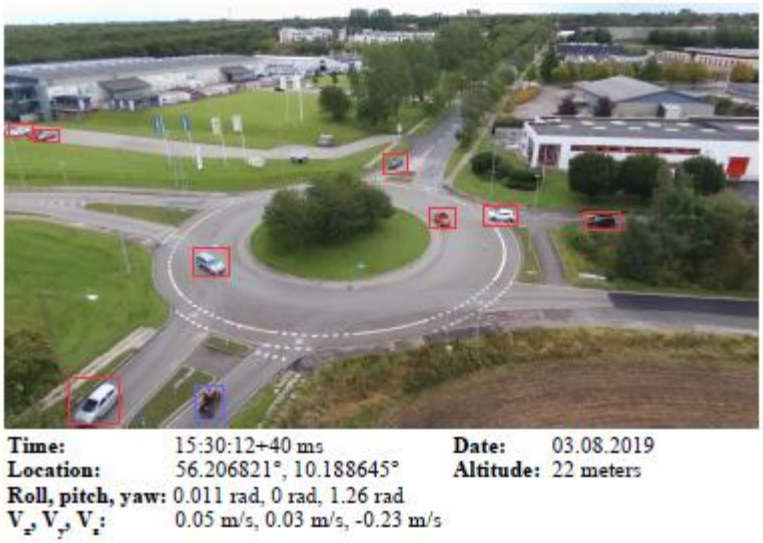

Figure 7: Sample image and data values from dataset

Dataset is created in 2020, the aerial images are taken by mounted camera on quadrocopter(Beepop) that records video footage in 1980x1080 pixel quality with 30 fps for 2 hours. But 30 fps is too fast to create recognizable change between sequential images as a result, published dataset includes 5 frame per second with same resolution. The aerial images are taken between 10-30 meters range and from 45-90 degree angle. Beside these images there exist annotations that are measured by using other sensors. The location which is in the form of latitude and longitude is found using GPS. The roll,pitch,yaw and linear velocity inx x,y and z direction is found by using IMU sensor. Eventhough there exist object annotations for certain type of object for every image, we didn't use this data.

3.2.1.2 Preprocessing the Data

First of all due to need for high processing power it is was not feasable form my personal computer to make necessary training. Because of that to leverage free processing power, in our project we used kaggle. In kaggle all coding is done using Python. Because of that our main programming language is Python. First we uploaded image data and annotations into Kaggle containers. Images are 1980x1080 pixel 3 channel RGB images in .jpg format and annotations are in .json format with the following syntax. Per image it is in the dictionary format, that includes following information upto significiant digit that is shown below.

"annotations": [{"image_name": "frame_20190829091111_x_0001973.jpg", "image_width:": 1920.0, "image_height": 1080.0, "platform": "Parrot Bebop 2", "time": {"year": 2019, "month": 8, "day": 29, "hour": 9, "min": 11, "sec": 11, "ms": 394400.0}, "longtitude": 10.18798203255313, "latitude": 56.20630134795274, "altitude": 19921.6, "linear_x": 0.03130074199289083, "linear_y": 0.028357808757573367, "linear_z": 0.0744575835764408, "angle_phi": -0.06713105738162994, "angle_theta": 0.06894744634628296, "angle_psi": 1.1161083340644837, "bbox": [{"top": 163, "left": 1098, "height": 185, "width": 420, "class": 1}, {"top": 421, "left": 1128, "height": 176, "width": 393, "class": 1}, {"top": 927, "left": 1703, "height": 153, "width": 183, "class": 0}]},

Figure 8: Sample of annotation entry

Before preprocess data we need to be careful about the certain feature of the dataset. As it is seen below the aerial image dataset is not uniformly distributed regarding it's altitude. As a result we expect our model learn certain features better which leads to performance in the test data could vary with respect to altitude. As a result to measure performance accuratelly we need to divide test data regarding its altitude and test them based on it.

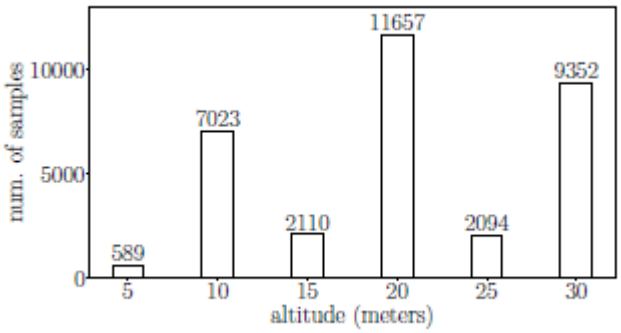

Figure 9: Number of image data after dividing it regarding altitudes

First we loaded our annotations.json file from Kaggle's container, then we saved image names and corresponding altitudes to array. Using scikitlearn library's traintestsplit function we dvided these image names and altitudes into train, test and validation set with ratio of 0. 64,0.1 and 0.2 respectively. After this we saved these partition as .pickle file so we don't have to repeat this data partition again and with further model improvements because of the same partition, it would create much accurate comparison.

Then we iterated over first training and validation imagename array and read them from Kaggle's container using OpenCV library's function cv2.imread() after that we resized image into (64, 64) array. Since it is aerial data it is not ideal to resize image to smaller version, this reduce the number of potential features and as altitude increase this can effect the performance of the algorithm more.

However, eventhough Kaggle gives superior computing resources comparing PC. Eventhough we deploy Kaggle's GPU unit it was not enough to train our algorithm. As a result, experimentally (64, 64) id the biggest resolution that our algorithm can be trained in Kaggle which is found experimentally.After that, again we saved our image data into arrays.

For test dataset, we iterated over testname array and divided them into 5 m intervals as follows(2.5-7.5, 7.5-12.5, 12.5-17.5, 17.5-22.5, 22.5-27.5, 27.5-32.5) and for each interval we divided appropriate imagename and corresponding altitude value. Again, we saved these files as .pickle. Finally we iterated over test images and resized them into (64, 64) arrays as we did to train and test data before and saved them as .pickle.

3.2.1.3 Creation of the Model

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 16)        448
_________________________________________________________________
activation_3 (Activation)    (None, 64, 64, 16)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                262160
_________________________________________________________________
activation_4 (Activation)    (None, 16)                0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16)                64
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 68
_________________________________________________________________
activation_5 (Activation)    (None, 4)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 5
=================================================================
Total params: 262,809
Trainable params: 262,745
Non-trainable params: 64
_________________________________________________________________
None
```

Figure 10: Summary of the CNN architecture:

To build architecture we used layers that are part of Tensorflow library. The architecture have input layer that is compatible with resized image of 64 pixel width, 64 pixel height and 3 channels which are RGB(Red,Green,Blue). Then we iterate over Convolutional(CONV)->Rectifier(RELU)->Batch-Normalization(BN)->Max-Pooling(MP) layers.

For every loop in iteration the filter sizes in the Conv2D() layer increase as the following (16, 32, 64, 128). These filter sizes represents how many different kernels we are going to use for each layer. The kernels are basically tilings that goes over the image and the overlapping regions between kernel and image takes product and creates output feature vector.

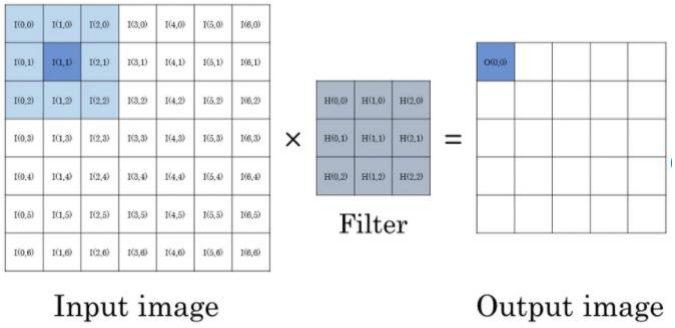

Figure 11: Sample 7x7 Image and 3x3 Filter

We choose (3,3) kernel size which is used in successful algorithms like VGGNet, and also we want to learn smaller features, because it is aerial image. Also, to preserve image data size we restrict our stride into (1,1) and padding as 'same', since we already reduced the image size enough.

After convolutional layer the architecture continues with Rectifier(RELU) layer which swaps the negative convolutional result and make them zero. By using this layer, we prevent our weight to stuck around zero or explode to infinity. After RELU we put batch-normalization layer the position of the layer which is coming after every RELU in the architecture is common practice it prevents gradient explosion as we update our weights by normalizing their values.

Then we use Max-Pooling layers which allows us to select the most important feature over a selected portion of the image and ignore the less important features. That way we decrease our computational load. To minimize the reduction of the image size we determine Max-Pooling layer's filter size as (2,2). As we go deeper into architecture to compensate the spatial reduction due to Max-Pooling layers which results decrease in feature number, we apply bigger filter sizes. Because of that our filter size increases. The size of the filters of being power of 2 is just a common practice. Before going into regular densely-connected NN layer we flatten our 2D data using Flatten(). Then following (16,4,1) size of densely-connected NN layers we finally make regression using 'linear' as activation function. To reduce overfit we used Dropout() layer with 0.5 threshold value after 16 node regular densely connected NN layer.

We used Adam optimizer which uses stochastic gradient descent to update weights. We also used 'mean_absolute_percentage_error' as our loss function which is generally used as loss function for regression problems. At is actual value Ft is predicted value, this equation later multiplied by 100.

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

*Figure 12: Mean Absolute Percentage Error Function**[6]***

3.2.1.4 Training and Testing

We train our model for 100 epoch. We have choosen 100 epoch regarding train and valudation loss graph down below. By looking at the proportion between train and validation loss we can understand whether our model is tend to overfit, underfit or how overall will perform.

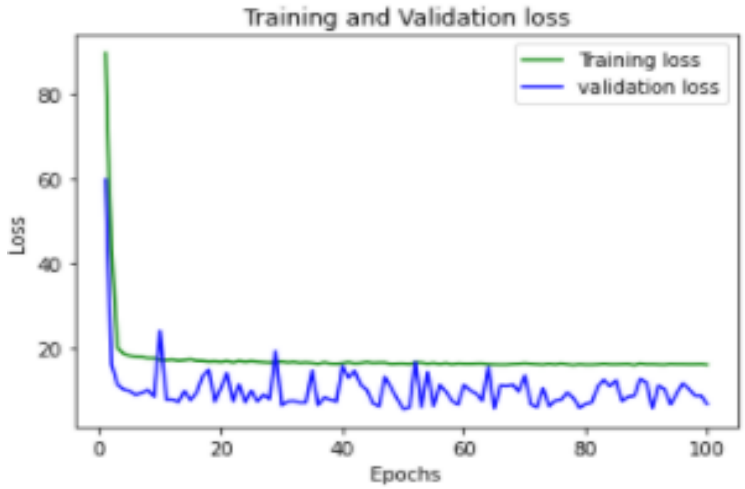

Figure 13: Train and Validation loss vs. Epoch

By looking at the box graph below, we can see how our algorithm performed for different range of altitudes. As we expect since slot 2,4, 6 have higher number of samples they have lower error average. But, as you can see eventhough 6 hoave high number of samples it perform poorly comparing slot 2 and 4. This happens because resizing most significiantly effected highest altitudes and highest altitudes intrinsically have less feature to distinguish them. But overall for every altitude slot we have in the margin of 5 percent error which corresponds to errors between 20 cm to 150 cm.

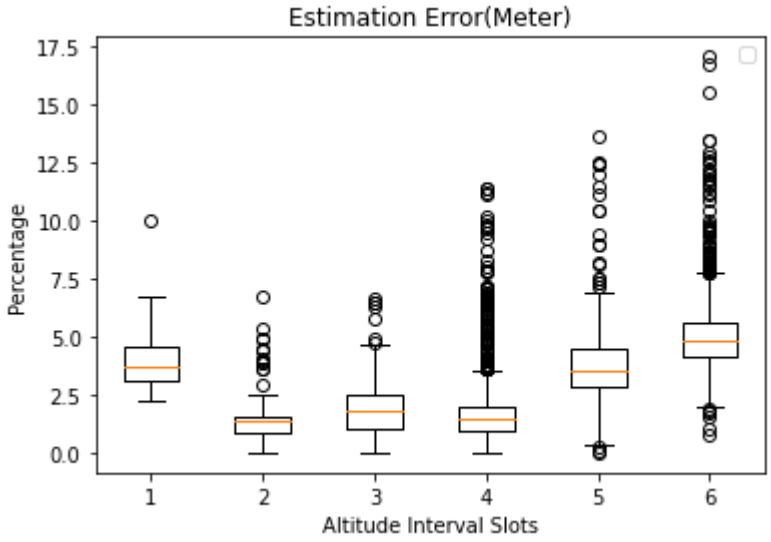

Figure 14: Estimation Error Percentage vs. Altitude Intervals

4.Conclusion

In this project we created end to end module that is trained with labeled aerial imagery and use CNN to make regression to predict altitude for given image. Resulting model is tested for different altitudes and it shows consistent error margin with estimations which reckons number of samples and natural limitations of images due to their altitudes. Overall the model gave promising error margin result which is under 5 percent which can be further improved with increasing number of samples, reducing resizing or including other sensor data. As a result it is proof of concept that we can replace external altimeters with machine learning models using image data.

5. References

1. https://www.basicairdata.eu/projects/barometric-altimeter/
2. https://en.wikipedia.org/wiki/Radar_altimeter
3. https://en.wikipedia.org/wiki/GNSS_applications
4. https://www.sciencedirect.com/topics/earth-and-planetary-sciences/laser-altimeter
5. Bozcan and E. Kayacan, "AU-AIR: A Multi-modal Unmanned Aerial Vehicle Dataset for Low Altitude Traffic Surveillance", IEEE International Conference on Robotics and Automation (ICRA) 2020.
6. https://en.wikipedia.org/wiki/Mean_absolute_percentage_error

6.Appendices

6.1 Data Preprocessing

```
from sklearn.model_selection import train_test_split
from keras.models import model_from_json
import json
import numpy as np
import cv2
import pickle

altitudes = []
image_names = []
altitudes1 = []
image_names1 = []
altitudes2 = []
image_names2 = []
altitudes3 = []
image_names3 = []
altitudes4 = []
image_names4 = []
altitudes5 = []
image_names5 = []
altitudes6 = []
image_names6 = []


arrays = []
list_altitudes = []
list_image_names = []
inputImages = []
trainImagesX1 = []
validationImagesX1= []
```

```
with open('../input/auair2019annotations1/annotations.json') as json_file:
    data = json.load(json_file)
    lim1 = data['annotations']
    for p in lim1:
        altitudes.append(p['altitude'])
        image_names.append(p['image_name'])

split = train_test_split(altitudes, image_names, test_size=0.2, random_state=42)
(trainAttrX, testAttrX, trainImagesX, testImagesX) = split

split = train_test_split(trainAttrX, trainImagesX, test_size=0.2, random_state=42)
(trainAttrX, validationAttrX, trainImagesX, validationImagesX) = split

with open("trainImagesXname.pickle", "wb") as fp:
    pickle.dump(trainImagesX, fp)

with open("validationImagesXname.pickle", "wb") as fp:
    pickle.dump(validationImagesX, fp)

with open("testImagesXname.pickle", "wb") as fp:
    pickle.dump(testImagesX, fp)

for path in trainImagesX:
    if type(path) is str:
        pathim = '../input/auair1/images/' + path
        image = cv2.imread(pathim)
        image = cv2.resize(image, (64, 64))
        trainImagesX1.append(image)

for path in validationImagesX:
    if type(path) is str:
        print("jk")
        pathim = '../input/auair1/images/' + path
        image = cv2.imread(pathim)
        image = cv2.resize(image, (64, 64))
        validationImagesX1.append(image)

with open("trainImagesX.pickle", "wb") as fp:
    pickle.dump(trainImagesX1, fp)

with open("validationImagesX.pickle", "wb") as fp:
    pickle.dump(validationImagesX1, fp)

with open("trainAttrX.pickle", "wb") as fp:
    pickle.dump(trainAttrX, fp)

with open("validationAttrX.pickle", "wb") as fp:
    pickle.dump(validationAttrX, fp)


for alt, img in zip(testAttrX, testImagesX):
    if 2500 < alt < 7500:
        altitudes1.append(alt)
        image_names1.append(img)
    elif 7500 < alt < 12500:
        altitudes2.append(alt)
        image_names2.append(img)
    elif 12500 < alt < 17500:
        altitudes3.append(alt)
        image_names3.append(img)
    elif 17500 < alt < 22500:
        altitudes4.append(alt)
        image_names4.append(img)
    elif 22500 < alt < 27500:
        altitudes5.append(alt)
        image_names5.append(img)
    elif 27500 < alt < 32500:
```

```
        altitudes6.append(alt)
        image_names6.append(img)

list_altitudes.append(altitudes1)
list_altitudes.append(altitudes2)
list_altitudes.append(altitudes3)
list_altitudes.append(altitudes4)
list_altitudes.append(altitudes5)
list_altitudes.append(altitudes6)

list_image_names.append(image_names1)
list_image_names.append(image_names2)
list_image_names.append(image_names3)
list_image_names.append(image_names4)
list_image_names.append(image_names5)
list_image_names.append(image_names6)

with open("list_image_names.pickle", "wb") as fp:
    pickle.dump(list_image_names, fp)

with open("list_altitudes.pickle", "wb") as fp:
    pickle.dump(list_altitudes, fp)

with open('list_image_names.pickle', 'rb') as list_image_names:
    list_image_names = pickle.load(list_image_names)

for number_image_names in list_image_names:
    inputImages = []
    for path in number_image_names:
        if type(path) is str:
            print("tjh")
            pathim = '../input/auair1/images/' + path
            image = cv2.imread(pathim)
            image = cv2.resize(image, (64, 64))
            inputImages.append(image)
    ar = np.array(inputImages)
    arrays.append(ar)

with open("images.pickle", "wb") as fp:
    pickle.dump(arrays, fp)
```

6.2 Training CNN Algorithm

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from keras.models import model_from_json
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
import json
import numpy as np
import pickle

arrays = []

with open('../input/inputs/trainImagesX.pickle', 'rb') as trainImagesX:
    trainImagesX = pickle.load(trainImagesX)

with open('../input/inputs/validationImagesX.pickle', 'rb') as validationImagesX:
```

```
    validationImagesX = pickle.load(validationImagesX)

with open('../input/inputs/trainAttrX.pickle', 'rb') as trainAttrX:
    trainAttrX = pickle.load(trainAttrX)

with open('../input/inputs/validationAttrX.pickle', 'rb') as validationAttrX:
    validationAttrX = pickle.load(validationAttrX)

def create_cnn(width, height, depth, filters=(16, 32, 64, 128)):
 inputShape = (height, width, depth)
 chanDim = -1
 inputs = Input(shape=inputShape)
 for (i, f) in enumerate(filters):
  if i == 0:
   x = inputs
  x = Conv2D(f, (3, 3), padding="same")(x)
  x = Activation("relu")(x)
  x = BatchNormalization(axis=chanDim)(x)
  x = MaxPooling2D(pool_size=(2, 2))(x)
  x = Flatten()(x)
  x = Dense(16)(x)
  x = Activation("relu")(x)
  x = BatchNormalization(axis=chanDim)(x)
  x = Dropout(0.5)(x)
  x = Dense(4)(x)
  x = Activation("relu")(x)
  x = Dense(1, activation="linear")(x)
  model = Model(inputs, x)
  return model

trainImagesX = np.asarray(trainImagesX)
validationImagesX = np.asarray(validationImagesX)
trainAttrX = np.asarray(trainAttrX)
validationAttrX = np.asarray(validationAttrX)

model = create_cnn(64, 64, 3)
opt = Adam(lr=1e-3, decay=1e-3 / 200)
model.compile(loss="mean_absolute_percentage_error", optimizer=opt)

print(model.summary())

history = model.fit(x=trainImagesX, y=trainAttrX,
validation_data=(validationImagesX, validationAttrX), epochs=50, batch_size=8)
loss_train = history.history['loss']
loss_val = history.history['val_loss']
epochs = range(1,51)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

model_json = model.to_json()

with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
```

6.3 Testing the Model

```
from tensorflow.keras.models import Model
from keras.models import model_from_json
import matplotlib.pyplot as plt
import numpy as np
import pickle
datas = []

with open('images.pickle', 'rb') as validationAttrX:
    arrays = pickle.load(images)

with open('list_altitudes.pickle', 'rb') as list_altitudes:
    list_altitudes = pickle.load(list_altitudes)

arrays = np.asarray(arrays)
list_altitudes = np.asarray(list_altitudes)

for f, b in zip(list_altitudes, arrays):
    f = np.array(f)
    b = np.array(b)
    preds = model.predict(b)
    diff = preds.flatten() - f
    diff = diff/1000
    absDiff = np.abs(diff)
    datas.append(absDiff)

with open("datas.txt", "wb") as fp:
  pickle.dump(datas, fp)

datas = np.load("datas.txt", allow_pickle=True)
plt.boxplot(datas[0], datas[1], datas[2],datas[3],datas[4],datas[5], names=c("2.5-7.5","7.5-
12.5","12.5-17.5","17.5-22.5","22.5-27.5","27.5-32.5"))
plt.title('Estimation Error(Meter)')
plt.legend()
plt.ylabel('Meter')
plt.legend()
plt.show()
```