Performance Analysis of Machine Learning Software

October 31, 2019

1 Performance analysis of Machine Learning (ML) software

[1]:

```
#import modules
import pickle
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import statsmodels as sm
import seaborn as sns
```

1.1 Acquire Data

We have 2 Dataframes: * Test info * Test results

[2]:

```
#import dataframes
test_info=pd.read_csv("./TestInfo.csv")
file = open("./TestResults.pickle",'rb')
test_results=pickle.load(file,encoding='latin1')
```

The dataframe have in common one field= “TestId”, we can join them

[3]:

```
df=pd.merge(test_info, test_results)#.sort_values(by=["TestId"])
df.head()

fig=0
```

We can study what is inside the dataframe

[4]:

```
df.describe()
```

[4]:

```
          TestId  CPUFrequency (MHz)    Threads      Build   Time (ms)  \
count  47.000000           47.000000  47.000000  47.000000   47.000000
mean   23.000000         1297.872340   4.148936   7.127660  128.801418
std    13.711309          462.267268   1.366982   3.187046  149.035930
min     0.000000         1000.000000   1.000000   1.000000   19.000000
25%    11.500000         1000.000000   3.500000   4.500000   51.500000
50%    23.000000         1000.000000   5.000000   8.000000   83.333333
```

```
75%    34.500000         2000.000000   5.000000  10.000000  114.500000
max    46.000000         2000.000000   5.000000  10.000000  800.000000

       PeakMemory (MB)
count        47.000000
mean        383.234043
std         125.454168
min          50.000000
25%         309.000000
50%         450.000000
75%         454.500000
max         460.000000
```

[5]:

```
df.head()
```

[5]:

```
   TestId    Device  CPUFrequency (MHz)  Threads MLNetwork  Build Optimised  \
0      17  Device_0                1000        5   AlexNet      9         N
1      16  Device_0                1000        5   AlexNet      8         N
2      39  Device_1                1000        3   AlexNet     10         N
3      31  Device_0                1000        5   AlexNet     10         N
4      30  Device_0                1000        4   AlexNet     10         N

    Time (ms)  PeakMemory (MB)
0  102.000000              449
1  104.000000              453
2  333.333333              302
3  100.000000              449
4  125.000000              450
```

[6]:

```
print("Devices:",df["Device"].unique())
print("MLNetworks:",df["MLNetwork"].unique())
print("Optimization:",df["Optimised"].unique())
print("CPUFrequencies (MHz):",df["CPUFrequency (MHz)"].unique())
print()
print("Min and Max of Builds:",min(df["Build"]),"-",max(df["Build"]))
print("Min and Max of TestId: ",min(df["TestId"]),"-",max(df["TestId"]))
print("Min and Max of Peak Memory: ",min(df["PeakMemory␣
 ↪(MB)"]),"-",max(df["PeakMemory (MB)"]))
print("Min and Max of Time: ",min(df["Time (ms)"]),"-",max(df["Time (ms)"]))
print("Min and Max of Threads:",min(df["Threads"]),"-",max(df["Threads"]))
```

```
Devices: ['Device_0' 'Device_1']
MLNetworks: ['AlexNet' 'MobileNet']
Optimization: ['N' 'Y']
CPUFrequencies (MHz): [1000 2000]

Min and Max of Builds: 1 - 10
Min and Max of TestId:  0 - 46
Min and Max of Peak Memory:  50 - 460
Min and Max of Time:  19.0 - 800.0
Min and Max of Threads: 1 - 5
```

1.2 Correlation

To help with Time and Memory we can create a new column, named Performance that is higher when both time and Memory are lower

Performance(1/(B*s*)) = (1/(Memory *Time*)) 1000 *1000

[7]:

```
df["Performance"]=(1/(df["PeakMemory (MB)"]*df["Time (ms)"])*1000 *1000)

df=df.sort_values(by=["Performance"],ascending=False)
```

[8]:

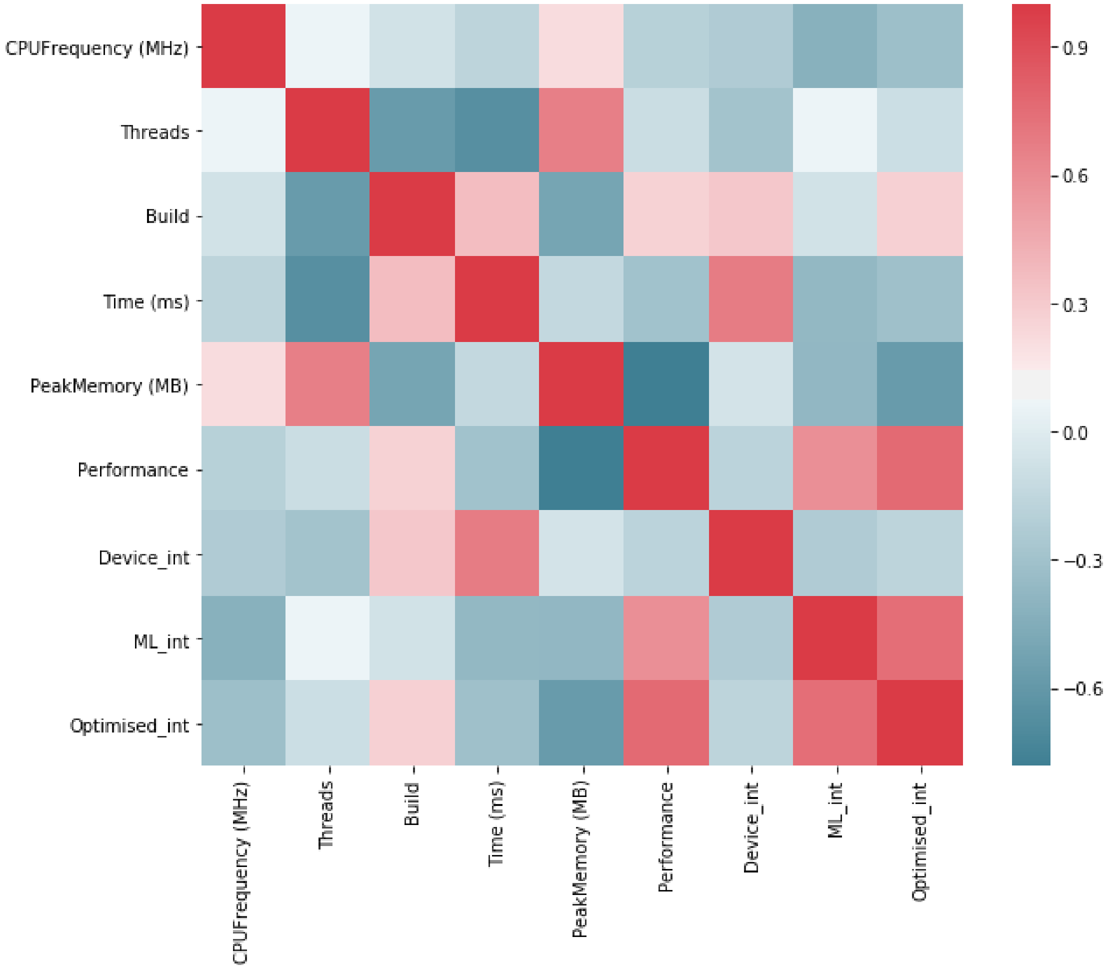
```
#Classification
df1=df
df1.loc[df["Device"] ==  "Device_0", 'Device_int'] = 0
df1.loc[df["Device"] ==  "Device_1", 'Device_int'] = 1
df1.loc[df["MLNetwork"] ==  "AlexNet", 'ML_int'] = 0
df1.loc[df["MLNetwork"] ==  "MobileNet", 'ML_int'] = 1
df1.loc[df["Optimised"] ==  "Y", 'Optimised_int'] = 1
df1.loc[df["Optimised"] ==  "N", 'Optimised_int'] = 0
df1=df1.drop(["TestId","MLNetwork","Device","Optimised"],1)
alex=df[df["MLNetwork"]=="AlexNet"]
mobile=df[df["MLNetwork"]=="MobileNet"]
```

[9]:

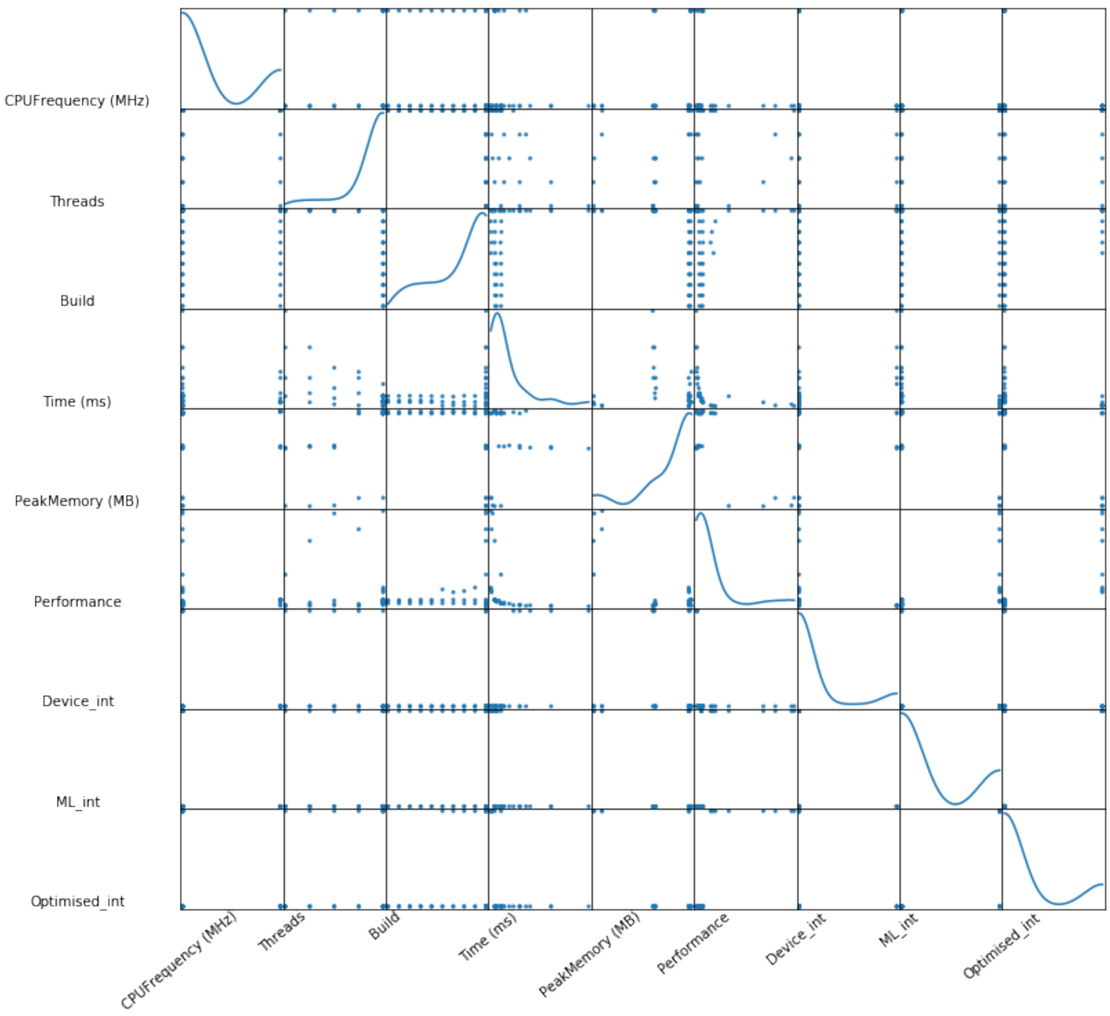
```
f, ax = plt.subplots(figsize=(10, 8))
corr = df1.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.
 ↪diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```

[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb7e21d46d0>
```



[10]:
```
sm = pd.plotting.scatter_matrix(df1, alpha=0.9, figsize=(12, 12),␣
 ↪diagonal='kde')

#Change label rotation
[s.xaxis.label.set_rotation(40) for s in sm.reshape(-1)]
[s.yaxis.label.set_rotation(0) for s in sm.reshape(-1)]

#May need to offset label when rotating to prevent overlap of figure
[s.get_yaxis().set_label_coords(-1.0,0.0) for s in sm.reshape(-1)]
[s.get_xaxis().set_label_coords(0.0,0.0) for s in sm.reshape(-1)]

#Hide all ticks
[s.set_xticks(()) for s in sm.reshape(-1)]
[s.set_yticks(()) for s in sm.reshape(-1)]

plt.show()
```



1.3 Findings:

- 47 Tests
- 2 MLNetworks (AlexNet, MobileNet)
- 2 Devices (Device_0, Device_1)
- 2 CPU Frequencies (1000, 2000)
- 2 Optimised Status (Yes or No)
- 5 Use of Threads (from 1 to 5)
- Peak Memory in MB (from 50 to 460)
- Time in ms (from 19 to 800)
- From heatmap and scatter matrix we can see that there is **correlation** between:

- **High**
- Threads and Memory -> More Threads use more memory

- Time and Device -> Device_0 uses more time
- ML and optimization -> Only MobileNet is tried optimized
- Performance and ML -> MobileNet is performing better
- Performance and Optimization -> Optimizing impact a lot on performance
- **Low**
- CPUFreq and Memory
- Build and Optimization
- Build and Device
- Build and Time
- Build and Performance

- As it is a Performance analysis of Machine Learning (ML) software the main point would be **maximization of Performance, minimizing Time and Memory**, so we should look on Time and Memory

Performance is maximized if: * More Thread are used * CPU Freq is Higher * Build are higher * Time and Memory are lower * Mobile Net is used * ML is optimized

1.3.1 Comparing the two ML Networks

```
[11]: for col in df.columns.drop(["Performance",'Device_int', 'ML_int',␣
       ↪'Optimised_int']):
          plt.
       ↪scatter(df[df["MLNetwork"]=="AlexNet"][col],df[df["MLNetwork"]=="AlexNet"]["Performance"],a
       ↪= 0.7)
          plt.
       ↪scatter(df[df["MLNetwork"]=="MobileNet"][col],df[df["MLNetwork"]=="MobileNet"]["Performance
       ↪= 0.7)
          fig+=1
          plt.title("Fig."+str(fig)+" Performance")
          plt.ylabel("Performance")
          plt.grid(b=True)
          plt.xlabel(col)
          plt.legend(["Performance AlexNet","Performance MobileNet"])
          plt.show()
```

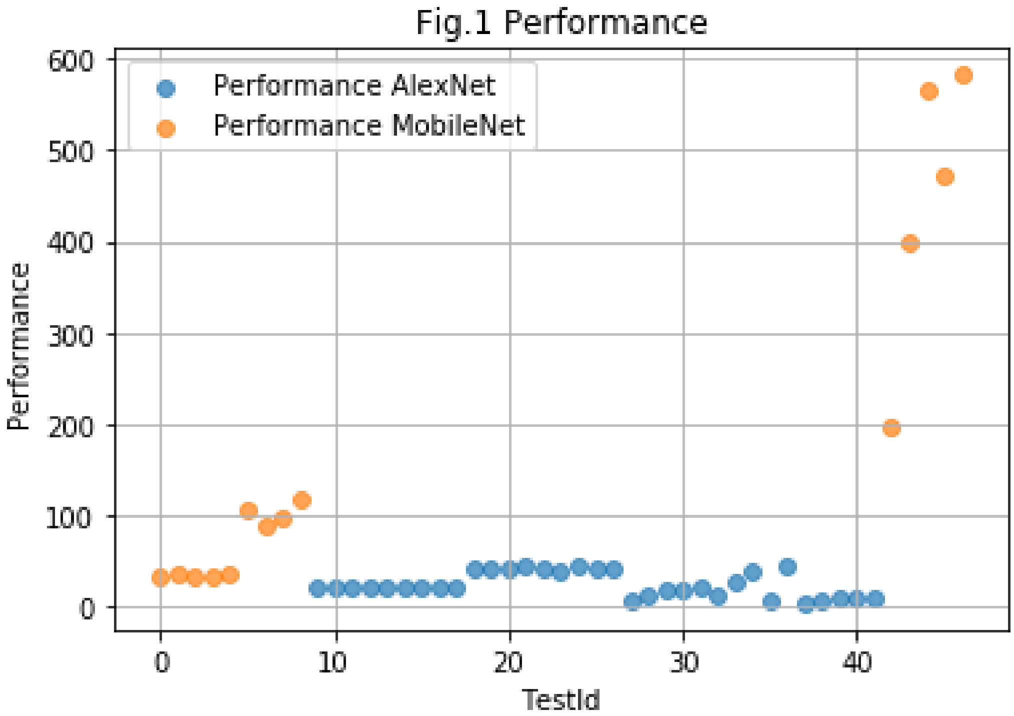


Fig.1 Performance
Performance AlexNet
Performance MobileNet
Performance
600
500
400
300
200
100
0
0
10
20
30
40
TestId

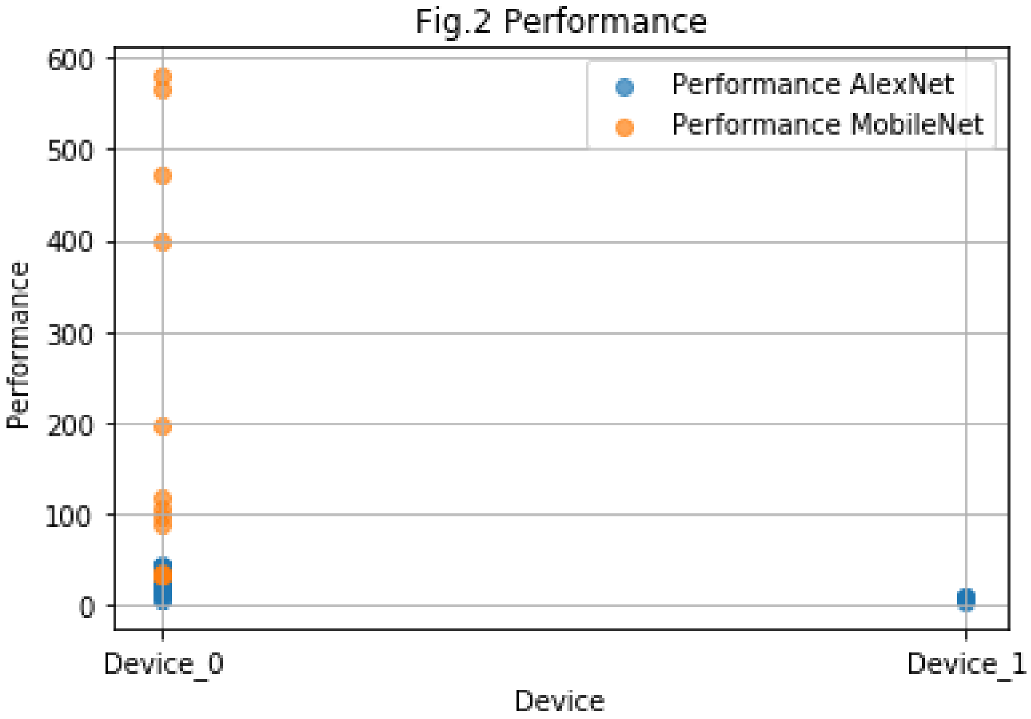


Fig.2 Performance
Performance AlexNet
Performance MobileNet
Performance
600
500
400
300
200
100
0
Device_0
Device_1
Device

Fig.3 Performance

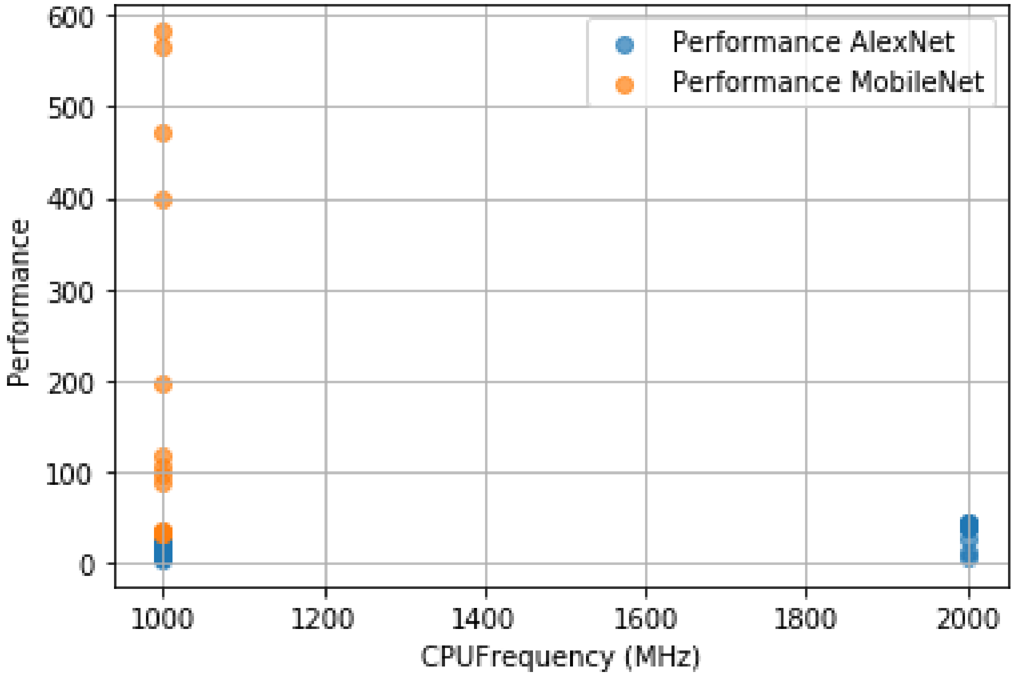
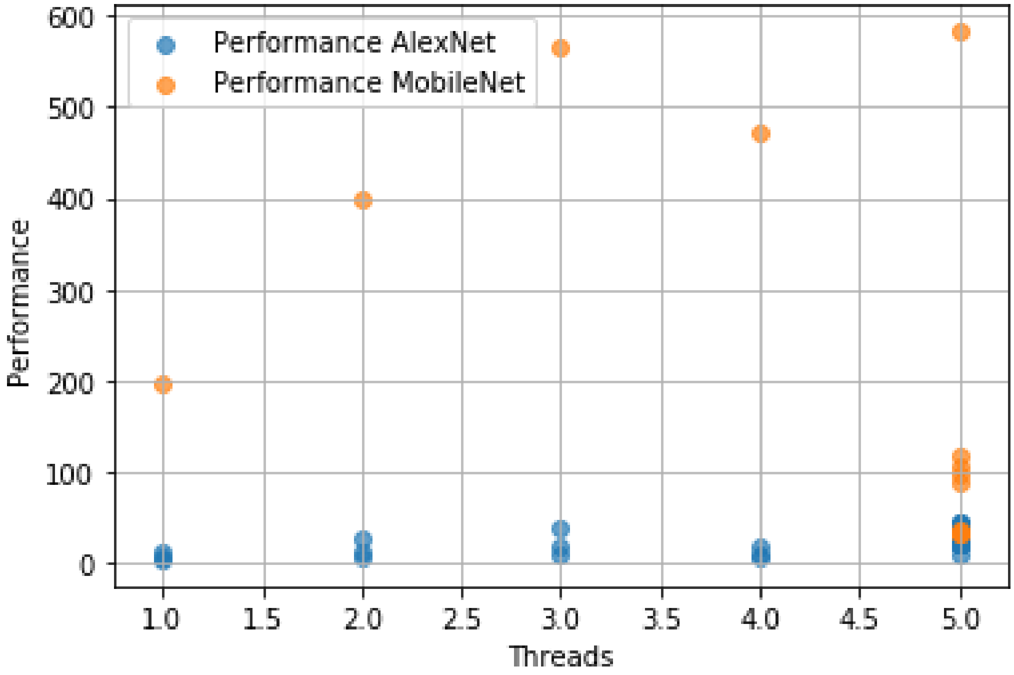


Fig.4 Performance



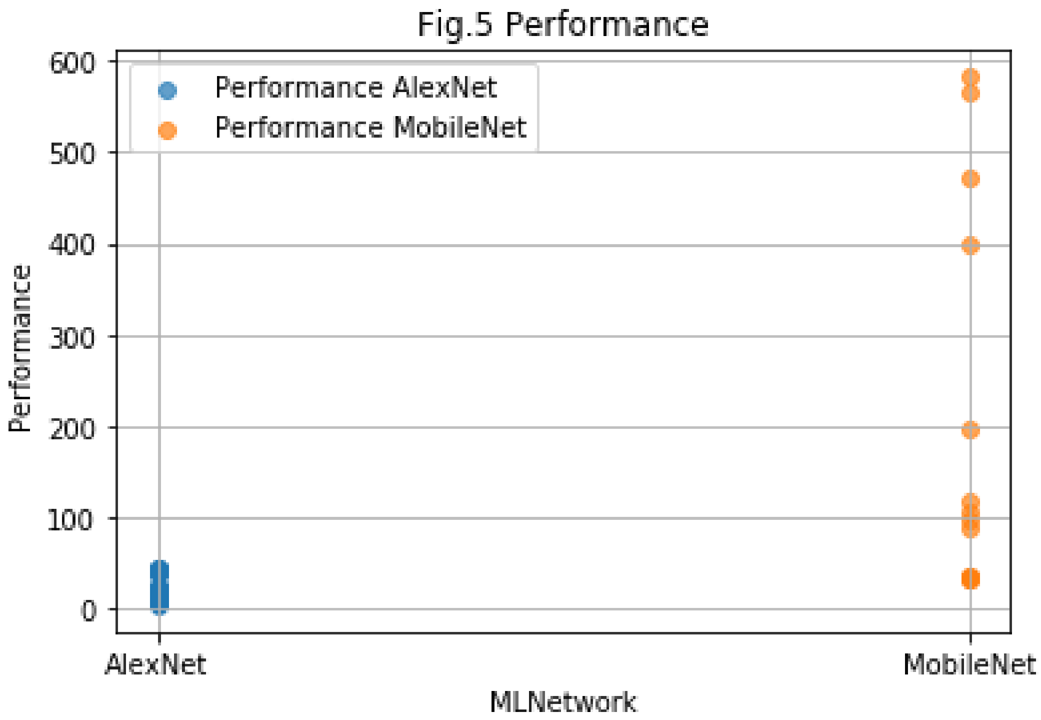


Fig.5 Performance
Performance AlexNet
Performance MobileNet
600
500
400
300
200
100
0
Performance
AlexNet
MobileNet
MLNetwork

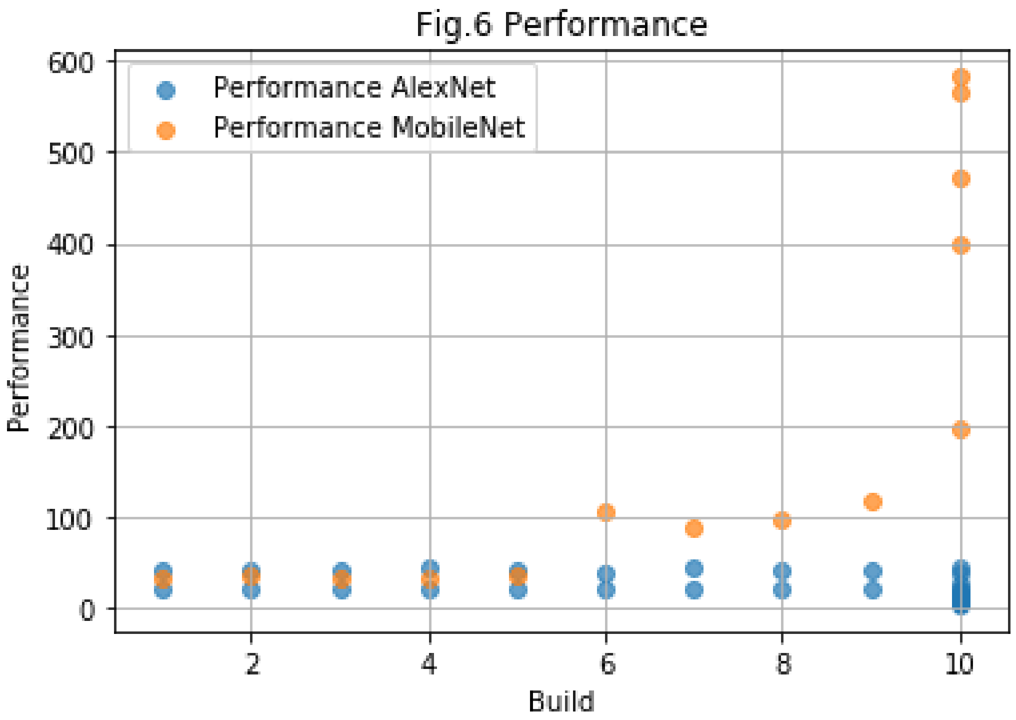


Fig.6 Performance
Performance AlexNet
Performance MobileNet
600
500
400
300
200
100
0
Performance
2
4
6
8
10
Build

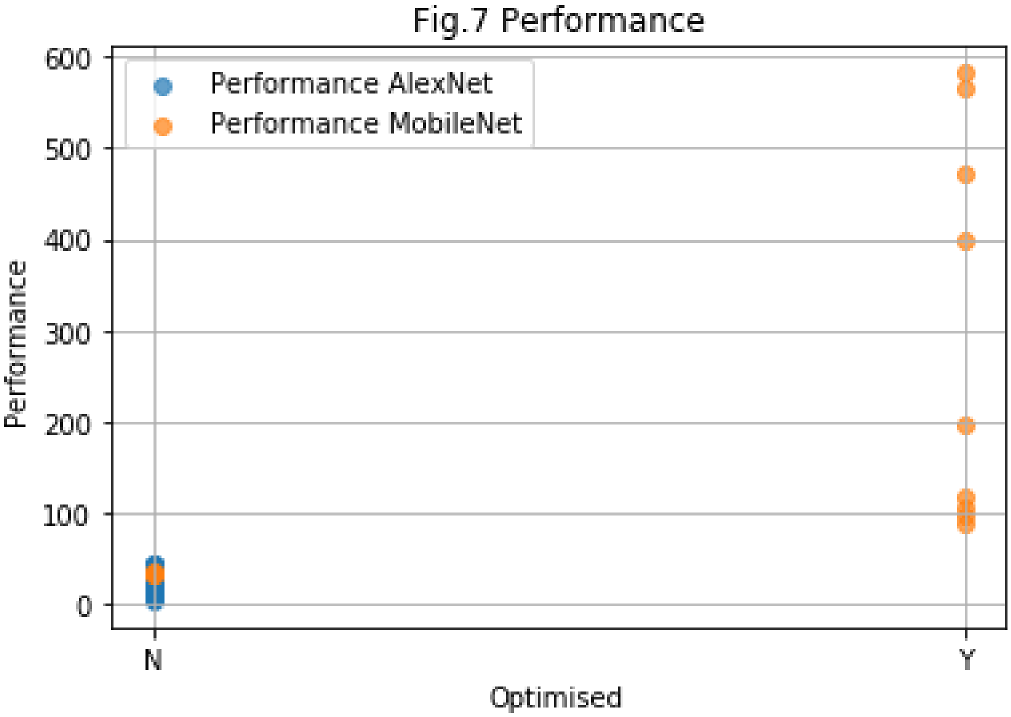


Fig.7 Performance
Performance
600
500
400
300
200
100
0
Performance AlexNet
Performance MobileNet
N
Y
Optimised

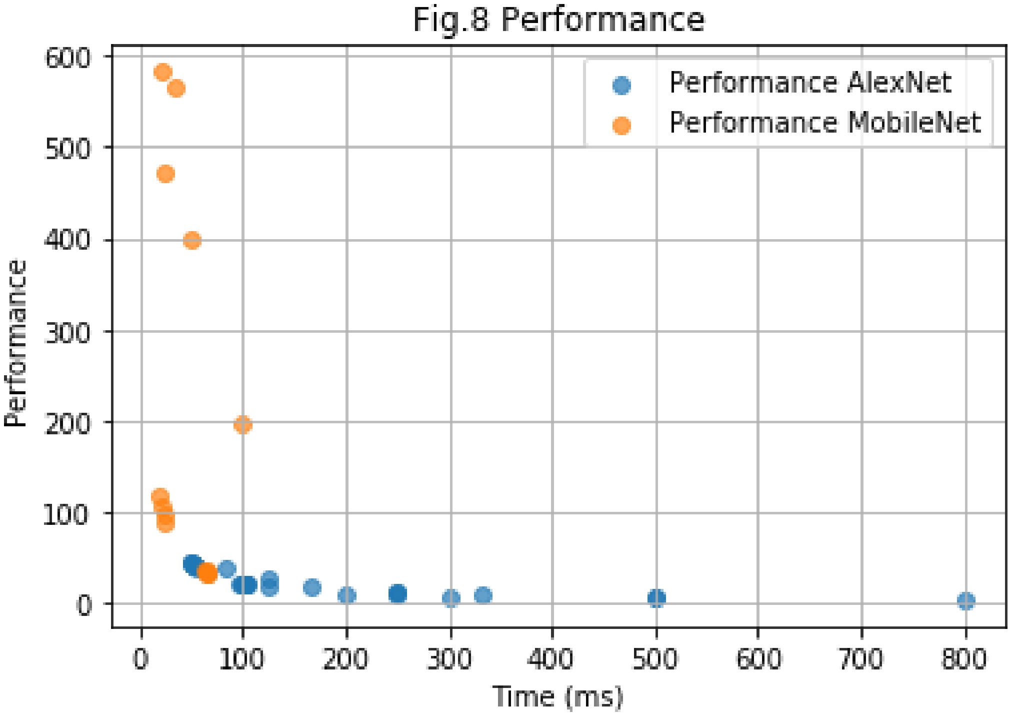
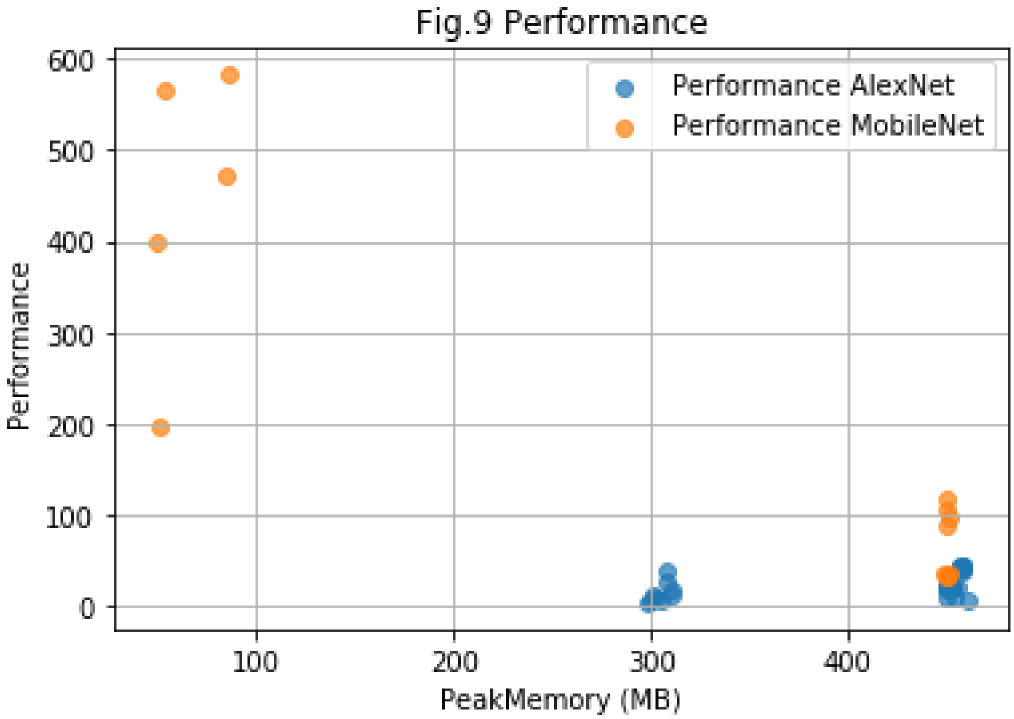


Fig.8 Performance
Performance
600
500
400
300
200
100
0
Performance AlexNet
Performance MobileNet
0
100
200
300
400
500
600
700
800
Time (ms)

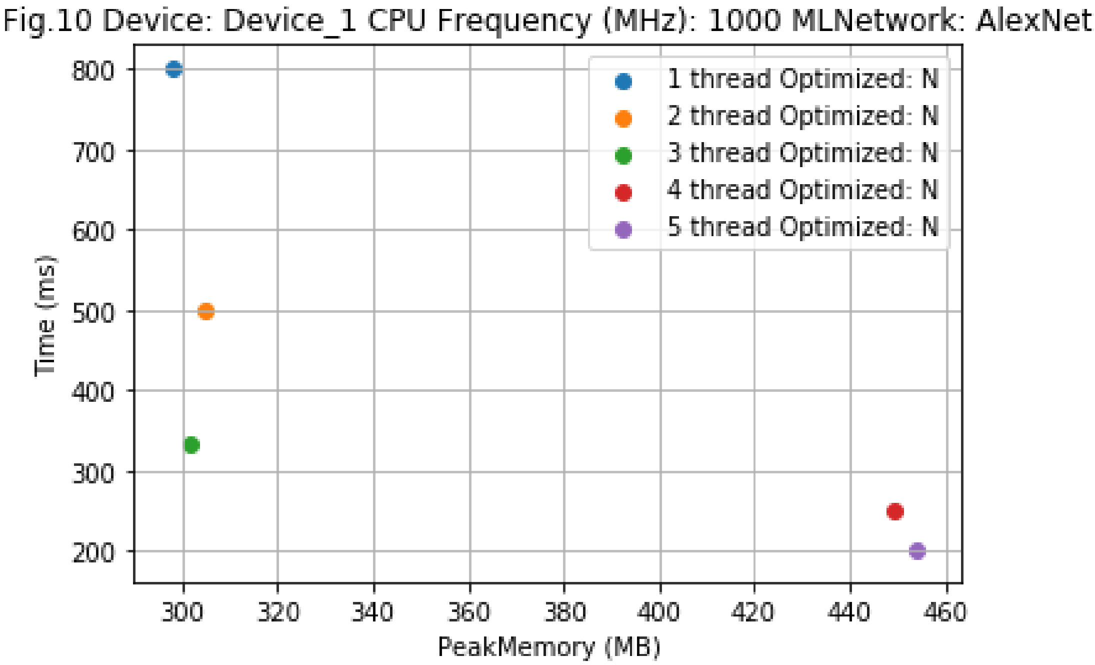


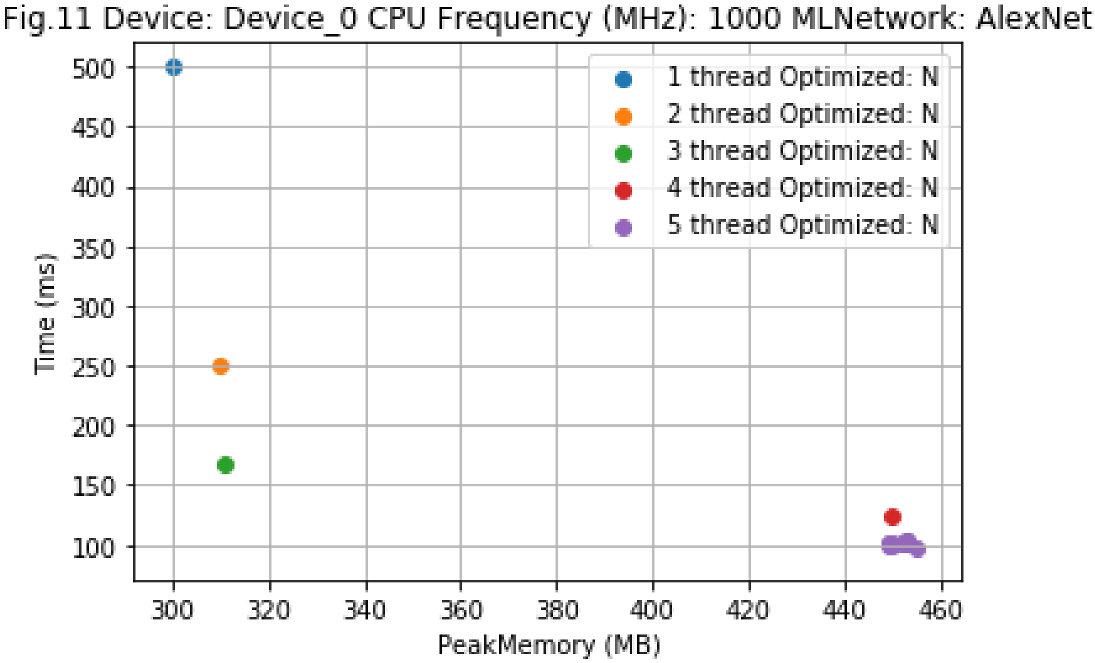
1.3.2 AlexNet

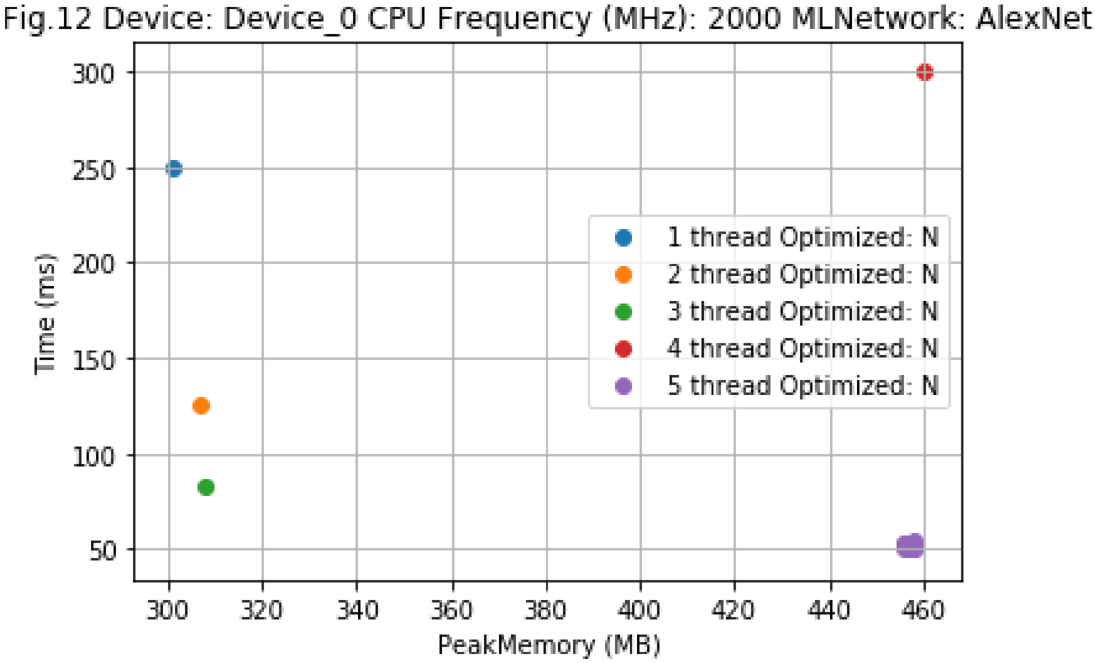
Findings

- AlexNet is not optimized
- More Threads uses more memory but less time
- More CPU Frequency means less time
- To minimize time and space we can check which value is closer to 0 in figure 13,14,15 Overall the best option performance wise is to use 5 threads, if memory is a constraint we can lowe the number of threads

[12]:
```
legend=[]
alex=alex.sort_values(by=["Threads"])

for o in alex["Optimised"].unique():
    for t in alex["Threads"].unique():
        legend.append(str(t)+" thread"+" Optimized: "+o)
for device in alex["Device"].unique():
    for cpu_freq in alex["CPUFrequency (MHz)"].unique():
        for optimised in alex["Optimised"].unique():
            for thread in alex["Threads"].unique():

                tmp=alex[alex["Threads"]==thread]
                tmp=tmp[tmp["CPUFrequency (MHz)"]==cpu_freq]
            tmp=tmp[tmp["Optimised"]==optimised]
            tmp=tmp[tmp["Device"]==device]
            if not tmp.empty:
                plt.scatter(tmp["PeakMemory (MB)"],tmp["Time (ms)"])

    if not tmp.empty:
        fig+=1;
        plt.title(("Fig."+str(fig)
                   +" Device: "+device
                   +" CPU Frequency (MHz): "+ str(cpu_freq)
                   +" MLNetwork: AlexNet" ))
        plt.grid(b=True)
        plt.xlabel("PeakMemory (MB)")
        plt.ylabel("Time (ms)")
        plt.legend(legend)
        plt.show()
```







1.3.3 MobileNet

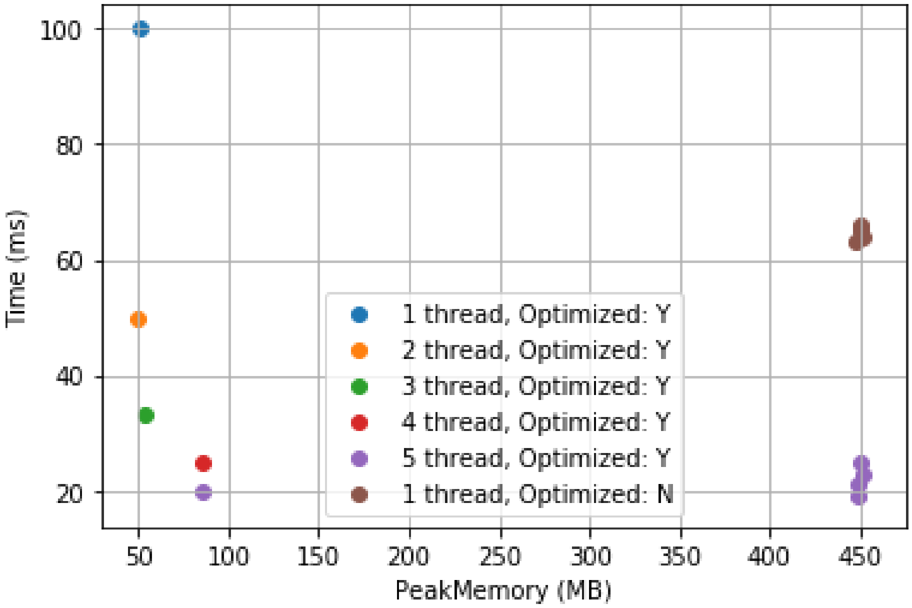
Findings

- MobileNet is tested only on Device_0
- MobileNet is tested only on CPUFreq: 1000MHz
- Optimization is more efficient than not optimised
- Best case: 5 Thread, Optimised

[13]:

```python
legend=[]
mobile=mobile.sort_values(by=["Threads"])
for o in mobile["Optimised"].unique():
    for t in mobile["Threads"].unique():
        legend.append(str(t)+" thread"+", Optimized: "+o)


for device in mobile["Device"].unique():
    for cpu_freq in mobile["CPUFrequency (MHz)"].unique():
        for optimised in mobile["Optimised"].unique():
            for thread in mobile["Threads"].unique():

                tmp=mobile[mobile["Threads"]==thread]
                tmp=tmp[tmp["CPUFrequency (MHz)"]==cpu_freq]
                tmp=tmp[tmp["Optimised"]==optimised]
                tmp=tmp[tmp["Device"]==device]
                if not tmp.empty:
                    plt.scatter(tmp["PeakMemory (MB)"],tmp["Time (ms)"])

        if not tmp.empty:
            fig+=1;
            plt.grid(b=True)
            plt.title(("Fig."+str(fig)
                       +" Device: "+device
                       +" CPU Frequency (MHz): "+ str(cpu_freq)
                       +" MLNetwork: MobileNet" ))
            plt.xlabel("PeakMemory (MB)")
            plt.ylabel("Time (ms)")
            plt.legend(legend)
            plt.show()
```

Fig.13 Device: Device_0 CPU Frequency (MHz): 1000 MLNetwork: MobileNet



1.4 Follow Up Actions

Talk with the engineers and show that * Tests are promising, they should try to test: * CPU Freq: 1000 and 2000MHz on Device_1 with MobileNet and 5 thread * AlexNet optimized

- MobileNet seems very promising as performance compared to AlexNet both for time and memory
- Optimization is fundamental, AlexNet should implement it
- Highest build generally perform better