Optimization Techniques

Final Project

Presented by:

Omar Ihab El-Hariry 5360
Ahmed Tharwat Magdy 5336
Aly Mahmoud 5304
Mostafa Ehab ElZwawy 5311
Malak Kassem 5979
Omar Hisham Tawfik 5845
Youssef Amr ElEssawi 4284
Ayman Ahmed Abdelaziz 5529

CONTENTS

Summary

Problem Statement:

It was required to find the resolution of the scanned image of the national ID, that achieves the best compromise between transfer time from the mobile and the computer using bluetooth interface.

Solution approach:

The solution approach was done in several steps:

1. Data acquisition using simulators
2. Creating the dataset
3. Analysing the data
4. Creating the surrogate model and training it
5. Testing the surrogate model and fine tune the hyperparameters
6. Get the pareto fronts and find the optimal, using multi-objective optimization methods

Language used:

- Python3

Framework used:

- Tensorflow 2.0

Libraries used:

1. Numpy
2. Matplotlib
3. Keras
4. Seaborn
5. Pandas

Overview

The problem is to get the optimal resolution, that is the width and the height of the scanned image, that achieves the best compromise between minimizing file transfer time using bluetooth and maximizing quality.

It was found that what affects the transfer time of the image, is the size of the image and the pixels of the image are what affects the size of the image, more pixels means bigger size, means longer transfer time, and what defines the quality of the image is how the pixels of the image are distributed throughout the screen, so the quality of the image can be measured using what is known by PPI or Pixels-Per-Inch, so that means the higher the resolution of the image, the more pixels per inch, so here we can see the conflict of interests between maximizing the quality and minimzing the transfer time.

Note:

Resizing the images causes the image to make an interpolation of the resized image using interpolation algorithms like KNN and bilinear that either fills the gaps if resizing to a bigger image or averaging the pixels if resizing to a smaller image, and these algorithms yields great results in interpolating the images, so the quality pretty much increases while increasing the size of the original image.

Dataset

Data acquisition and dataset creation:

Data acquisition was done by simulating the transfer process by trying different resolutions with different aspect ratios, starting all the way from a 16x16 image to a 16K image, and then calculating the size of the image taking into consideration no compression used to the image, after calculating the size the transfer time was calculated in seconds using Bluetooth 1.0 interface with latency up to 4 seconds applied, and the PPI was calculated by the reciever's screen dimensions.

After aquiring the data, the dataset was created in the form of a spreadsheet (csv file)

The dataset created has 194 enteries, and has 5 columns: Width, Height, size in Megabytes, transfer time in Seconds and quality in PPI

The dataset was split to 60% of its enteries to training and 40% to validation, this split was chosen because the dataset isn't quite large, and to make get a rough sense if our model is overfitting the training data or not.

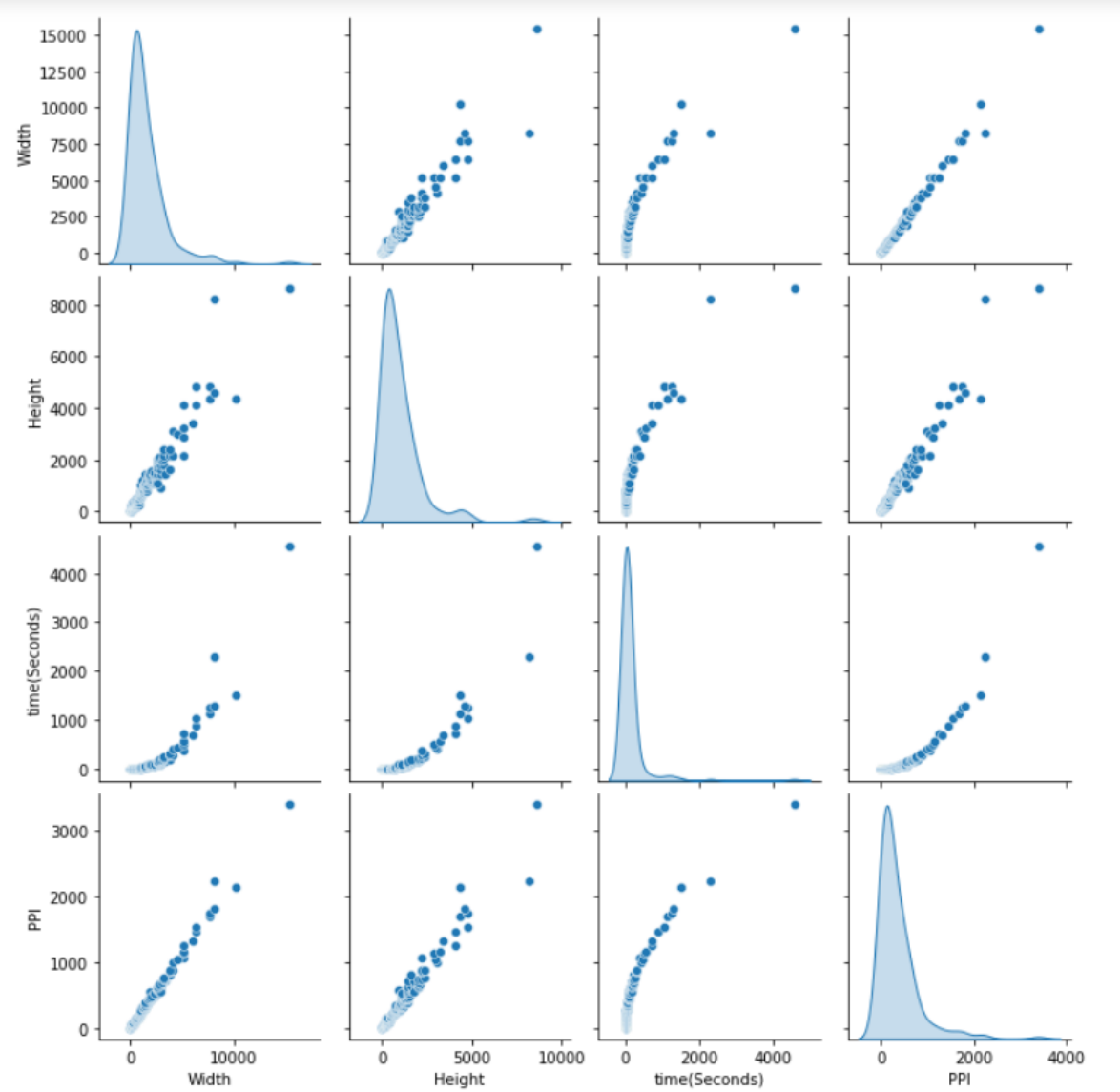
Assumptions:

1. Bluetooth 1.0 interface
2. No compression of the image
3. Egyptian National ID image
4. PPI was calculated using the dimensions of 1 screen
5. The image was scanned using the camera of of 1 phone

Width	Height	size(MB)	time(Seconds)	PPI
1440	1080	4.666	53	346.15
1600	1024	4.915	56	365.3
1680	1050	5.292	60	381
1776	1000	5.328	60	391.96
1600	1200	5.76	65	384.6
1600	1280	6.144	70	394
1920	1080	6.221	71	423.6
1440	1440	6.221	71	391.6
2048	1080	6.636	75	445.25
1920	1200	6.912	78	555.8
2160	1080	6.998	79	464.4
2048	1152	7.078	80	451.9
1792	1344	7.172	81	430.8
1920	1280	7.373	84	443.8
2280	1080	7.387	84	485.2
1856	1392	7.75	88	446.15
1800	1440	7.776	88	443.3
2880	900	7.776	88	580.3
2160	1200	7.776	88	475.2
2048	1280	7.864	89	464.4
1920	1400	8.064	92	457
2436	1125	8.222	93	516
2538	1080	8.223	93	530.4
1920	1440	8.294	94	461.5
2560	1080	8.294	94	534.3
2160	1440	9.331	106	499.2
2048	1536	9.437	107	492.3
2304	1440	9.953	113	522.5

Data Analysis

To have some knowledge about the data before creating the model is very important, so simple data analysis was done by plotting the features of the dataset against each other to have some sense about how the model would look like



Surrogate Model

A Surrogate model is a model used if the adressed problem is hard to get solutions using analytic methods, like here in the case of this problem that we cannot find an equation that defines time and quality given width and height, so the Surrogate model used was an artificial neural network regressor, a simple linear regression model didn't yield great results because as we can see in the plots above the time and PPI are not linear functions of width and time, so a more complicated model was required.

After exhaustive model researching to get the best neural network that fits the data best, the model chosen was a neural network with 2 input features, width and height and several hidden layers and then 2 outputs, transfer time in seconds and quality in PPI

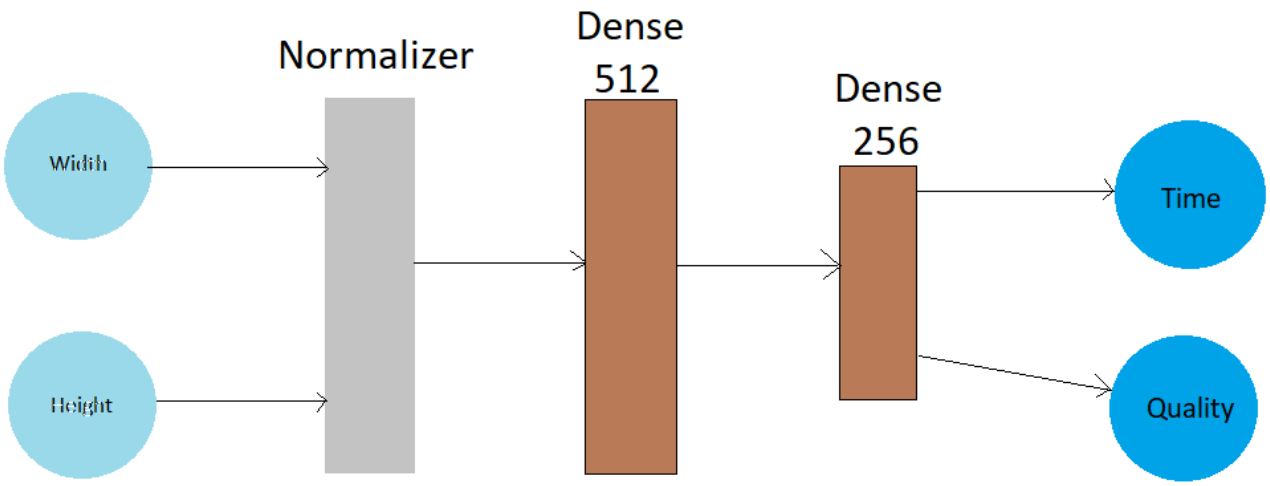
Model description:

To get the best results, a normalization to the features feeding into the model is required, so a normalizing layer was placed right after the input layer that takes the features and normalize them, to make the scale of the training features between 0 and 1, the normalization is adapted to the training features only, without the test features because the test data should be for testing only and the model shouldn't have any estimate of how the test data look like.

After exhaustive research of the hyperparameters, the hyperparameters choses were:

- Adam optimizer with learning rate of 0.001, beta1=0.9,beta2=0.999 and epsilon=10^{-8}
- Mean Absolute Error loss function
- 2 hidden layers with 512 and 256 neurons respectively with activation function ReLU on both

So the blueprint of out model looks like this:



Surrogate Model

The optimization function used for this model was Adam which uses exponential weighted averages of gradients and exponential weighted averages of the square of the gradients to compute the new gradient, and it's a form of stochastic gradient descent, the new weights is caluclated using the below formula:

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$

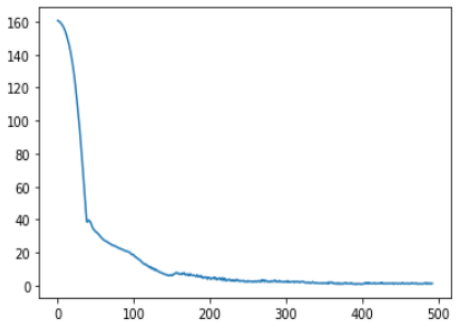
$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

The model training was set to a maximum of 2000 epochs over the training data with a cross-validation split of 20% of the training data, to make get a rough sense of how the model would perform on the validation set.

An Early Stop callback function was used to ensure that the model doesn't overfit the data, with a patience of 100 iterations that if the cross-validation loss didn't decrease in 100 iterations the model stops training and it returns the weights that resulted in the minimum cross-validation loss, this plot shows how the loss function decreases over the iterations.

```
In [326]: hist_train=model.fit(train_features,train_labels,validation_split=0.2,epochs=2000,callbacks=[early_stop])
3/3 [==============================] - 0s 5ms/step - loss: 7.7174 - val_loss: 1.7554
Epoch 485/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.7096 - val_loss: 1.5421
Epoch 486/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.5436 - val_loss: 0.9068

Epoch 487/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.4103 - val_loss: 1.6370
Epoch 488/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.8512 - val_loss: 1.3696
Epoch 489/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.2858 - val_loss: 1.4862
Epoch 490/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.6269 - val_loss: 1.0470
Epoch 491/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.5226 - val_loss: 1.4205
Epoch 492/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.4151 - val_loss: 1.1664
Epoch 493/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.3551 - val_loss: 1.4021
```



After the model finished training, the validation set is used to evaluate the model, this procedure was done many times to ensure that the final model design performed the best out of all the models tried and the loss of the final model is acceptable, and then when the final model is chosen, then we train the model furthermore on the validation data to further improve the model, but this time since the validation data is less than the training data we define make a new callback that has patience over 50 iterations only.

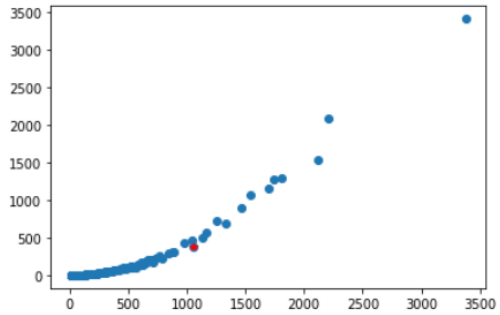
Multi-Objective Optimization

There exists a conflict of interests between transfer time and quality as we explained before, so we need to find the best trade-off point, that compromises between transfer time and quality, assuming that there are no boundaries to neither transfer time nor quality.

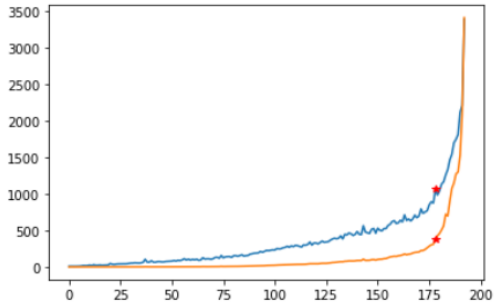
The multiobjective optimization solution used was weighted sum of objective functions, and the weight was chosen by exhaustive brute force to find the point that was least changed by trying different weights.

To maximize a function is to minimize its inverse, so for the sake of simplicity our 2 objectives are minimizing the transfer time and minimizing the inverse of the quality, so a weight was placed to the quality objective starting by -10000, because prior to that weight, it was found that the results were not pleasing, and down to a weight of -1500000, and it was found that the best trade-off point in the dataset is 5120x2160, as this point was stable from the weight value of -400000 to -1500000 and that is 110 iterations with a steo of -10000

Scatter plot of quality on x-axis and time on the y axis



Plot of quality (Blue) and time (orange) through the iterations



The red dots mark the optimal point's quality and transfer time, and the plot on the right shows that the point chosen, truly compromises between quality and time with no boundaries placed as by increasing the quality more, it will lead to significally greater time (orange plot) and by decreasing the time more it will lead to significally less quality (blue plot).

Code

Task 1: Creating the Surrogate Model

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input,Dense
from keras.callbacks import EarlyStopping
from tensorflow.keras.layers.experimental import preprocessing
import pandas as pd
import seaborn as sns
```

Loading the dataset and splitting it to 60% training and 40% validation

In [2]:
```
dataset=pd.read_csv("./Desktop/ds_opt/dataset2.csv")
```

In [3]:
```
dataset['Res']=dataset['Width']*dataset['Height']
dataset
```

Out[3]:

	Resolution	Width	Height	size(MB)	time(Seconds)	PPI	PPI(Normalized)	Res
0	16x16	16	16	0.000768	0	4.351	0.000000	256
1	42x11	42	11	0.001386	0	8.350	0.001182	462
2	32x32	32	32	0.003072	0	8.703	0.001286	1024
3	40x30	40	30	0.003600	0	9.615	0.001555	1200
4	42x32	42	32	0.004032	0	10.154	0.001715	1344
...	...	...	...	...	...	...	...	...
188	7680x4800	7680	4800	110.600000	1264	1741.700	0.513303	36864000
189	8192x4608	8192	4608	113.250000	1294	1807.500	0.532743	37748736
190	10240x4320	10240	4320	132.700000	1516	2137.300	0.630183	44236800
191	8192x8192	8192	8192	201.330000	2300	2228.000	0.656981	67108864
192	15360x8640	15360	8640	398.100000	4549	3389.000	1.000000	132710400

193 rows × 8 columns

In [4]:
```
train_data=dataset.sample(frac=0.6,random_state=0)
test_data=dataset.drop(train_data.index)
```

Sanity check that train and test set are unique

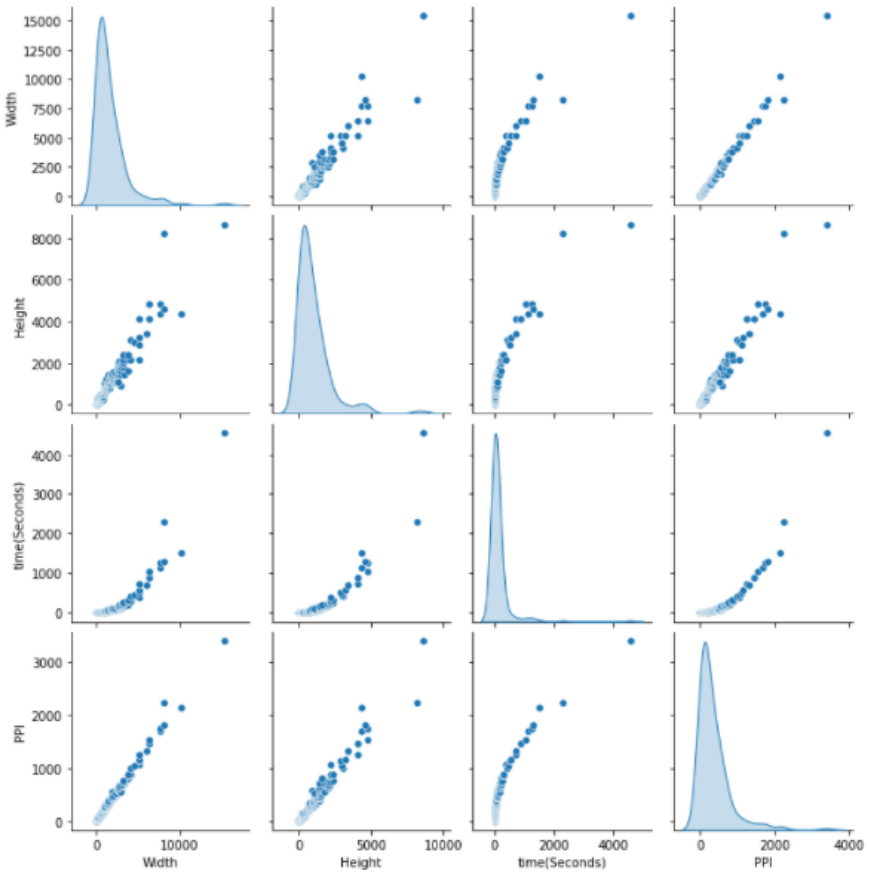
In [5]:
```
print(test_data.merge(train_data).empty)
```

```
True
```

Code

As we can see below, the time and quality are non-linear functions of Width and Height

```
In [6]: sns.pairplot(dataset[['Width','Height','time(Seconds)','PPI']],diag_kind='kde')
Out[6]: <seaborn.axisgrid.PairGrid at 0x25658772ca0>
```



Code

Getting rid of useless features, because we are interested in the Width and Height only

In [8]:
```
train_data.pop('PPI(Normalized)')
train_data.pop('size(MB)')
train_data.pop('Resolution')
train_data.pop('Res')
test_data.pop('PPI(Normalized)')
test_data.pop('size(MB)')
test_data.pop('Resolution')
test_data.pop('Res')
```

Out[8]:
```
1           462
3          1200
6          2400
9          4608
11         4800
          ...
177     9216000
182    16384000
184    20358144
187    33177600
191    67108864
Name: Res, Length: 77, dtype: int64
```

Separate features from labels in both Training and Test set

In [9]:
```
train_features=train_data.copy()
test_features=test_data.copy()
train_labels1=train_features.pop('time(Seconds)').tolist()
train_labels2=train_features.pop('PPI').tolist()
test_labels1=test_features.pop('time(Seconds)').tolist()
test_labels2=test_features.pop('PPI').tolist()

train_labels=np.concatenate((np.array(train_labels1).reshape((len(train_features),1)),np.array(train_labels2).reshape(len(train_
test_labels=np.concatenate((np.array(test_labels1).reshape((len(test_features),1)),np.array(test_labels2).reshape(len(test_featur
```

Creating the model

In [10]:
```
normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(train_features))
#Feature normalization layer that normalizes the training features
```

Code

Creating the model

In [10]:
```
normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(train_features))
#Feature normalization layer that normalizes the training features
```

In [11]:
```
model=keras.Sequential([
    normalizer,
    Dense(256,activation='relu'),
    Dense(512,activation='relu'),
    Dense(2)
])
model.compile(optimizer='adam',loss='MeanAbsoluteError')
early_stop=EarlyStopping(monitor='val_loss',mode='min',patience=100,restore_best_weights=True)
early_stop2=EarlyStopping(monitor='loss',mode='min',patience=50,restore_best_weights=True)
```

Training the model using a callback function for early stopping to avoid overfitting, and using 20% of the training set as a cross-validation set

In [326]:
```
hist_train=model.fit(train_features,train_labels,validation_split=0.2,epochs=2000,callbacks=[early_stop])
```

```
Epoch 461/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.7563 - val_loss: 1.1408
Epoch 462/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.9688 - val_loss: 1.3626
Epoch 463/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.9155 - val_loss: 1.1776
Epoch 464/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.7167 - val_loss: 1.2536
Epoch 465/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.6014 - val_loss: 1.5567
Epoch 466/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.7269 - val_loss: 1.2917
Epoch 467/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.6936 - val_loss: 1.5987
Epoch 468/2000
3/3 [==============================] - 0s 5ms/step - loss: 7.6257 - val_loss: 1.4026
Epoch 469/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.7317 - val_loss: 1.2832
Epoch 470/2000
3/3 [==============================] - 0s 6ms/step - loss: 7.4581 - val_loss: 1.3770
```

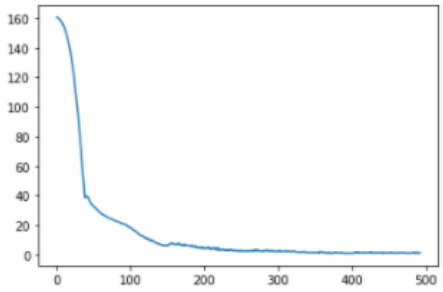
In [330]:
```
hist_acc=model.evaluate(test_features,test_labels)
```

```
3/3 [==============================] - 0s 997us/step - loss: 3.5949
```

The loss is acceptable, so we can now train the model on the test set so it can improve even more

Code

```
In [328]: plt.plot(range(len(hist_train.history['val_loss'])),hist_train.history['val_loss'])
Out[328]: [<matplotlib.lines.Line2D at 0x123e11934f0>]
```




```
In [329]: model.fit(test_features,test_labels,epochs=1000,callbacks=[early_stop],validation_split=0.2)
2/2 [==============================] - 0s 7ms/step - loss: 0.9009 - val_loss: 17.5342
Epoch 89/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8774 - val_loss: 17.6868
Epoch 90/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8872 - val_loss: 17.7257
Epoch 91/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8740 - val_loss: 17.6464
Epoch 92/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8792 - val_loss: 17.5506
Epoch 93/1000
2/2 [==============================] - 0s 8ms/step - loss: 0.8735 - val_loss: 17.4235
Epoch 94/1000
2/2 [==============================] - 0s 8ms/step - loss: 0.8621 - val_loss: 17.6694
Epoch 95/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8564 - val_loss: 17.8209
Epoch 96/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8798 - val_loss: 17.8184
Epoch 97/1000
2/2 [==============================] - 0s 7ms/step - loss: 0.8880 - val_loss: 17.6796
```

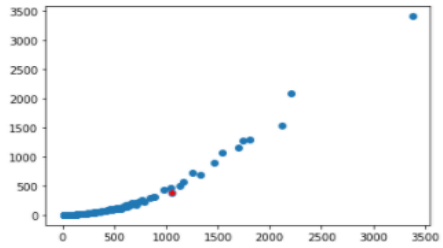
Code

Task 2: Finding the best trade-off (Multi-objective optimization)

In [355]:
```
#This cell was made using exhaustive brute force of the weight below on the PPI (Quality), and it was concluded that by
#increasing the weight of the PPI objective function, it changes the resolution, but between -500,000 and -1,500,000
#The minimum weighted sum remained the same, and though we achieved the best trade-off with no boundaries on either quality
#nor time.
#The method used is weighted sum that minimizes time and maximizes quality, so to make it simpler, the weighted sum was used to
#minimize time and minimize the inverse of quality.

list_time=[]
list_ppi=[]
list_res=[]
min_sum=10000000
min_i=0
min_j=0
k=0
for i,j in zip(dataset['Width'],dataset['Height']):
        arr=model.predict([[i,j]])
        ppi_now=-1./arr[0][1].squeeze()
        time_now=arr[0][0].squeeze()
        list_time.append(max(0,time_now))
        list_res.append((i,j))
        list_ppi.append(ppi_now)
        i_sum=(-1500000*ppi_now)+time_now
        if i_sum<min_sum:
            min_sum=i_sum
            min_i=i
            min_j=j
```

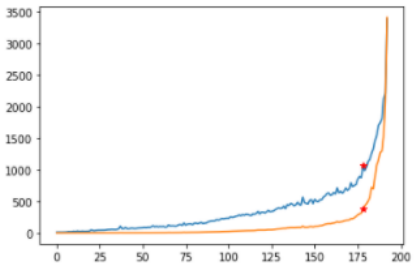
In [356]:
```
list_ppi=-1./np.array(list_ppi)
ind=list_res.index((min_i,min_j))
plt.scatter(list_ppi,list_time)
plt.plot(list_ppi[ind],list_time[ind],"r*")

##A scatter plot of time and quality and the red dot represents the best tradeoff
```

Out[356]: [<matplotlib.lines.Line2D at 0x123eb957a90>]



In [357]:
```
plt.plot(range(len(list_ppi)),list_ppi)
plt.plot(range(len(list_time)),list_time)
plt.plot(ind,list_time[ind],"r*")
plt.plot(ind,list_ppi[ind],"r*")
##A plot of time and quality and the red dots represents the best tradeoff time and quality
```

Out[357]: [<matplotlib.lines.Line2D at 0x123eadc43a0>]



In [361]:
```
arr=model.predict([[5120,2160]])
time=arr[0][0].squeeze()
qua=arr[0][1].squeeze()
print("Best trade-off resolution is: "+str(min_i)+"x"+str(min_j)+" that takes time= "+str(time)+" seconds and quality= "+str(qua
```

```
Best trade-off resolution is: 5120x2160 that takes time= 381.56366 seconds and quality= 1058.0343 PPI(Pixels per inch)
```