MTP_BNN

May 15, 2022

packages

[ ]:
```
#empty
```

[ ]:
```
!pip install pyforest
# a package which automatically installs a package as an when it is used.
# may not work sometimes in notebook.
```

```
Collecting pyforest
  Downloading pyforest-1.1.0.tar.gz (15 kB)
Building wheels for collected packages: pyforest
  Building wheel for pyforest (setup.py) … done
  Created wheel for pyforest: filename=pyforest-1.1.0-py2.py3-none-any.whl
size=14607
sha256=df6567374f1522dce81901a780b0086c688d472482d4400bd4a7247df5f522d8
  Stored in directory: /root/.cache/pip/wheels/61/1c/da/48e6c884142d485475d852d6
9d20a096aba5beceb338822893
Successfully built pyforest
Installing collected packages: pyforest
Successfully installed pyforest-1.1.0
```

[ ]:
```
#automatic imports required packages as per usage in code
import pyforest
```

[ ]:
```
#packages
!pip install tensorflow-probability
!pip install nbconvert
```

```
Requirement already satisfied: tensorflow-probability in
/usr/local/lib/python3.7/dist-packages (0.16.0)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-packages
(from tensorflow-probability) (0.1.7)
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages
(from tensorflow-probability) (1.0.0)
Requirement already satisfied: gast>=0.3.2 in /usr/local/lib/python3.7/dist-
packages (from tensorflow-probability) (0.5.3)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from tensorflow-probability) (4.4.2)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-
```

```
packages (from tensorflow-probability) (1.15.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-
packages (from tensorflow-probability) (1.21.6)
Requirement already satisfied: cloudpickle>=1.3 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-probability) (1.3.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-
packages (5.6.1)
Requirement already satisfied: nbformat>=4.4 in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (5.3.0)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.7/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (2.6.1)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (0.6.0)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.7/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages
(from nbconvert) (5.0.0)
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (2.11.3)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (5.1.1)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (4.10.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.7/dist-packages (from nbconvert) (1.5.0)
Requirement already satisfied: MarkupSafe>=0.23 in
/usr/local/lib/python3.7/dist-packages (from jinja2>=2.4->nbconvert) (2.0.1)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.7/dist-
packages (from nbformat>=4.4->nbconvert) (2.15.3)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-
packages (from nbformat>=4.4->nbconvert) (4.3.3)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-
packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert) (21.4.0)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from
jsonschema>=2.6->nbformat>=4.4->nbconvert) (4.11.3)
Requirement already satisfied: importlib-resources>=1.4.0 in
/usr/local/lib/python3.7/dist-packages (from
jsonschema>=2.6->nbformat>=4.4->nbconvert) (5.7.1)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/usr/local/lib/python3.7/dist-packages (from
jsonschema>=2.6->nbformat>=4.4->nbconvert) (0.18.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from
```

```
jsonschema>=2.6->nbformat>=4.4->nbconvert) (4.2.0)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-
packages (from importlib-
resources>=1.4.0->jsonschema>=2.6->nbformat>=4.4->nbconvert) (3.8.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert) (1.15.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert) (0.5.1)
```

[ ]:
```
import pandas as pd
import numpy as np
```

0.0.1 DATA

import data

[ ]:
```
#using official url to load data
# this is the dataset which is used throughout the project. taken from uci␣
 ↪repository.
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00601/ai4i2020.
 ↪csv'
#loading the dataset into data variable
data = pd.read_csv(url)

data.head()
```

[ ]:
```
   UDI Product ID Type  Air temperature [K]  Process temperature [K]  \
0    1     M14860    M                298.1                    308.6
1    2     L47181    L                298.2                    308.7
2    3     L47182    L                298.1                    308.5
3    4     L47183    L                298.2                    308.6
4    5     L47184    L                298.2                    308.7

   Rotational speed [rpm]  Torque [Nm]  Tool wear [min]  Machine failure  TWF  \
0                    1551         42.8                0                0    0
1                    1408         46.3                3                0    0
2                    1498         49.4                5                0    0
3                    1433         39.5                7                0    0
4                    1408         40.0                9                0    0

   HDF  PWF  OSF  RNF
0    0    0    0    0
1    0    0    0    0
2    0    0    0    0
3    0    0    0    0
4    0    0    0    0
```

data description taken from UCI:

Abstract: The AI4I 2020 Predictive Maintenance Dataset is a synthetic dataset that reflects real predictive maintenance data encountered in industry.

Variable	Value
Data Set Characteristics:	Multivariate, Time-Series
Number of Instances:	10000
Area:	Computer
Attribute Characteristics:	Real
Number of Attributes:	14
Date Donated:	2020-08-30
Associated Tasks:	Classification, Regression, Causal-Discovery
Missing Values?	N/A
Number of Web Hits:	33135

** Data Set Information: **

Since real predictive maintenance datasets are generally difficult to obtain and in particular difficult to publish, we present and provide a synthetic dataset that reflects real predictive maintenance encountered in industry to the best of our knowledge.

Attribute Information:

The dataset consists of 10 000 data points stored as rows with 14 features in columns UID: unique identifier ranging from 1 to 10000 product ID: consisting of a letter L, M, or H for low (50% of all products), medium (30%) and high (20%) as product quality variants and a variant-specific serial number air temperature [K]: generated using a random walk process later normalized to a standard deviation of 2 K around 300 K process temperature [K]: generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K. rotational speed [rpm]: calculated from a power of 2860 W, overlaid with a normally distributed noise torque [Nm]: torque values are normally distributed around 40 Nm with a Ïf = 10 Nm and no negative values. tool wear [min]: The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process. and a 'machine failure' label that indicates, whether the machine has failed in this particular datapoint for any of the following failure modes are true.

The machine failure consists of five independent failure modes tool wear failure (TWF): the tool will be replaced of fail at a randomly selected tool wear time between 200 â€“ 240 mins (120 times in our dataset). At this point in time, the tool is replaced 69 times, and fails 51 times (randomly assigned). heat dissipation failure (HDF): heat dissipation causes a process failure, if the difference between air- and process temperature is below 8.6 K and the toolâ€™s rotational speed is below 1380 rpm. This is the case for 115 data points. power failure (PWF): the product of torque and rotational speed (in rad/s) equals the power required for the process. If this power is below 3500 W or above 9000 W, the process fails, which is the case 95 times in our dataset. overstrain failure (OSF): if the product of tool wear and torque exceeds 11,000 minNm for the L product variant (12,000 M, 13,000 H), the process fails due to overstrain. This is true for 98 datapoints. random failures (RNF): each process has a chance of 0,1 % to fail regardless of its process parameters. This is the case for only 5 datapoints, less than could be expected for 10,000 datapoints in our dataset.

If at least one of the above failure modes is true, the process fails and the 'machine failure' label is set to 1. It is therefore not transparent to the machine learning method, which of the failure modes has caused the process to fail

Relevant Papers:

Stephan Matzka, 'Explainable Artificial Intelligence for Predictive Maintenance Applications', Third International Conference on Artificial Intelligence for Industries (AI4I 2020), 2020 (in press)

[ ]:
```
data.describe()
```

[ ]:
```
                UDI  Air temperature [K]  Process temperature [K]  \
count  10000.00000         10000.000000             10000.000000
mean    5000.50000           300.004930               310.005560
std     2886.89568             2.000259                 1.483734
min        1.00000           295.300000               305.700000
25%     2500.75000           298.300000               308.800000
50%     5000.50000           300.100000               310.100000
75%     7500.25000           301.500000               311.100000
max    10000.00000           304.500000               313.800000

       Rotational speed [rpm]   Torque [Nm]  Tool wear [min]  Machine failure  \
count            10000.000000  10000.000000     10000.000000     10000.000000
mean              1538.776100     39.986910       107.951000         0.033900
std                179.284096      9.968934        63.654147         0.180981
min               1168.000000      3.800000         0.000000         0.000000
25%               1423.000000     33.200000        53.000000         0.000000
50%               1503.000000     40.100000       108.000000         0.000000
75%               1612.000000     46.800000       162.000000         0.000000
max               2886.000000     76.600000       253.000000         1.000000

                TWF           HDF           PWF           OSF          RNF
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.00000
mean       0.004600      0.011500      0.009500      0.009800      0.00190
std        0.067671      0.106625      0.097009      0.098514      0.04355
min        0.000000      0.000000      0.000000      0.000000      0.00000
25%        0.000000      0.000000      0.000000      0.000000      0.00000
50%        0.000000      0.000000      0.000000      0.000000      0.00000
75%        0.000000      0.000000      0.000000      0.000000      0.00000
max        1.000000      1.000000      1.000000      1.000000      1.00000
```

[ ]:
```
#for i in data:
    #print(data[i].unique())
```

[ ]:
```
# checking about different attributes in the data
data.nunique()
```

[ ]:
```
UDI                        10000
Product ID                 10000
Type                           3
Air temperature [K]           93
Process temperature [K]       82
Rotational speed [rpm]       941
Torque [Nm]                  577
Tool wear [min]              246
Machine failure                2
TWF                            2
HDF                            2
PWF                            2
OSF                            2
RNF                            2
dtype: int64
```

[ ]:
```
#basic info about dataset
df = data
df.shape
df.index
df.columns
df.info()
df.count()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   UDI                      10000 non-null  int64
 1   Product ID               10000 non-null  object
 2   Type                     10000 non-null  object
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64
 8   Machine failure          10000 non-null  int64
 9   TWF                      10000 non-null  int64
 10  HDF                      10000 non-null  int64
 11  PWF                      10000 non-null  int64
 12  OSF                      10000 non-null  int64
 13  RNF                      10000 non-null  int64
dtypes: float64(3), int64(9), object(2)
memory usage: 1.1+ MB
```

```
[ ]: UDI                        10000
     Product ID                 10000
     Type                       10000
     Air temperature [K]        10000
     Process temperature [K]    10000
     Rotational speed [rpm]     10000
     Torque [Nm]                10000
     Tool wear [min]            10000
     Machine failure            10000
     TWF                        10000
     HDF                        10000
     PWF                        10000
     OSF                        10000
     RNF                        10000
     dtype: int64
```

```
[ ]: # not used all these but just to check the data.
     df.sum()
     df.cumsum()
     df.min()
     df.max()
     df.describe()
     df.mean()
     df.median()
```

```
/usr/local/lib/python3.7/dist-packages/pyforest/__init__.py:7: FutureWarning:
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None')
is deprecated; in a future version this will raise TypeError.  Select only valid
columns before calling the reduction.
  install_labextension,
/usr/local/lib/python3.7/dist-packages/pyforest/__init__.py:8: FutureWarning:
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None')
is deprecated; in a future version this will raise TypeError.  Select only valid
columns before calling the reduction.
  )
```

```
[ ]: UDI                        5000.5
     Air temperature [K]         300.1
     Process temperature [K]     310.1
     Rotational speed [rpm]     1503.0
     Torque [Nm]                  40.1
     Tool wear [min]             108.0
     Machine failure               0.0
     TWF                           0.0
     HDF                           0.0
     PWF                           0.0
     OSF                           0.0
```

```
RNF                             0.0
dtype: float64
```

preprocessing data

[ ]:
```
#define X and y from df
# product id is unique for each data row and its not important
# but we have product type of 3 categories
# L, M, H are three types representing for low (50% of all products),
# medium (30%) and high (20%) as product quality variants respectively
df['Type'].unique()
```

[ ]:
```
array(['M', 'L', 'H'], dtype=object)
```

[ ]:
```
# converting this categorical data to numerical with class 0, 1, 2 for L,M,H␣
↪respectively
# using OrdinalEncoder from sklearn for ordinal data of product quality variant
# indicating l for low quality, m for medium quality, h for high quality
# one-hot encoding is not suitable for ordinal data
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
df['Type'] = ordinal_encoder.fit_transform(df[['Type']])
df['Type'].unique()
# this gives categories converted into integers
```

[ ]:
```
array([2., 1., 0.])
```

[ ]:
```
# these are original categories in data
ordinal_encoder.categories_
```

[ ]:
```
[array(['H', 'L', 'M'], dtype=object)]
```

[ ]:
```
# this sorts all the categories present and assigns values to them in␣
↪alphabetical order
# 0 for H
# 1 for L
# 2 for M
print(ordinal_encoder.inverse_transform([[0]]))
print(ordinal_encoder.inverse_transform([[1]]))
print(ordinal_encoder.inverse_transform([[2]]))
```

```
[['H']]
[['L']]
[['M']]
```

[ ]:
```
df.describe()
```

[ ]:
```
                UDI         Type  Air temperature [K]  Process temperature [K]  \
count  10000.00000  10000.00000         10000.000000             10000.000000
mean    5000.50000      1.19940           300.004930               310.005560
std     2886.89568      0.60023             2.000259                 1.483734
min        1.00000      0.00000           295.300000               305.700000
25%     2500.75000      1.00000           298.300000               308.800000
50%     5000.50000      1.00000           300.100000               310.100000
75%     7500.25000      2.00000           301.500000               311.100000
max    10000.00000      2.00000           304.500000               313.800000

       Rotational speed [rpm]   Torque [Nm]  Tool wear [min]  Machine failure  \
count            10000.000000  10000.000000     10000.000000     10000.000000
mean              1538.776100     39.986910       107.951000         0.033900
std                179.284096      9.968934        63.654147         0.180981
min               1168.000000      3.800000         0.000000         0.000000
25%               1423.000000     33.200000        53.000000         0.000000
50%               1503.000000     40.100000       108.000000         0.000000
75%               1612.000000     46.800000       162.000000         0.000000
max               2886.000000     76.600000       253.000000         1.000000

                TWF           HDF           PWF           OSF          RNF
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.00000
mean       0.004600      0.011500      0.009500      0.009800      0.00190
std        0.067671      0.106625      0.097009      0.098514      0.04355
min        0.000000      0.000000      0.000000      0.000000      0.00000
25%        0.000000      0.000000      0.000000      0.000000      0.00000
50%        0.000000      0.000000      0.000000      0.000000      0.00000
75%        0.000000      0.000000      0.000000      0.000000      0.00000
max        1.000000      1.000000      1.000000      1.000000      1.00000
```

[ ]:
```
df.nunique()
```

[ ]:
```
UDI                        10000
Product ID                 10000
Type                           3
Air temperature [K]           93
Process temperature [K]       82
Rotational speed [rpm]       941
Torque [Nm]                  577
Tool wear [min]              246
Machine failure                2
TWF                            2
HDF                            2
PWF                            2
OSF                            2
RNF                            2
dtype: int64
```

[ ]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   UDI                      10000 non-null  int64
 1   Product ID               10000 non-null  object
 2   Type                     10000 non-null  float64
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64
 8   Machine failure          10000 non-null  int64
 9   TWF                      10000 non-null  int64
 10  HDF                      10000 non-null  int64
 11  PWF                      10000 non-null  int64
 12  OSF                      10000 non-null  int64
 13  RNF                      10000 non-null  int64
dtypes: float64(4), int64(9), object(1)
memory usage: 1.1+ MB
```

[ ]:
```python
# now make the final dataset to be used in NN
# remove the product id variable
# remaining attributes are of types either int64 or float64
df.drop('Product ID', axis=1, inplace=True)
df.drop('UDI', axis=1, inplace=True)
```

[ ]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Type                     10000 non-null  float64
 1   Air temperature [K]      10000 non-null  float64
 2   Process temperature [K]  10000 non-null  float64
 3   Rotational speed [rpm]   10000 non-null  int64
 4   Torque [Nm]              10000 non-null  float64
 5   Tool wear [min]          10000 non-null  int64
 6   Machine failure          10000 non-null  int64
 7   TWF                      10000 non-null  int64
 8   HDF                      10000 non-null  int64
 9   PWF                      10000 non-null  int64
 10  OSF                      10000 non-null  int64
```

```
 11  RNF                        10000 non-null  int64
dtypes: float64(4), int64(8)
memory usage: 937.6 KB
```

[ ]:
```python
## add mitosheet data visualization
```

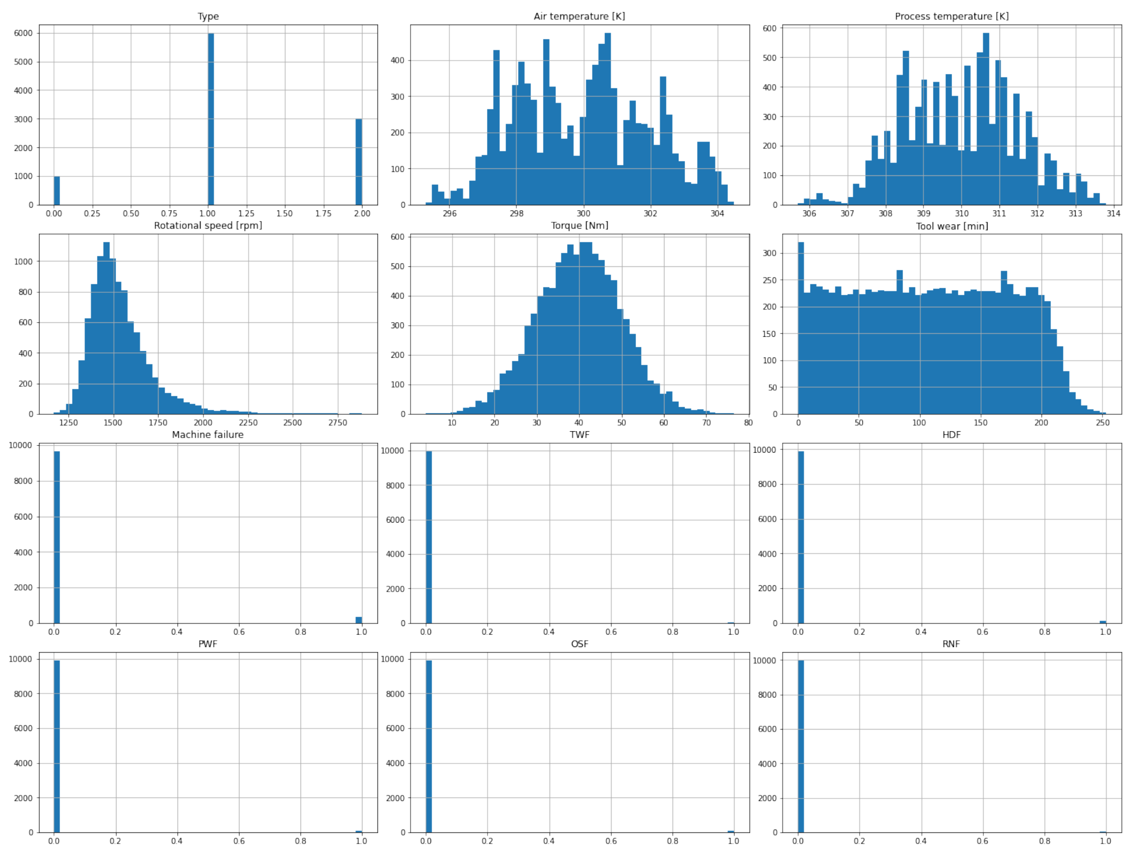
mitosheet visualization code

[ ]:
```python
# exploring data with various plots to know more about it
```

[ ]:
```python
df.hist(bins=50, figsize=(20,15))
plt.tight_layout(pad=0.4)
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

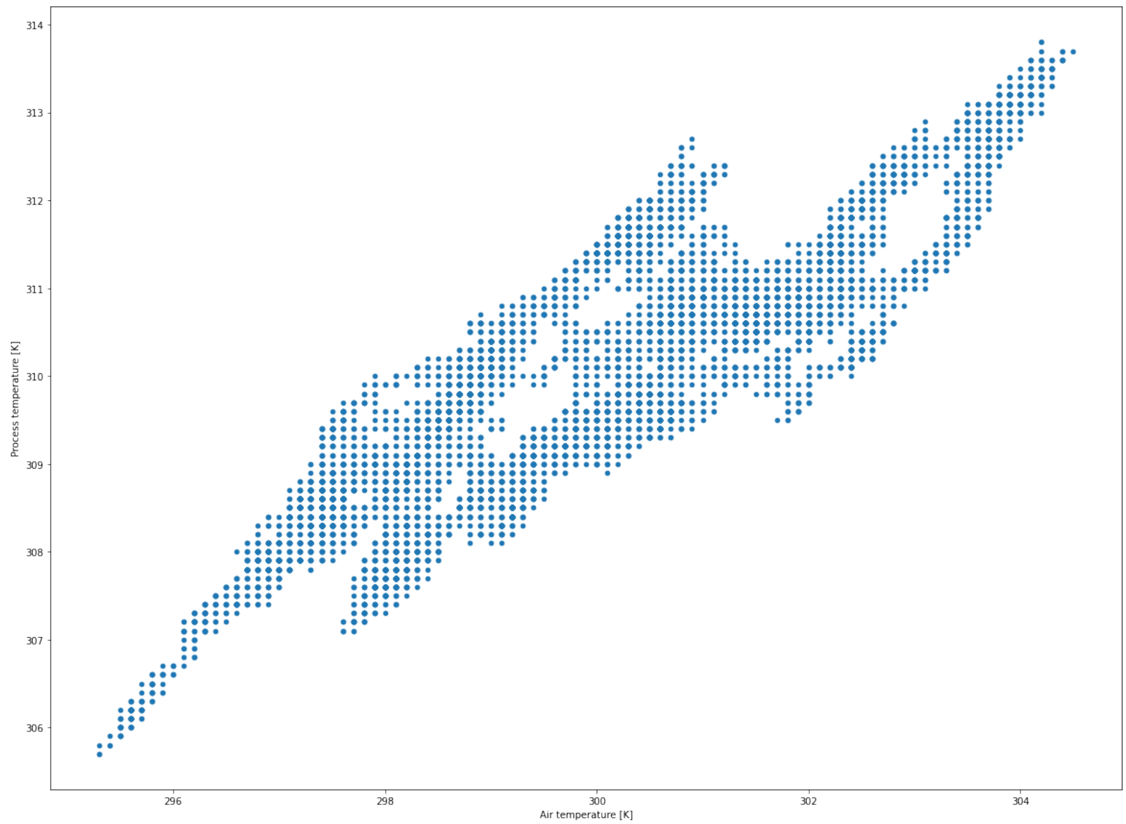


[ ]:
```python
df.plot.scatter(y = 'Type',x='Air temperature [K]', figsize=(20,15))
plt.show()
```

<IPython.core.display.Javascript object>

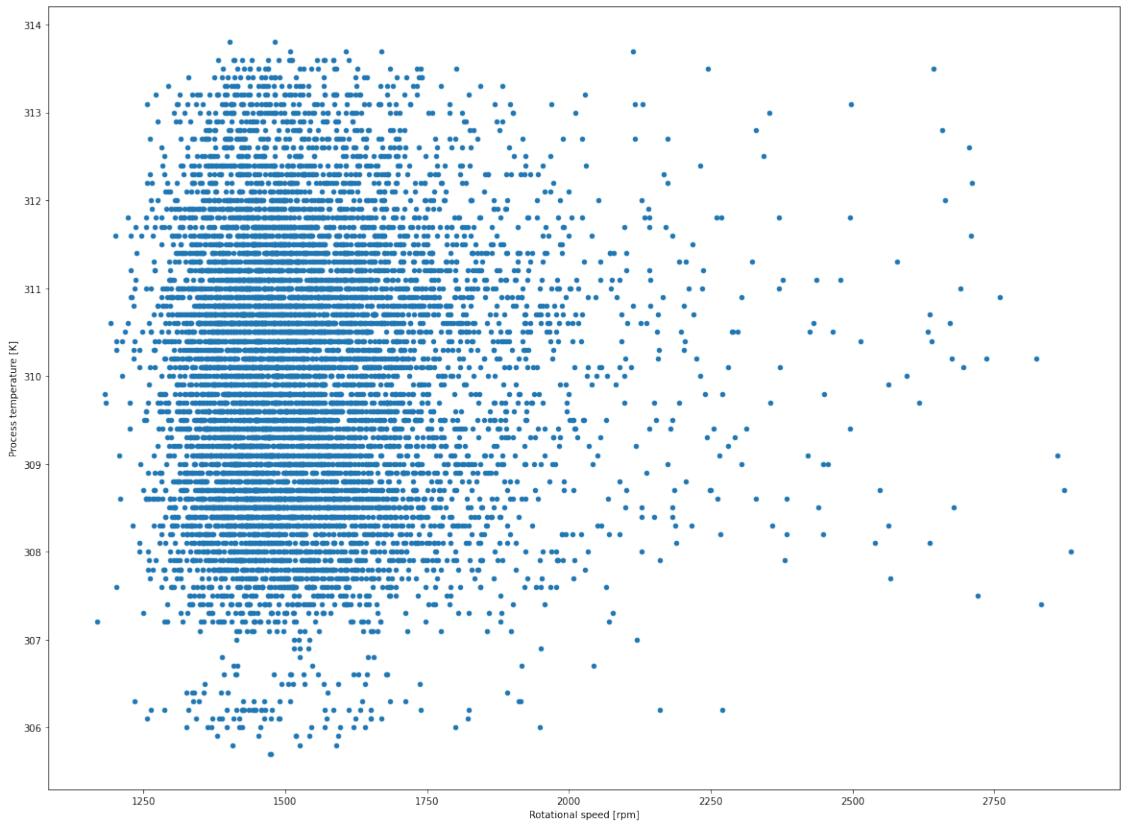
[ ]:
```
df.plot.scatter(y = 'Process temperature [K]',x='Air temperature [K]',␣
↪figsize=(20,15))
plt.show()
```

<IPython.core.display.Javascript object>



[ ]:
```
df.plot.scatter(y = 'Process temperature [K]',x='Rotational speed [rpm]',␣
↪figsize=(20,15))
plt.show()
```

<IPython.core.display.Javascript object>



[ ]:
```
#confusion matrix
```

[ ]:
```
import seaborn as sns

#correlation matrix
numeric_col = ['Type', 'Air temperature [K]', 'Process temperature [K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]',
        'TWF', 'HDF', 'PWF', 'OSF', 'RNF','Machine failure']
# Correlation Matrix formation
corr_matrix = df.loc[:,numeric_col].corr()
print(corr_matrix)
#Using heatmap to visualize the correlation matrix
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr_matrix, annot=True,ax=ax)
```

```
                             Type  Air temperature [K]  \
Type                     1.000000             0.017599
Air temperature [K]      0.017599             1.000000
Process temperature [K]  0.013444             0.876107
Rotational speed [rpm]  -0.002693             0.022670
Torque [Nm]              0.004011            -0.013778
```

Tool wear [min]	-0.003930	0.013853
TWF	-0.005349	0.009955
HDF	0.000108	0.137831
PWF	0.012121	0.003470
OSF	-0.021211	0.001988
RNF	-0.022147	0.017688
Machine failure	-0.005152	0.082556

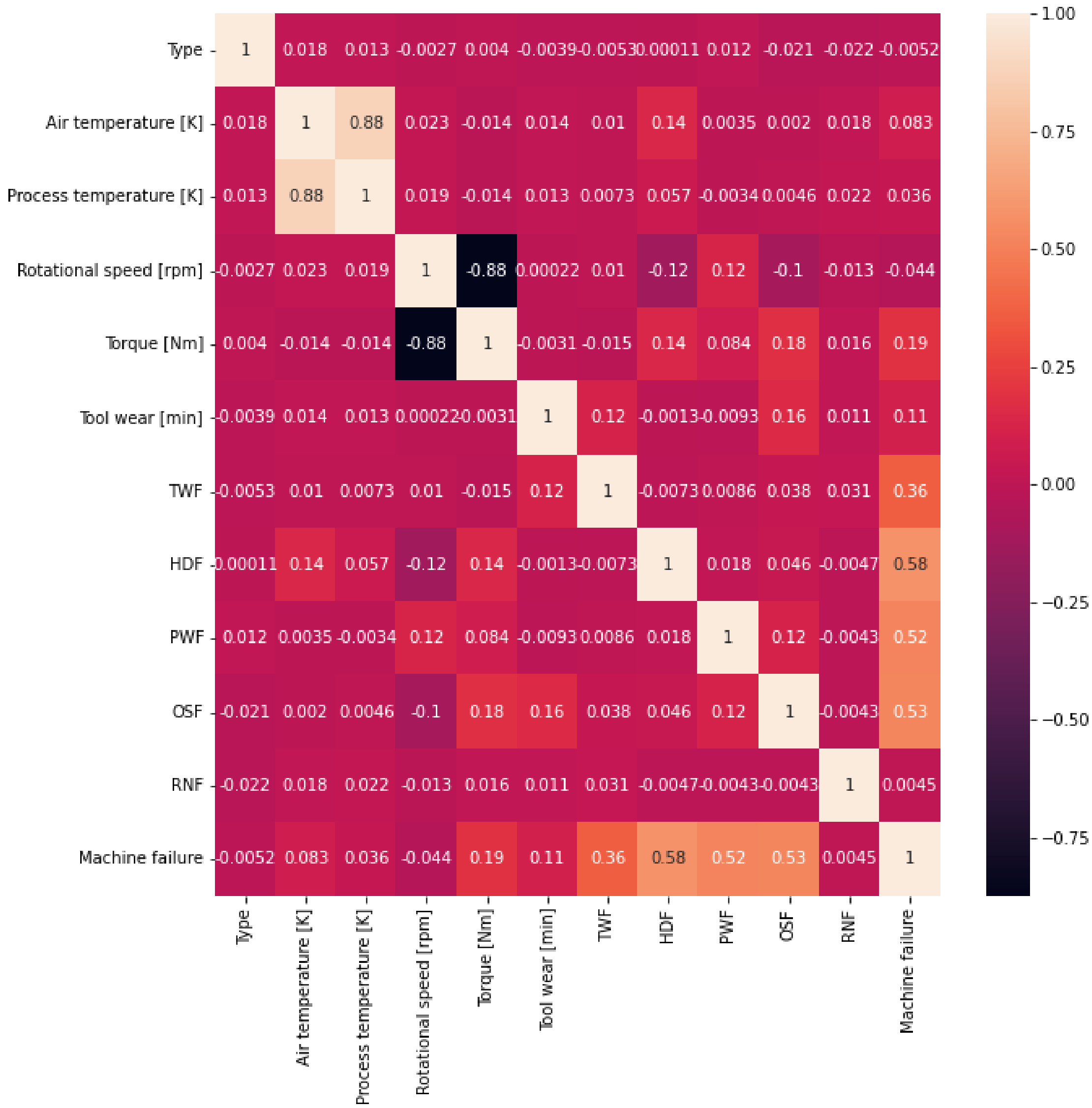
	Process temperature [K]	Rotational speed [rpm]
Type	0.013444	-0.002693
Air temperature [K]	0.876107	0.022670
Process temperature [K]	1.000000	0.019277
Rotational speed [rpm]	0.019277	1.000000
Torque [Nm]	-0.014061	-0.875027
Tool wear [min]	0.013488	0.000223
TWF	0.007315	0.010389
HDF	0.056933	-0.121241
PWF	-0.003355	0.123018
OSF	0.004554	-0.104575
RNF	0.022279	-0.013088
Machine failure	0.035946	-0.044188

	Torque [Nm]	Tool wear [min]	TWF	HDF
Type	0.004011	-0.003930	-0.005349	0.000108
Air temperature [K]	-0.013778	0.013853	0.009955	0.137831
Process temperature [K]	-0.014061	0.013488	0.007315	0.056933
Rotational speed [rpm]	-0.875027	0.000223	0.010389	-0.121241
Torque [Nm]	1.000000	-0.003093	-0.014662	0.142610
Tool wear [min]	-0.003093	1.000000	0.115792	-0.001287
TWF	-0.014662	0.115792	1.000000	-0.007332
HDF	0.142610	-0.001287	-0.007332	1.000000
PWF	0.083781	-0.009334	0.008577	0.018443
OSF	0.183465	0.155894	0.038243	0.046396
RNF	0.016136	0.011326	0.030970	-0.004706
Machine failure	0.191321	0.105448	0.362904	0.575800

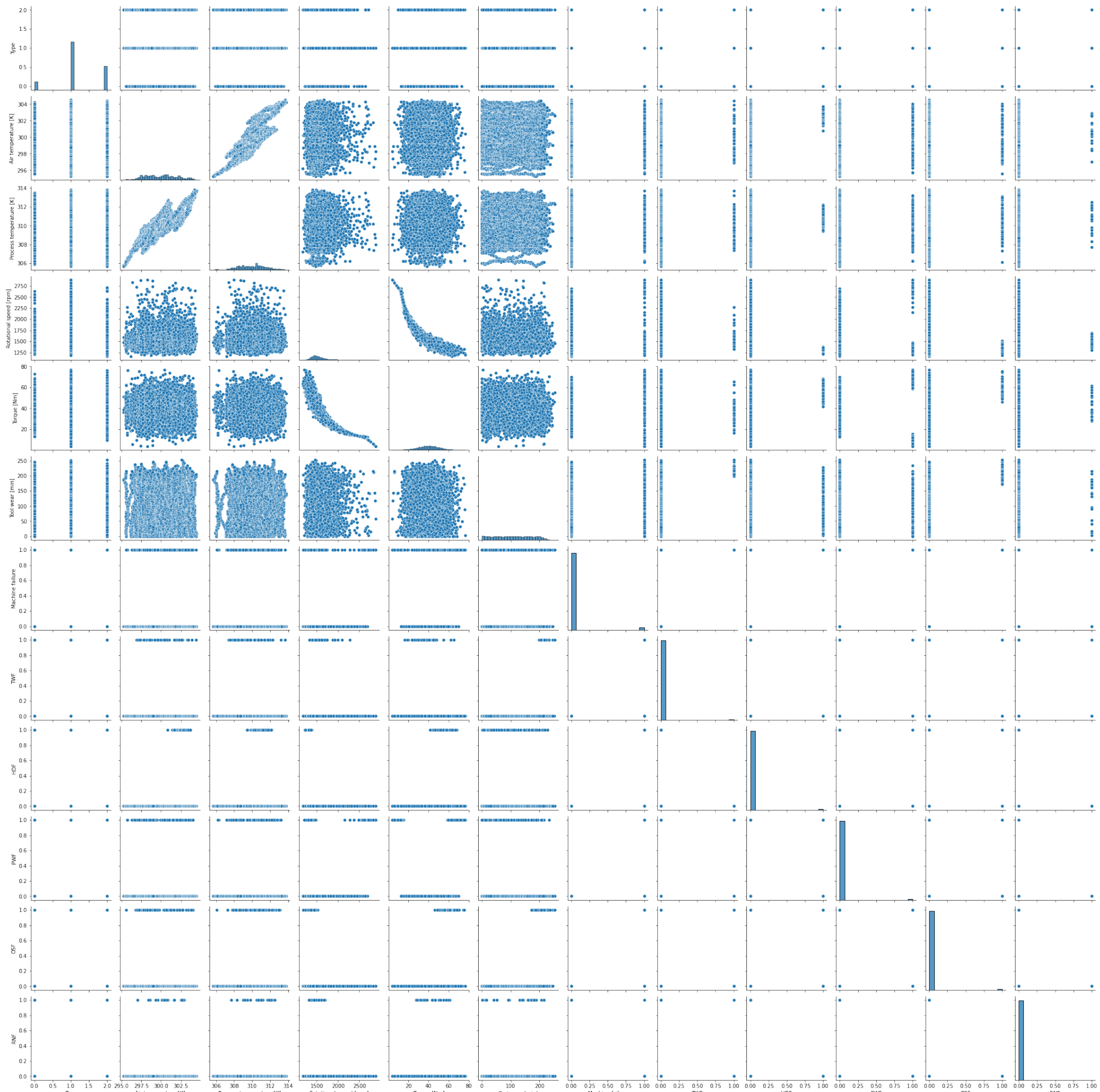
	PWF	OSF	RNF	Machine failure
Type	0.012121	-0.021211	-0.022147	-0.005152
Air temperature [K]	0.003470	0.001988	0.017688	0.082556
Process temperature [K]	-0.003355	0.004554	0.022279	0.035946
Rotational speed [rpm]	0.123018	-0.104575	-0.013088	-0.044188
Torque [Nm]	0.083781	0.183465	0.016136	0.191321
Tool wear [min]	-0.009334	0.155894	0.011326	0.105448
TWF	0.008577	0.038243	0.030970	0.362904
HDF	0.018443	0.046396	-0.004706	0.575800
PWF	1.000000	0.115836	-0.004273	0.522812
OSF	0.115836	1.000000	-0.004341	0.531083
RNF	-0.004273	-0.004341	1.000000	0.004516

```
Machine failure          0.522812  0.531083  0.004516          1.000000

<IPython.core.display.Javascript object>
```

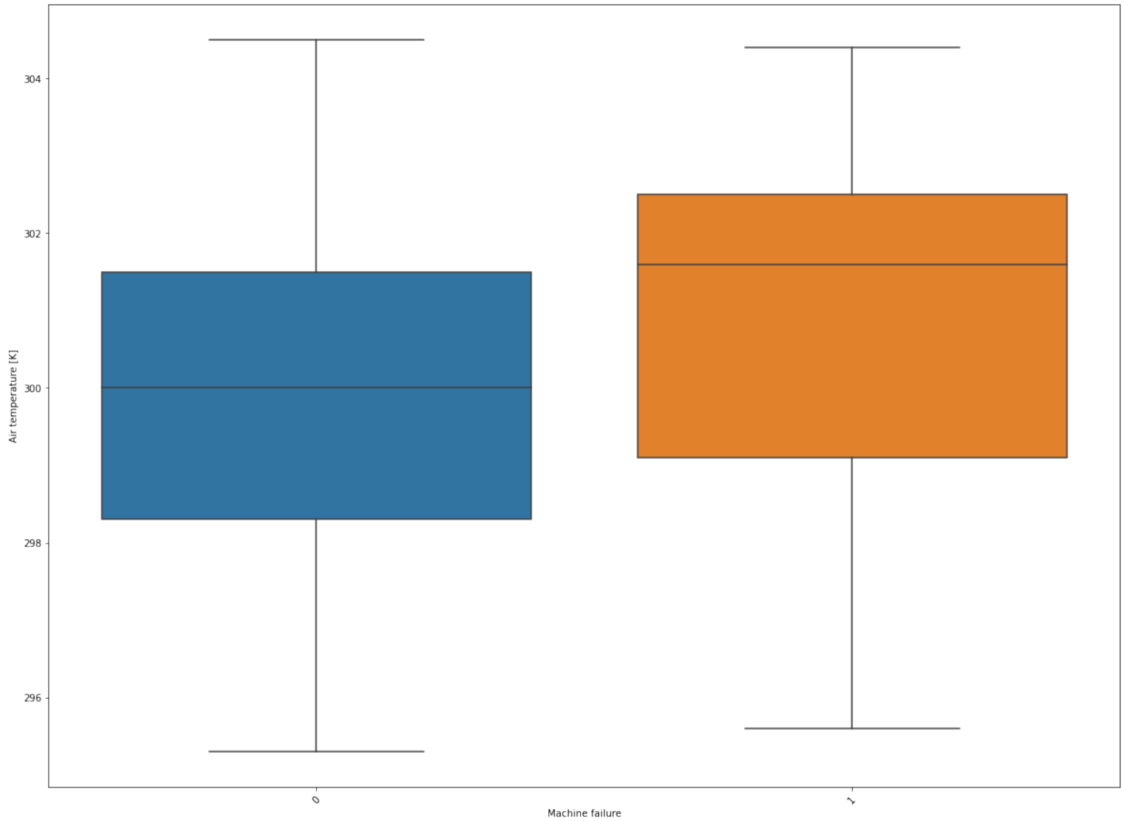
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f982679d450>



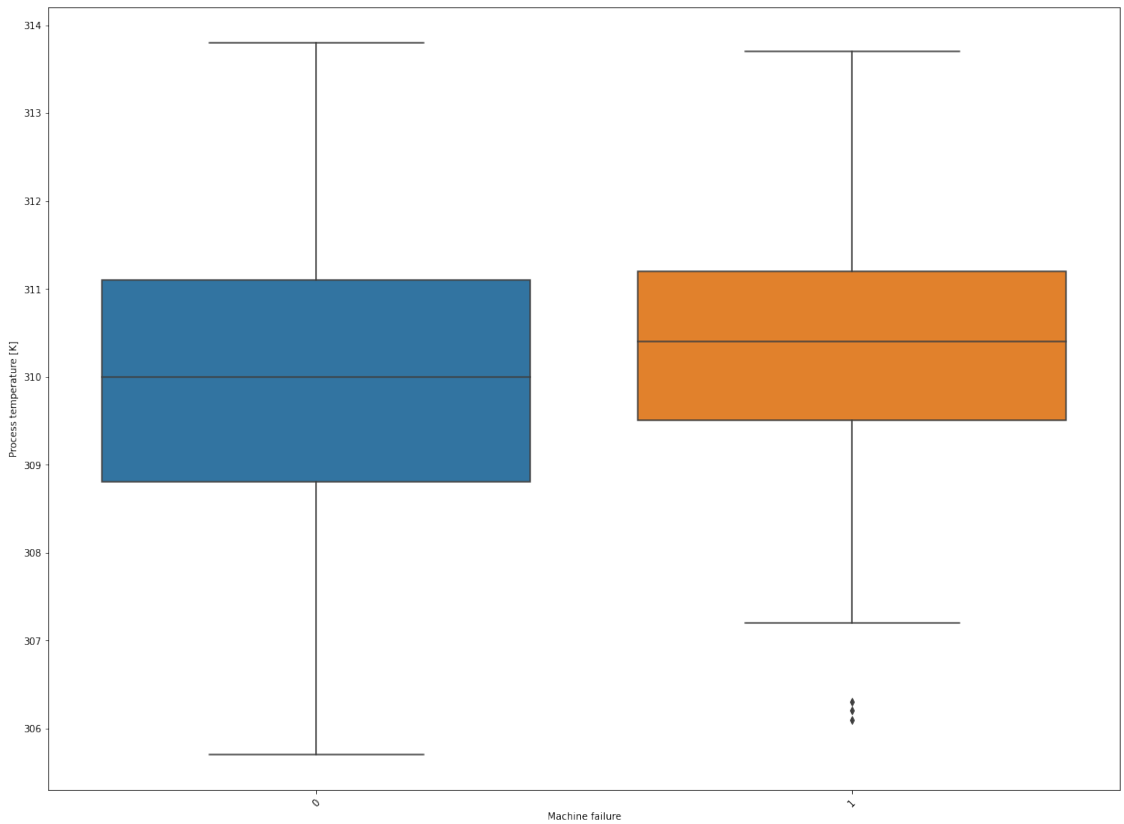
[ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
sns.pairplot(df, kind="scatter")
plt.show()
```



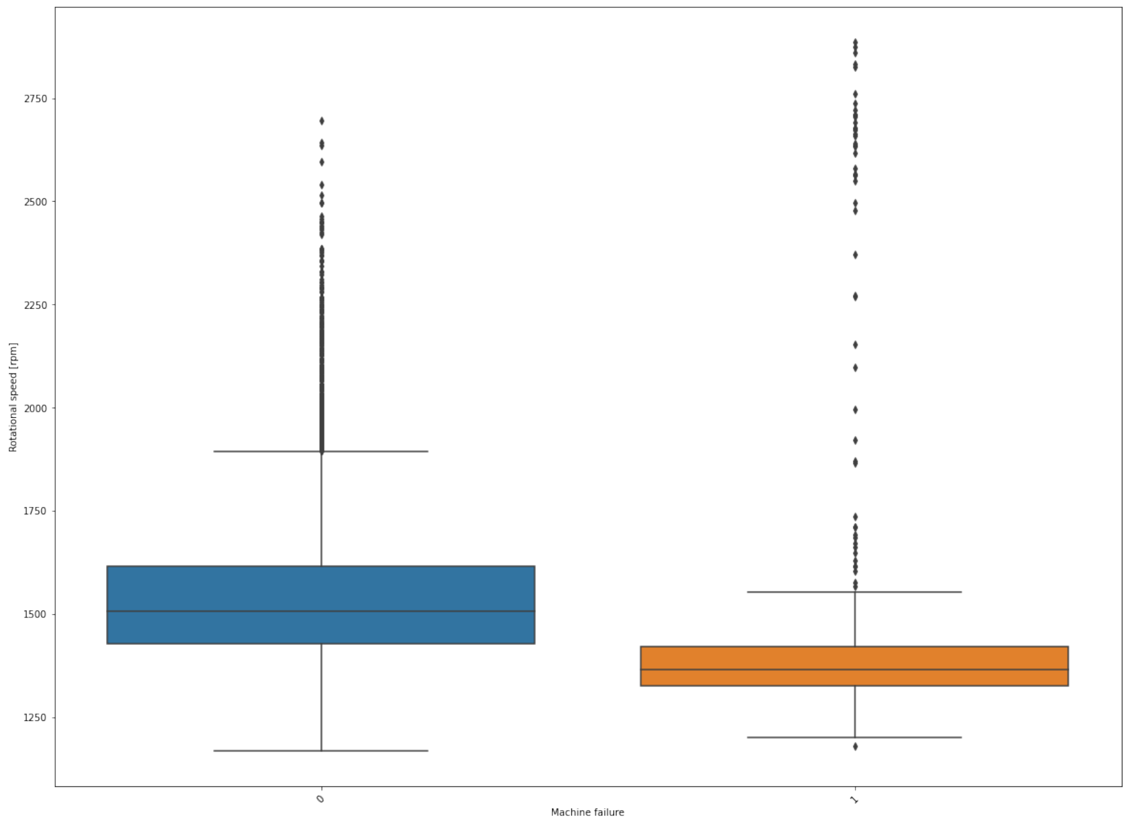
```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Air temperature [K]', x = 'Machine failure');
```



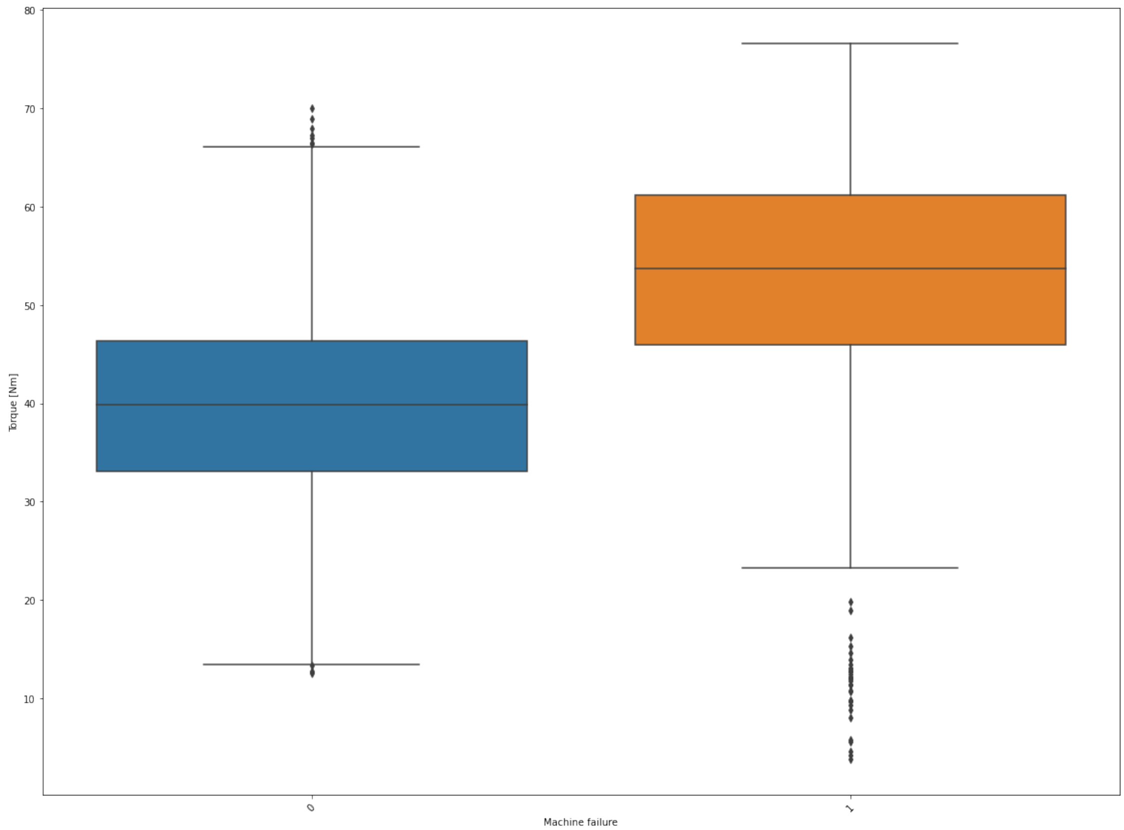

```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Process temperature [K]', x = 'Machine failure');
```



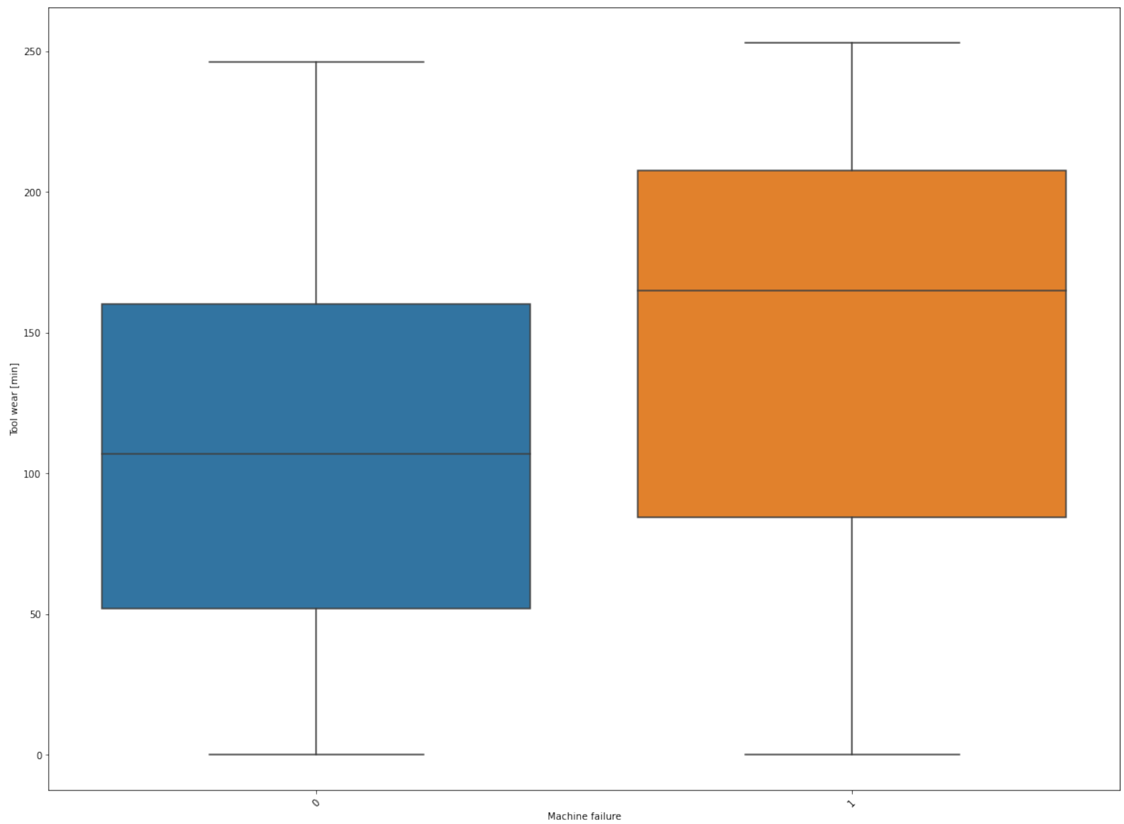
[ ]:
```
plt.figure(figsize=(20,15))
plt.xticks(rotation=45)
sns.boxplot(data = df, y = 'Rotational speed [rpm]', x = 'Machine failure');
```



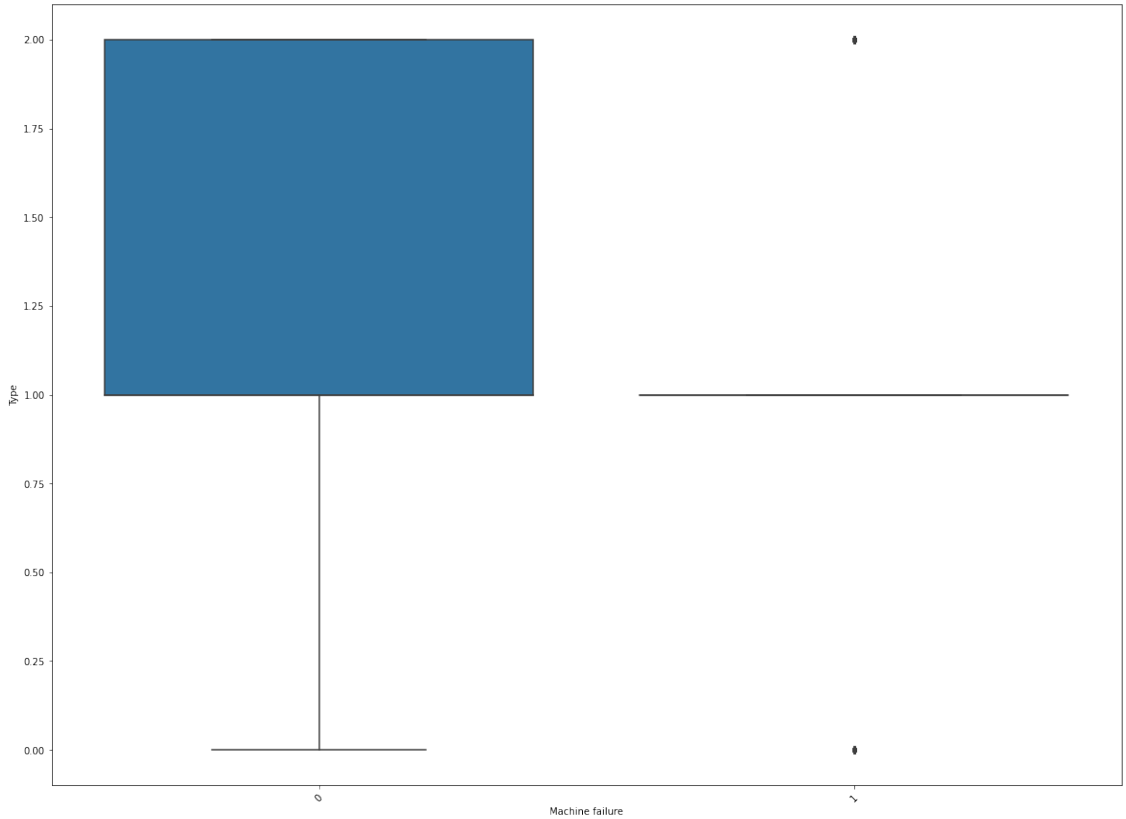

```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Torque [Nm]', x = 'Machine failure');
```



[ ]:
```
plt.figure(figsize=(20,15))
plt.xticks(rotation=45)
sns.boxplot(data = df, y = 'Tool wear [min]', x = 'Machine failure');
```




```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Type', x = 'Machine failure');
```



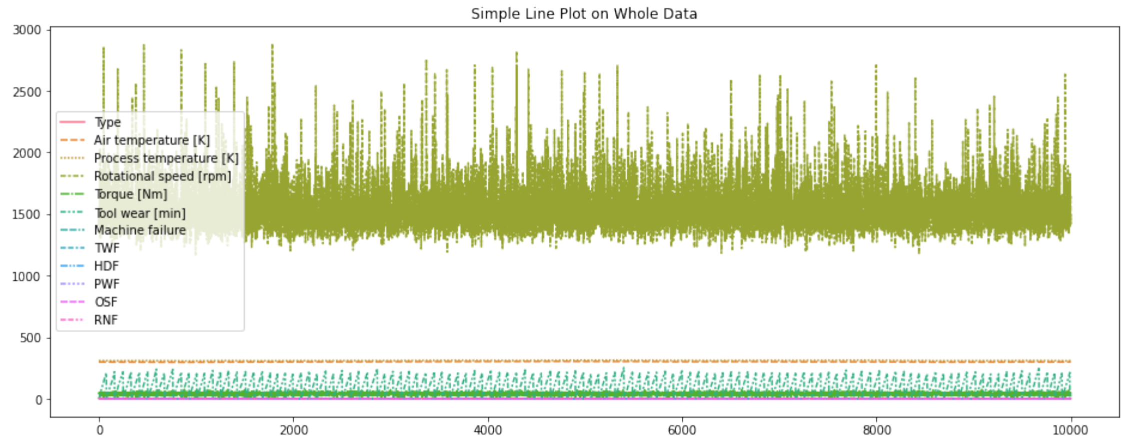
0.0.2 BNN

```
[ ]: #importing all the required packages for building a bnn
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_datasets as tfds
import tensorflow_probability as tfp
```

visualizing data

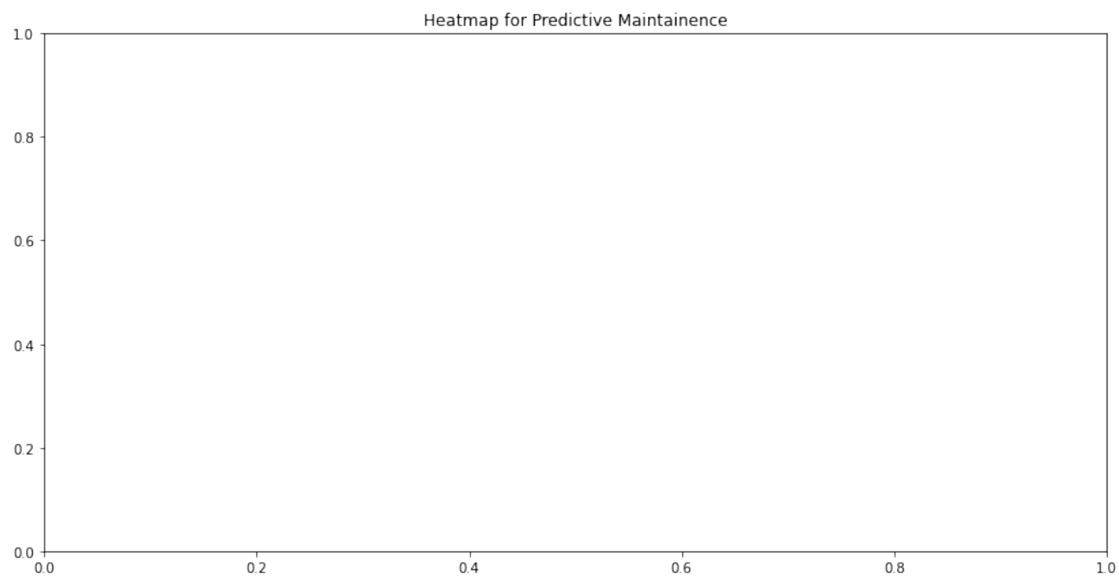
```
[ ]: import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plt.figure(figsize=(16,6))
plt.title("Simple Line Plot on Whole Data")
sns.lineplot(data=df)
```

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9820e31d10>




```
[ ]: #heatmaps on whole data
plt.figure(figsize=(14,7))
# Add title
plt.title("Heatmap for Predictive Maintainence")
# Heatmap
#sns.heatmap(data=df['Machine failure'], annot=True)
# Add label for horizontal axis
#plt.xlabel("Axis")
```

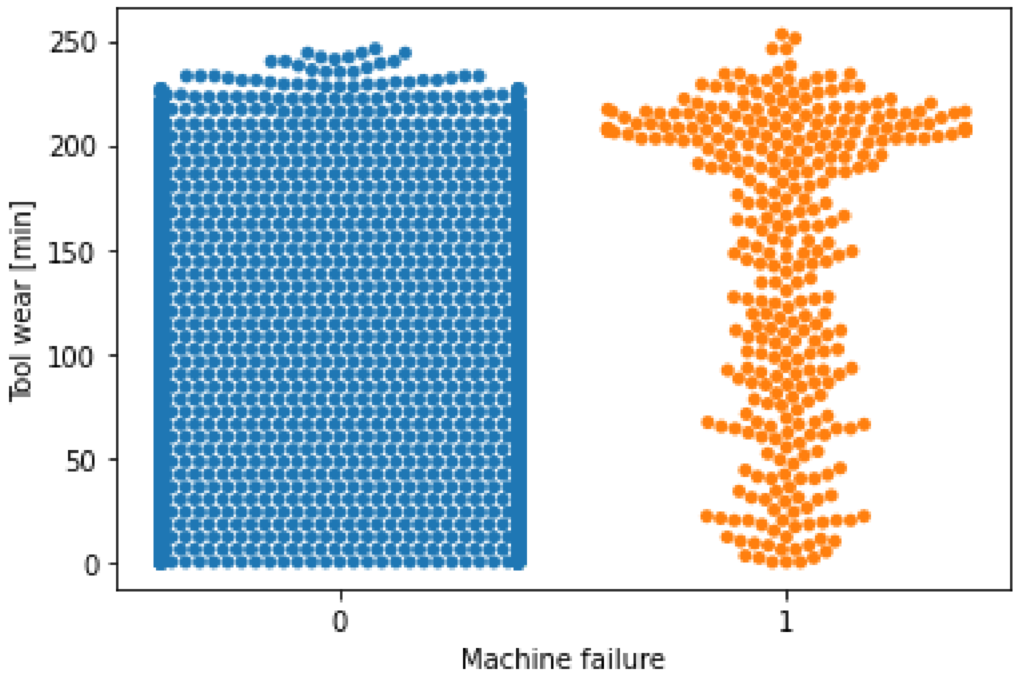
[ ]: Text(0.5, 1.0, 'Heatmap for Predictive Maintainence')



[ ]:
```
sns.swarmplot(x=df['Machine failure'],y=df['Tool wear [min]'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning:
89.6% of the points cannot be placed; you may want to decrease the size of the
markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

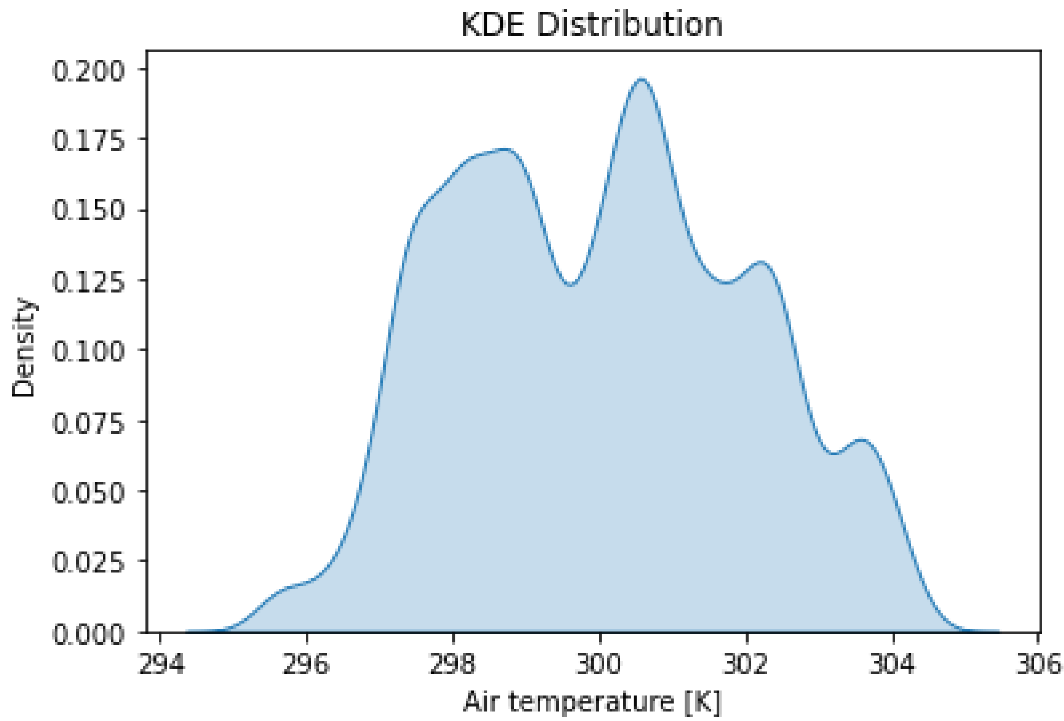
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f97ba4fb090>



[ ]:
```
#stripplot
```

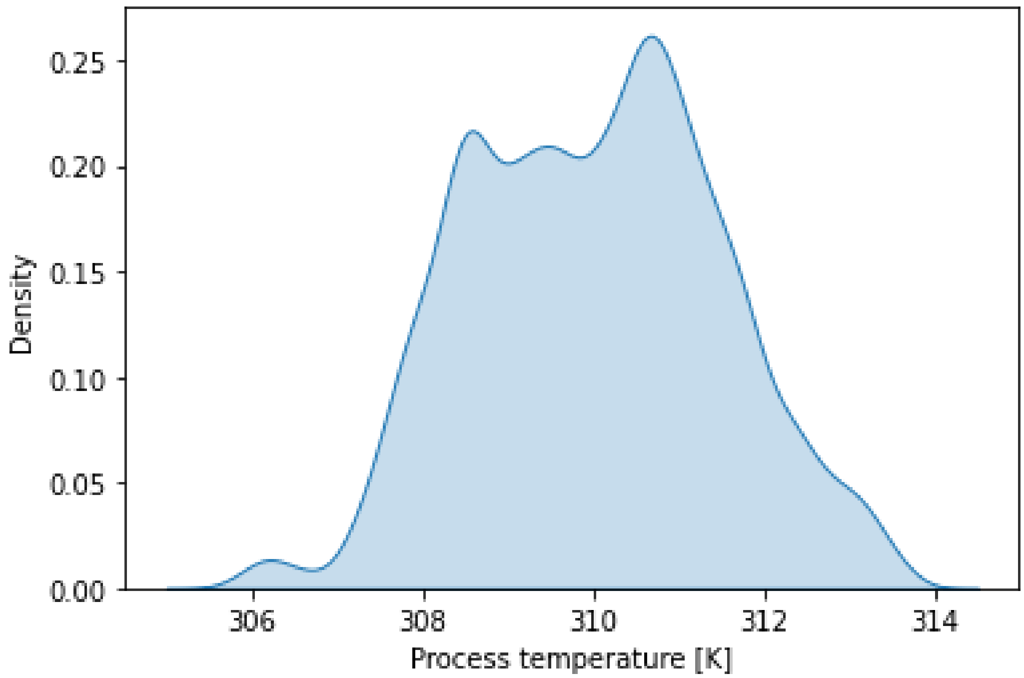
[ ]:
```
#distribution
#for i in df:
sns.kdeplot(data=df['Air temperature [K]'], label='Air temperature [K]',␣
 ↪shade=True)

plt.title('KDE Distribution')
```

[ ]: Text(0.5, 1.0, 'KDE Distribution')



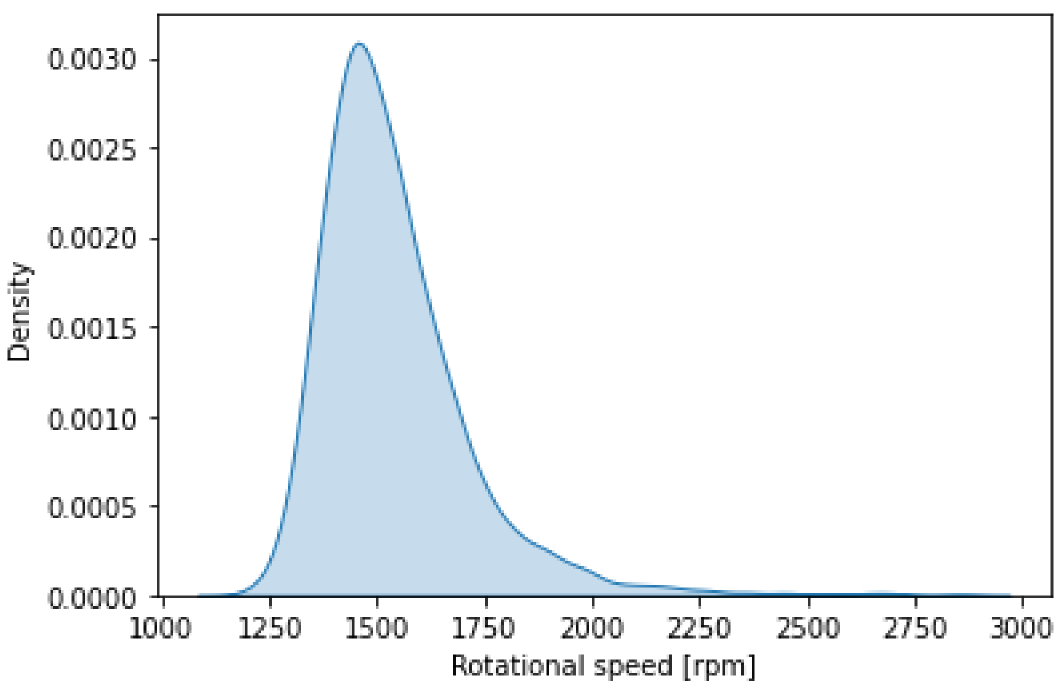
[ ]:
```
sns.kdeplot(data=df['Process temperature [K]'], label='Process temperature␣
↪[K]', shade=True)
```

[ ]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f97ba43e7d0>`



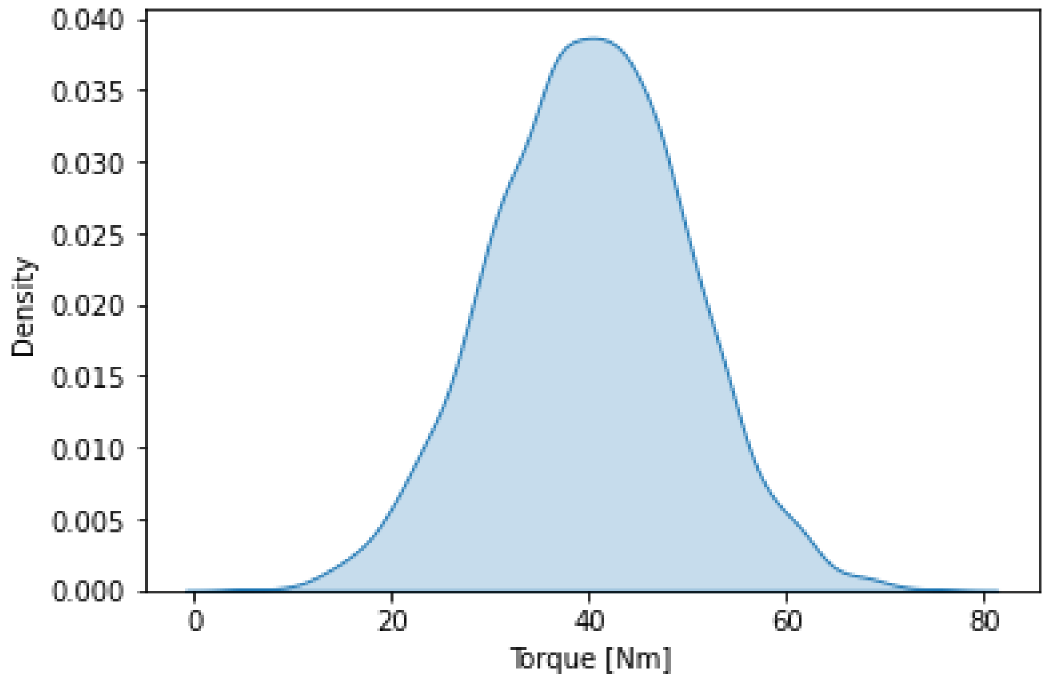

```
[ ]: sns.kdeplot(data=df['Rotational speed [rpm]'], label='Rotational speed [rpm]',
     ↪shade=True)
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f97ba346750>
```



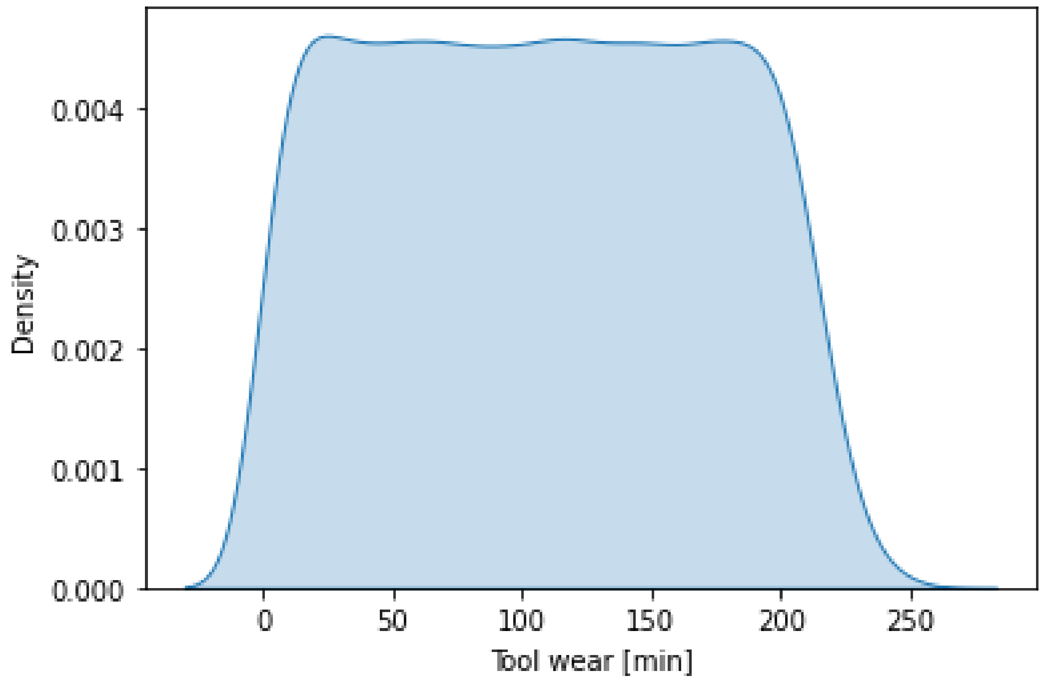
[ ]:
```python
sns.kdeplot(data=df['Torque [Nm]'], label='Torque [Nm]', shade=True)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f97ba3321d0>
```



[ ]:
```python
sns.kdeplot(data=df['Tool wear [min]'], label='Tool wear [min]', shade=True)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f97b818d250>
```



Create training and evaluation datasets

[ ]:
```
# listing all the columns in the dataset
df.columns
```

[ ]:
```
Index(['Type', 'Air temperature [K]', 'Process temperature [K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]',
       'Machine failure', 'TWF', 'HDF', 'PWF', 'OSF', 'RNF'],
      dtype='object')
```

[ ]:
```
from sklearn.model_selection import train_test_split

#first moving target variable "Machine Failure" to end and then defining X and y
df = df[['Type', 'Air temperature [K]', 'Process temperature [K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]',
        'TWF', 'HDF', 'PWF', 'OSF', 'RNF','Machine failure']]
print(df.shape)
# excluding last variable for target variable
X = df.iloc[:, :-1]
print(X.shape)
# making last variable as target variable
y = df.iloc[:, -1]
print(y.shape)
```

```
# using 70:30 split for making training and testing datasets and using random␣
↪state as 42 to repeat this random split.
X_train,X_test,y_train,y_test = train_test_split(X, y,test_size=0.
↪3,random_state=42)
```

```
(10000, 12)
(10000, 11)
(10000,)
```

[ ]:
```
# the shapes of X_train,X_test,y_train,y_test
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7000, 11)
(3000, 11)
(7000,)
(3000,)
```

[ ]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(7000, 11)
(7000,)
```

[ ]:
```
y_train.head()
```

[ ]:
```
9069    0
2603    0
7738    0
1579    0
5058    0
Name: Machine failure, dtype: int64
```

[ ]:
```
# correct
#done
#train dataset
train_d = pd.DataFrame(X_train)
train_d['y_train'] = y_train
print(train_d.shape)
print(train_d.shape)

#test dataset
test_d = pd.DataFrame(X_test)
test_d['y_test'] = y_test
print(test_d.shape)
```

```python
print(test_d.shape)
```

```
(7000, 12)
(7000, 12)
(3000, 12)
(3000, 12)
```

[ ]:
```python
train_d.head()
```

[ ]:
```
      Type  Air temperature [K]  Process temperature [K]  \
9069   2.0                297.2                    308.2
2603   2.0                299.3                    309.2
7738   2.0                300.5                    312.0
1579   1.0                298.3                    308.3
5058   1.0                303.9                    312.9

      Rotational speed [rpm]  Torque [Nm]  Tool wear [min]  TWF  HDF  PWF  \
9069                    1678         28.1              133    0    0    0
2603                    1334         46.3               31    0    0    0
7738                    1263         60.8              146    0    0    0
1579                    1444         43.8              176    0    0    0
5058                    1526         42.5              194    0    0    0

      OSF  RNF  y_train
9069    0    0        0
2603    0    0        0
7738    0    0        0
1579    0    0        0
5058    0    0        0
```

[ ]:
```python
test_d.head()
```

[ ]:
```
      Type  Air temperature [K]  Process temperature [K]  \
6252   1.0                300.8                    310.3
4684   2.0                303.6                    311.8
1731   2.0                298.3                    307.9
4742   1.0                303.3                    311.3
4521   1.0                302.4                    310.4

      Rotational speed [rpm]  Torque [Nm]  Tool wear [min]  TWF  HDF  PWF  \
6252                    1538         36.1              198    0    0    0
4684                    1421         44.8              101    0    0    0
1731                    1485         42.0              117    0    0    0
4742                    1592         33.7               14    0    0    0
4521                    1865         23.9              129    0    0    0

      OSF  RNF  y_test
```

6252	0	0	0
4684	0	0	1
1731	0	0	0
4742	0	0	0
4521	0	0	0

Compile, train, and evaluate the model

[ ]:
```
# from here will write in the form of functions
# but not used
```

Create model inputs

Experiment 1: standard neural network(Non-bayesian neural network)

[ ]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from keras.models import Sequential # to initialize NN
from keras.layers import Dense # to build layers
# building a standard neural network with 3 layers
classifier = Sequential()
classifier.add(Dense(units = 5, input_dim = X_train.shape[1])) # changed this
classifier.add(Dense(units = 3, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])
history = classifier.fit(X_train, y_train, epochs=50)
#validation_data = (np.asarray(X_test), np.asarray(y_test)),verbose=0
test_loss, test_acc = classifier.evaluate(X_test,  y_test, verbose=2)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
```

```
Epoch 1/50
219/219 [==============================] - 1s 2ms/step - loss: 0.6464 -
accuracy: 0.9643
Epoch 2/50
219/219 [==============================] - 0s 2ms/step - loss: 0.5604 -
accuracy: 0.9649
Epoch 3/50
219/219 [==============================] - 0s 2ms/step - loss: 0.4896 -
accuracy: 0.9649
Epoch 4/50
219/219 [==============================] - 0s 2ms/step - loss: 0.4311 -
accuracy: 0.9649
Epoch 5/50
219/219 [==============================] - 0s 2ms/step - loss: 0.3829 -
```

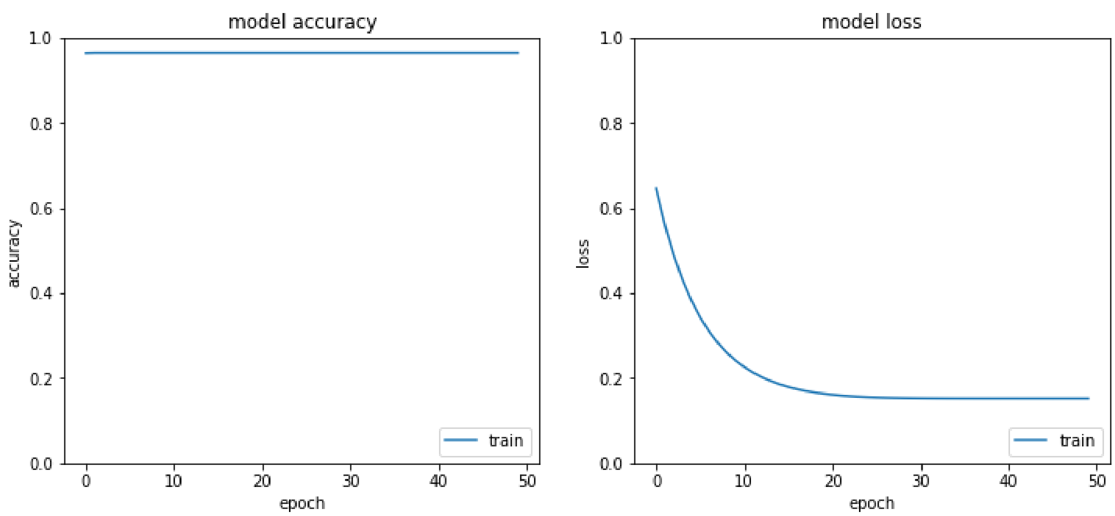
```
accuracy: 0.9649
Epoch 6/50
219/219 [==============================] - 0s 2ms/step - loss: 0.3432 -
accuracy: 0.9649
Epoch 7/50
219/219 [==============================] - 0s 2ms/step - loss: 0.3104 -
accuracy: 0.9649
Epoch 8/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2831 -
accuracy: 0.9649
Epoch 9/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2606 -
accuracy: 0.9649
Epoch 10/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2419 -
accuracy: 0.9649
Epoch 11/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2263 -
accuracy: 0.9649
Epoch 12/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2134 -
accuracy: 0.9649
Epoch 13/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2025 -
accuracy: 0.9649
Epoch 14/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1935 -
accuracy: 0.9649
Epoch 15/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1859 -
accuracy: 0.9649
Epoch 16/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1796 -
accuracy: 0.9649
Epoch 17/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1744 -
accuracy: 0.9649
Epoch 18/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1700 -
accuracy: 0.9649
Epoch 19/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1664 -
accuracy: 0.9649
Epoch 20/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1635 -
accuracy: 0.9649
Epoch 21/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1610 -
```

```
accuracy: 0.9649
Epoch 22/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1591 -
accuracy: 0.9649
Epoch 23/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1575 -
accuracy: 0.9649
Epoch 24/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1563 -
accuracy: 0.9649
Epoch 25/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1552 -
accuracy: 0.9649
Epoch 26/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1545 -
accuracy: 0.9649
Epoch 27/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1539 -
accuracy: 0.9649
Epoch 28/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1534 -
accuracy: 0.9649
Epoch 29/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1531 -
accuracy: 0.9649
Epoch 30/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1528 -
accuracy: 0.9649
Epoch 31/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1526 -
accuracy: 0.9649
Epoch 32/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1525 -
accuracy: 0.9649
Epoch 33/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1524 -
accuracy: 0.9649
Epoch 34/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1523 -
accuracy: 0.9649
Epoch 35/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1523 -
accuracy: 0.9649
Epoch 36/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1523 -
accuracy: 0.9649
Epoch 37/50
219/219 [==============================] - 1s 4ms/step - loss: 0.1522 -
```

```
accuracy: 0.9649
Epoch 38/50
219/219 [==============================] - 1s 3ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 39/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 40/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 41/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 42/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 43/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 44/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 45/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 46/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 47/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 48/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 49/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
Epoch 50/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1522 -
accuracy: 0.9649
94/94 - 0s - loss: 0.1385 - accuracy: 0.9690 - 245ms/epoch - 3ms/step

Test accuracy: 0.968999981880188

Test loss: 0.1384744942188263
```

train accuracy: 0.9649, loss: 0.1522 after 50 epochs test accuracy: 0.9690, loss: 0.1385

[ ]:
```
# plotting the performance of the model with the below parameters.
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.ylim(0, 1)
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.ylim(0, 1)
plt.show()
```



[ ]:
```
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 65

 dense_1 (Dense)             (None, 3)                 18
```

```
 dense_2 (Dense)             (None, 1)                 4

=================================================================
Total params: 87
Trainable params: 87
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
# checking the probabilities : not used but tried initially
probability_model = Sequential([classifier, tf.keras.layers.Softmax()])
predictions = probability_model.predict(X_test)
predictions[0]
```

[ ]:
```
array([1.], dtype=float32)
```

[ ]:
```
np.argmax(predictions[0])
```

[ ]:
```
0
```

[ ]:
```
y_test[0]
```

[ ]:
```
0
```

[ ]:
```
predictions
```

[ ]:
```
array([[1.],
       [1.],
       [1.],
       …,
       [1.],
       [1.],
       [1.]], dtype=float32)
```

[ ]:
```
y_test.nunique
```

[ ]:
```
<bound method IndexOpsMixin.nunique of 6252    0
4684    1
1731    0
4742    0
4521    0
       ..
8014    0
1074    0
3063    0
6487    0
4705    0
```

```
Name: Machine failure, Length: 3000, dtype: int64>
```

Experiment 2: Bayesian neural network (BNN)

dependencies and prerequisites

```
[ ]: from pprint import pprint
     import matplotlib.pyplot as plt
     import numpy as np
     import seaborn as sns

     import tensorflow.compat.v2 as tf
     tf.enable_v2_behavior()

     import tensorflow_probability as tfp

     sns.reset_defaults()
     sns.set_context(context='talk',font_scale=0.7)
     plt.rcParams['image.cmap'] = 'viridis'

     %matplotlib inline

     tfd = tfp.distributions
     tfb = tfp.bijectors
```

define priors and other functions

```
[ ]: # to build the bnn
```

define bnn functions and class

```
[ ]: from keras.wrappers.scikit_learn import KerasClassifier
     from sklearn.model_selection import cross_val_score
     from keras.models import Sequential # to initialize NN
     from keras.layers import Dense # to build layers
     '''
     classifier = Sequential()
     classifier.add(Dense(units = 8, input_dim = X_train.shape[1])) # changed this
     classifier.add(Dense(units = 4, activation = 'relu'))
     classifier.add(Dense(units = 1, activation = 'sigmoid'))
     classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
      ↪['accuracy'])
     classifier.fit(X_train, y_train, epochs=100)
     test_loss, test_acc = classifier.evaluate(X_test,  y_test, verbose=2)
     print('\nTest accuracy:', test_acc)
```

```
'''
```

```
"\nclassifier = Sequential()\nclassifier.add(Dense(units = 8, input_dim =
X_train.shape[1])) # changed this\nclassifier.add(Dense(units = 4, activation =
'relu'))\nclassifier.add(Dense(units = 1, activation =
'sigmoid'))\nclassifier.compile(optimizer = 'adam', loss =
'binary_crossentropy', metrics = ['accuracy'])\nclassifier.fit(X_train, y_train,
epochs=100)\ntest_loss, test_acc = classifier.evaluate(X_test,  y_test,
verbose=2)\nprint('\nTest accuracy:', test_acc)\n\n"
```

target is machine failure variable

```
from sklearn.model_selection import train_test_split
# resetting the data to initial dataset
#first moving target variable "Machine Failure" to end and then defining X and y
df = df[['Type', 'Air temperature [K]', 'Process temperature [K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]',
        'TWF', 'HDF', 'PWF', 'OSF', 'RNF','Machine failure']]
print(df.shape)
# excluding last variable for target variable
X = df.iloc[:, :-1]
print(X.shape)
# making last variable as target variable
y = df.iloc[:, -1]
print(y.shape)
# using 70:30 split for making training and testing datasets and using random␣
↪state as 42 to repeat this random split.
X_train,X_test,y_train,y_test = train_test_split(X, y,test_size=0.
↪3,random_state=42)
```

```
(10000, 12)
(10000, 11)
(10000,)
```

```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
↪cast(dataset_size, dtype=tf.float32))
# a bnn model with 3 layers which are denseflipout layers
model_tfp = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(16, kernel_divergence_fn=kl_divergence_function),#,␣
↪activation=tf.nn.relu),
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
↪activation=tf.nn.softmax),
```

```
])

learning_rate = 1e-06#0.001
model_tfp.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:
```
model_tfp.fit(X_train, y_train, epochs=50)
test_loss, test_acc = model_tfp.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
```

```
Epoch 1/50
219/219 [==============================] - 2s 2ms/step - loss: 4.9793 -
accuracy: 0.6621
Epoch 2/50
219/219 [==============================] - 0s 2ms/step - loss: 4.9491 -
accuracy: 0.6650
Epoch 3/50
219/219 [==============================] - 0s 2ms/step - loss: 4.9786 -
accuracy: 0.6446
Epoch 4/50
219/219 [==============================] - 0s 2ms/step - loss: 4.9533 -
accuracy: 0.6479
Epoch 5/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8661 -
accuracy: 0.6417
Epoch 6/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8756 -
accuracy: 0.6359
Epoch 7/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8554 -
accuracy: 0.6444
Epoch 8/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8155 -
accuracy: 0.6340
Epoch 9/50
219/219 [==============================] - 0s 2ms/step - loss: 4.9210 -
```

```
accuracy: 0.6321
Epoch 10/50
219/219 [==============================] - 0s 2ms/step - loss: 4.7639 -
accuracy: 0.6317
Epoch 11/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8633 -
accuracy: 0.6217
Epoch 12/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8131 -
accuracy: 0.6276
Epoch 13/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8182 -
accuracy: 0.6276
Epoch 14/50
219/219 [==============================] - 0s 2ms/step - loss: 4.7730 -
accuracy: 0.6090
Epoch 15/50
219/219 [==============================] - 0s 2ms/step - loss: 4.8059 -
accuracy: 0.6011
Epoch 16/50
219/219 [==============================] - 1s 2ms/step - loss: 4.8013 -
accuracy: 0.6164
Epoch 17/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6785 -
accuracy: 0.5919
Epoch 18/50
219/219 [==============================] - 0s 2ms/step - loss: 4.7840 -
accuracy: 0.5953
Epoch 19/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6446 -
accuracy: 0.6099
Epoch 20/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6887 -
accuracy: 0.6200
Epoch 21/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6245 -
accuracy: 0.6084
Epoch 22/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6764 -
accuracy: 0.6007
Epoch 23/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6526 -
accuracy: 0.6057
Epoch 24/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6039 -
accuracy: 0.6057
Epoch 25/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6111 -
```

```
accuracy: 0.5969
Epoch 26/50
219/219 [==============================] - 0s 2ms/step - loss: 4.5355 -
accuracy: 0.5987
Epoch 27/50
219/219 [==============================] - 0s 2ms/step - loss: 4.5705 -
accuracy: 0.5894
Epoch 28/50
219/219 [==============================] - 0s 2ms/step - loss: 4.6398 -
accuracy: 0.5846
Epoch 29/50
219/219 [==============================] - 0s 2ms/step - loss: 4.5473 -
accuracy: 0.5874
Epoch 30/50
219/219 [==============================] - 0s 2ms/step - loss: 4.4550 -
accuracy: 0.5816
Epoch 31/50
219/219 [==============================] - 0s 2ms/step - loss: 4.4687 -
accuracy: 0.5881
Epoch 32/50
219/219 [==============================] - 0s 2ms/step - loss: 4.5343 -
accuracy: 0.5857
Epoch 33/50
219/219 [==============================] - 0s 2ms/step - loss: 4.5332 -
accuracy: 0.5893
Epoch 34/50
219/219 [==============================] - 0s 2ms/step - loss: 4.4869 -
accuracy: 0.5900
Epoch 35/50
219/219 [==============================] - 0s 2ms/step - loss: 4.3845 -
accuracy: 0.5839
Epoch 36/50
219/219 [==============================] - 0s 2ms/step - loss: 4.5090 -
accuracy: 0.5820
Epoch 37/50
219/219 [==============================] - 0s 2ms/step - loss: 4.3882 -
accuracy: 0.5801
Epoch 38/50
219/219 [==============================] - 0s 2ms/step - loss: 4.4900 -
accuracy: 0.5736
Epoch 39/50
219/219 [==============================] - 0s 2ms/step - loss: 4.4199 -
accuracy: 0.5743
Epoch 40/50
219/219 [==============================] - 0s 2ms/step - loss: 4.3626 -
accuracy: 0.5626
Epoch 41/50
219/219 [==============================] - 0s 2ms/step - loss: 4.3990 -
```

```
accuracy: 0.5817
Epoch 42/50
219/219 [==============================] - 0s 2ms/step - loss: 4.2596 -
accuracy: 0.5711
Epoch 43/50
219/219 [==============================] - 0s 2ms/step - loss: 4.2012 -
accuracy: 0.5730
Epoch 44/50
219/219 [==============================] - 0s 2ms/step - loss: 4.2389 -
accuracy: 0.5759
Epoch 45/50
219/219 [==============================] - 0s 2ms/step - loss: 4.2572 -
accuracy: 0.5731
Epoch 46/50
219/219 [==============================] - 1s 2ms/step - loss: 4.3044 -
accuracy: 0.5679
Epoch 47/50
219/219 [==============================] - 0s 2ms/step - loss: 4.1681 -
accuracy: 0.5594
Epoch 48/50
219/219 [==============================] - 1s 3ms/step - loss: 4.2756 -
accuracy: 0.5629
Epoch 49/50
219/219 [==============================] - 1s 2ms/step - loss: 4.1925 -
accuracy: 0.5636
Epoch 50/50
219/219 [==============================] - 0s 2ms/step - loss: 4.2051 -
accuracy: 0.5669
94/94 [==============================] - 1s 2ms/step - loss: 4.1917 - accuracy:
0.5703

Test accuracy: 0.5703333616256714

Test loss: 4.19170618057251
```

Test accuracy: 0.968666672706604 after 50 epochs and test loss: 0.450

[ ]:
```
history = model_tfp.fit(np.asarray(X_train), np.asarray(y_train),epochs=50)#,␣
↪validation_split=0.3,  shuffle=True)
```

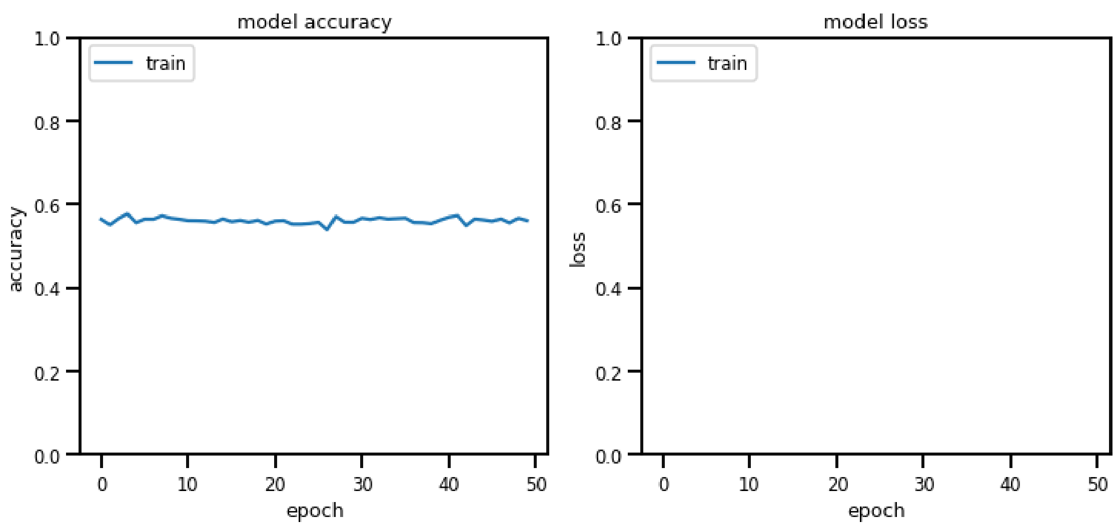
```
Epoch 1/50
219/219 [==============================] - 2s 3ms/step - loss: 4.1039 -
accuracy: 0.5636
Epoch 2/50
219/219 [==============================] - 1s 3ms/step - loss: 4.1626 -
accuracy: 0.5506
Epoch 3/50
219/219 [==============================] - 1s 3ms/step - loss: 4.2166 -
```

```
accuracy: 0.5656
Epoch 4/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0658 -
accuracy: 0.5766
Epoch 5/50
219/219 [==============================] - 1s 3ms/step - loss: 4.2160 -
accuracy: 0.5556
Epoch 6/50
219/219 [==============================] - 1s 3ms/step - loss: 4.1678 -
accuracy: 0.5640
Epoch 7/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0427 -
accuracy: 0.5639
Epoch 8/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0795 -
accuracy: 0.5720
Epoch 9/50
219/219 [==============================] - 1s 3ms/step - loss: 3.9967 -
accuracy: 0.5663
Epoch 10/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0352 -
accuracy: 0.5637
Epoch 11/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0163 -
accuracy: 0.5604
Epoch 12/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0071 -
accuracy: 0.5601
Epoch 13/50
219/219 [==============================] - 1s 3ms/step - loss: 4.0005 -
accuracy: 0.5593
Epoch 14/50
219/219 [==============================] - 1s 4ms/step - loss: 4.0057 -
accuracy: 0.5561
Epoch 15/50
219/219 [==============================] - 2s 7ms/step - loss: 3.9246 -
accuracy: 0.5646
Epoch 16/50
219/219 [==============================] - 1s 6ms/step - loss: 3.8869 -
accuracy: 0.5580
Epoch 17/50
219/219 [==============================] - 1s 5ms/step - loss: 3.9229 -
accuracy: 0.5610
Epoch 18/50
219/219 [==============================] - 1s 6ms/step - loss: 3.8767 -
accuracy: 0.5567
Epoch 19/50
219/219 [==============================] - 1s 6ms/step - loss: 3.8720 -
```

```
accuracy: 0.5613
Epoch 20/50
219/219 [==============================] - 1s 5ms/step - loss: 3.7971 -
accuracy: 0.5523
Epoch 21/50
219/219 [==============================] - 2s 7ms/step - loss: 3.8586 -
accuracy: 0.5594
Epoch 22/50
219/219 [==============================] - 1s 5ms/step - loss: 3.7703 -
accuracy: 0.5604
Epoch 23/50
219/219 [==============================] - 1s 3ms/step - loss: 3.8136 -
accuracy: 0.5520
Epoch 24/50
219/219 [==============================] - 1s 3ms/step - loss: 3.8157 -
accuracy: 0.5520
Epoch 25/50
219/219 [==============================] - 1s 3ms/step - loss: 3.7007 -
accuracy: 0.5534
Epoch 26/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6814 -
accuracy: 0.5566
Epoch 27/50
219/219 [==============================] - 1s 3ms/step - loss: 3.7446 -
accuracy: 0.5384
Epoch 28/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6257 -
accuracy: 0.5701
Epoch 29/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6237 -
accuracy: 0.5569
Epoch 30/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6199 -
accuracy: 0.5566
Epoch 31/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6670 -
accuracy: 0.5664
Epoch 32/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6639 -
accuracy: 0.5633
Epoch 33/50
219/219 [==============================] - 1s 3ms/step - loss: 3.5723 -
accuracy: 0.5677
Epoch 34/50
219/219 [==============================] - 1s 3ms/step - loss: 3.5594 -
accuracy: 0.5643
Epoch 35/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4878 -
```

```
accuracy: 0.5654
Epoch 36/50
219/219 [==============================] - 1s 3ms/step - loss: 3.6038 -
accuracy: 0.5666
Epoch 37/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4938 -
accuracy: 0.5560
Epoch 38/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4517 -
accuracy: 0.5556
Epoch 39/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4588 -
accuracy: 0.5533
Epoch 40/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4524 -
accuracy: 0.5616
Epoch 41/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4106 -
accuracy: 0.5684
Epoch 42/50
219/219 [==============================] - 1s 3ms/step - loss: 3.4266 -
accuracy: 0.5724
Epoch 43/50
219/219 [==============================] - 1s 3ms/step - loss: 3.3977 -
accuracy: 0.5484
Epoch 44/50
219/219 [==============================] - 2s 8ms/step - loss: 3.3903 -
accuracy: 0.5643
Epoch 45/50
219/219 [==============================] - 1s 3ms/step - loss: 3.3331 -
accuracy: 0.5620
Epoch 46/50
219/219 [==============================] - 1s 3ms/step - loss: 3.3223 -
accuracy: 0.5590
Epoch 47/50
219/219 [==============================] - 1s 3ms/step - loss: 3.3016 -
accuracy: 0.5644
Epoch 48/50
219/219 [==============================] - 1s 3ms/step - loss: 3.2589 -
accuracy: 0.5551
Epoch 49/50
219/219 [==============================] - 1s 3ms/step - loss: 3.1965 -
accuracy: 0.5661
Epoch 50/50
219/219 [==============================] - 1s 3ms/step - loss: 3.2328 -
accuracy: 0.5606
```

[ ]:
```
# doing all the same steps of building model, fitting it to the data and␣
↪evaluating it and plotting parameters for all the models built in the␣
↪notebook.
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
plt.show()
```



[ ]:
```
model_tfp.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout (DenseFlipout  (None, 16)               368
```

```
 )

 dense_flipout_1 (DenseFlipo  (None, 6)                198
 ut)

 dense_flipout_2 (DenseFlipo  (None, 2)                26
 ut)

=================================================================
Total params: 592
Trainable params: 592
Non-trainable params: 0
_________________________________________________________________
```

doing all the same steps of building model, fitting it to the data and evaluating it and plotting parameters for all the models built in the notebook.

define tensorboard variables for we plots

Train BNN with a small training subset.

Train BNN with the whole training set. building different versions of bnn with different parameters.

Steps done in implementing all kind of bnns * Building a model * Fitting the model on the data * Evaluating the model * Plotting different parameters of the model for comparision * saving the model as a file * saving the model architecture as a image All these models with different versions in it as described below.

EXP VBNN:

[ ]:
```
dist = tfp.distributions
dataset_size = len(X_train)
#defining kl_divergence function
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))
#defining model
model_tfp_v1 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])
# compiling the model
learning_rate = 1e-06 #0.002
```

```
model_tfp_v1.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:
```
from keras.utils.vis_utils import plot_model
```

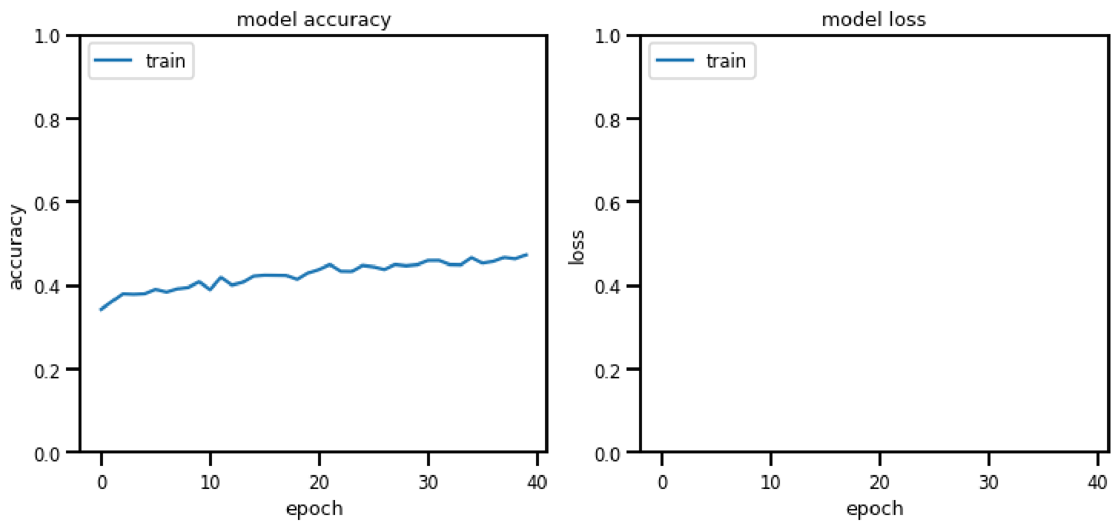
[ ]:
```
#fitting the model on the training data
history = model_tfp_v1.fit(X_train, y_train,␣
 ↪epochs=40)#,batch_size=1,validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
#evaluating the model on the test dataset
test_loss, test_acc = model_tfp_v1.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
# TRY REMOVING THE VALIDATION PART FROM THE FIT
# validation_data = (np.asarray(X_test), np.asarray(y_test))
#history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=50,validation_split=0.2, shuffle=True)
# to see history:
# list all data in history
print(history.history.keys())
# summarize history for accuracy

#normal_bnn_model.save('model_tfp_v1.h5')
#normal_bnn_model.save('saved_model/model_tfp_v1')
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
```

```
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
plt.show()
plot_model(model_tfp_v1, to_file='model_plot.png', show_shapes=True,␣
↪show_layer_names=True)
```

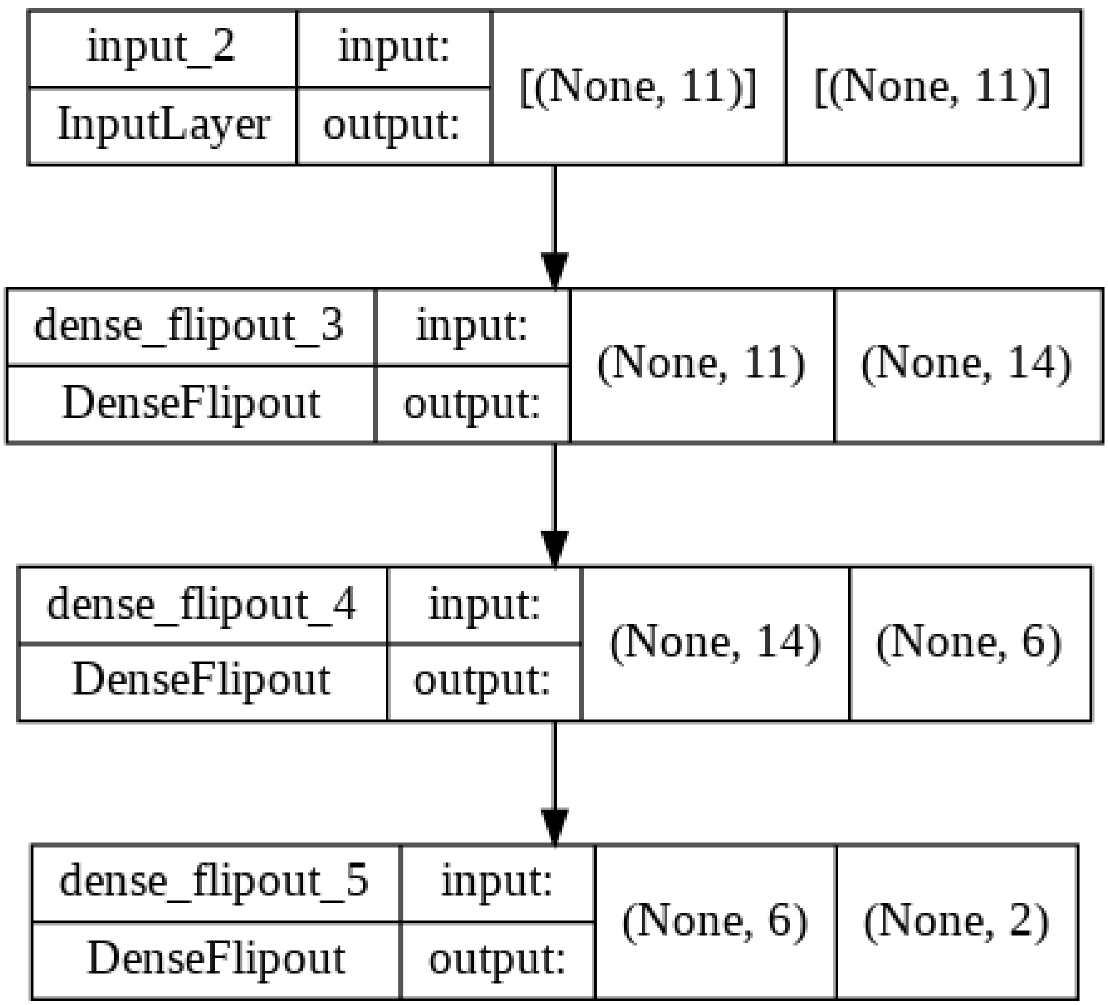
```
Epoch 1/40
219/219 [==============================] - 3s 3ms/step - loss: 4.3856 -
accuracy: 0.3429
Epoch 2/40
219/219 [==============================] - 1s 3ms/step - loss: 4.4277 -
accuracy: 0.3613
Epoch 3/40
219/219 [==============================] - 1s 3ms/step - loss: 4.4439 -
accuracy: 0.3797
Epoch 4/40
219/219 [==============================] - 1s 4ms/step - loss: 4.3698 -
accuracy: 0.3787
Epoch 5/40
219/219 [==============================] - 1s 3ms/step - loss: 4.4050 -
accuracy: 0.3800
Epoch 6/40
219/219 [==============================] - 1s 3ms/step - loss: 4.3590 -
accuracy: 0.3906
Epoch 7/40
219/219 [==============================] - 1s 3ms/step - loss: 4.3491 -
accuracy: 0.3841
Epoch 8/40
219/219 [==============================] - 1s 3ms/step - loss: 4.3033 -
accuracy: 0.3919
Epoch 9/40
219/219 [==============================] - 1s 3ms/step - loss: 4.2803 -
accuracy: 0.3951
Epoch 10/40
219/219 [==============================] - 1s 2ms/step - loss: 4.2487 -
accuracy: 0.4084
Epoch 11/40
219/219 [==============================] - 1s 3ms/step - loss: 4.2922 -
accuracy: 0.3897
Epoch 12/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1837 -
accuracy: 0.4189
Epoch 13/40
```

```
219/219 [==============================] - 1s 3ms/step - loss: 4.2391 -
accuracy: 0.4007
Epoch 14/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1435 -
accuracy: 0.4073
Epoch 15/40
219/219 [==============================] - 1s 3ms/step - loss: 4.2099 -
accuracy: 0.4221
Epoch 16/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1831 -
accuracy: 0.4243
Epoch 17/40
219/219 [==============================] - 1s 3ms/step - loss: 4.2688 -
accuracy: 0.4240
Epoch 18/40
219/219 [==============================] - 1s 3ms/step - loss: 4.2200 -
accuracy: 0.4236
Epoch 19/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1437 -
accuracy: 0.4141
Epoch 20/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1588 -
accuracy: 0.4297
Epoch 21/40
219/219 [==============================] - 1s 3ms/step - loss: 4.2099 -
accuracy: 0.4380
Epoch 22/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1639 -
accuracy: 0.4496
Epoch 23/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1884 -
accuracy: 0.4339
Epoch 24/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0153 -
accuracy: 0.4334
Epoch 25/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0867 -
accuracy: 0.4476
Epoch 26/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1144 -
accuracy: 0.4443
Epoch 27/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0698 -
accuracy: 0.4380
Epoch 28/40
219/219 [==============================] - 1s 3ms/step - loss: 4.1419 -
accuracy: 0.4499
Epoch 29/40
```

```
219/219 [==============================] - 1s 2ms/step - loss: 3.9630 -
accuracy: 0.4467
Epoch 30/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0868 -
accuracy: 0.4494
Epoch 31/40
219/219 [==============================] - 1s 2ms/step - loss: 4.0406 -
accuracy: 0.4594
Epoch 32/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0591 -
accuracy: 0.4596
Epoch 33/40
219/219 [==============================] - 1s 3ms/step - loss: 3.9526 -
accuracy: 0.4497
Epoch 34/40
219/219 [==============================] - 1s 2ms/step - loss: 4.0730 -
accuracy: 0.4491
Epoch 35/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0146 -
accuracy: 0.4663
Epoch 36/40
219/219 [==============================] - 1s 3ms/step - loss: 3.9890 -
accuracy: 0.4531
Epoch 37/40
219/219 [==============================] - 1s 3ms/step - loss: 4.0057 -
accuracy: 0.4573
Epoch 38/40
219/219 [==============================] - 1s 2ms/step - loss: 3.9413 -
accuracy: 0.4667
Epoch 39/40
219/219 [==============================] - 1s 2ms/step - loss: 3.9652 -
accuracy: 0.4634
Epoch 40/40
219/219 [==============================] - 1s 2ms/step - loss: 3.9933 -
accuracy: 0.4729
94/94 [==============================] - 1s 2ms/step - loss: 3.9907 - accuracy:
0.4857

Test accuracy: 0.4856666624546051

Test loss: 3.9907209873199463
dict_keys(['loss', 'accuracy'])
```


model accuracy
model loss
train
accuracy
loss
epoch

[ ]:


input_2
InputLayer
input:
output:
[(None, 11)]
[(None, 11)]
dense_flipout_3
DenseFlipout
input:
output:
(None, 11)
(None, 14)
dense_flipout_4
DenseFlipout
input:
output:
(None, 14)
(None, 6)
dense_flipout_5
DenseFlipout
input:
output:
(None, 6)
(None, 2)

```
[ ]: #prints model summary
model_tfp_v1.summary()
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_3 (DenseFlipo  (None, 14)               322
 ut)

 dense_flipout_4 (DenseFlipo  (None, 6)                174
 ut)

 dense_flipout_5 (DenseFlipo  (None, 2)                26
 ut)

=================================================================
Total params: 522
Trainable params: 522
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]: !pip install pickle5
```

```
Collecting pickle5
  Downloading
pickle5-0.0.12-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (256 kB)
     |                  | 256 kB 5.2 MB/s
Installing collected packages: pickle5
Successfully installed pickle5-0.0.12
```

```
[ ]: import pickle
# used to save model as a pkl file and can be loaded anywhere ans used directly␣
↪with required packages.
filename = 'model_tfp1v1.pkl'
tf.saved_model.SaveOptions(
    namespace_whitelist=None, save_debug_info=False, function_aliases=None,
    experimental_io_device=None, experimental_variable_policy=None,
    experimental_custom_gradients=True
)
pickle.dump(model_tfp_v1, open(filename, 'wb'))
```

```
INFO:tensorflow:Assets written to:
ram://8d4e6946-7a50-422b-b9af-069382b34d78/assets
```

```
[ ]: !mkdir -p saved_model
```

[ ]:
```
#saving tensorflow model of version v1 to drive. download this and place it in
#streamlit local folder and load it using tensorflow load model
model_tfp_v1.save('saved_model/model_tfp_v1')
```

```
INFO:tensorflow:Assets written to: saved_model/model_tfp_v1/assets
```

[ ]:
```
#saving model into hdf5 format and load the same file using same loadmodel␣
↪function
model_tfp_v1.save('model_tfp_v1.h5')
```

[ ]:
```
# use this to load the model into local
new_model = tf.keras.models.load_model('saved_model/model_tfp_v1')

# Check its architecture
new_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_3 (DenseFlipo  (None, 14)               322
 ut)

 dense_flipout_4 (DenseFlipo  (None, 6)                174
 ut)

 dense_flipout_5 (DenseFlipo  (None, 2)                26
 ut)

=================================================================
Total params: 522
Trainable params: 522
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
!pip3 install ann_visualizer
!pip install graphviz
```

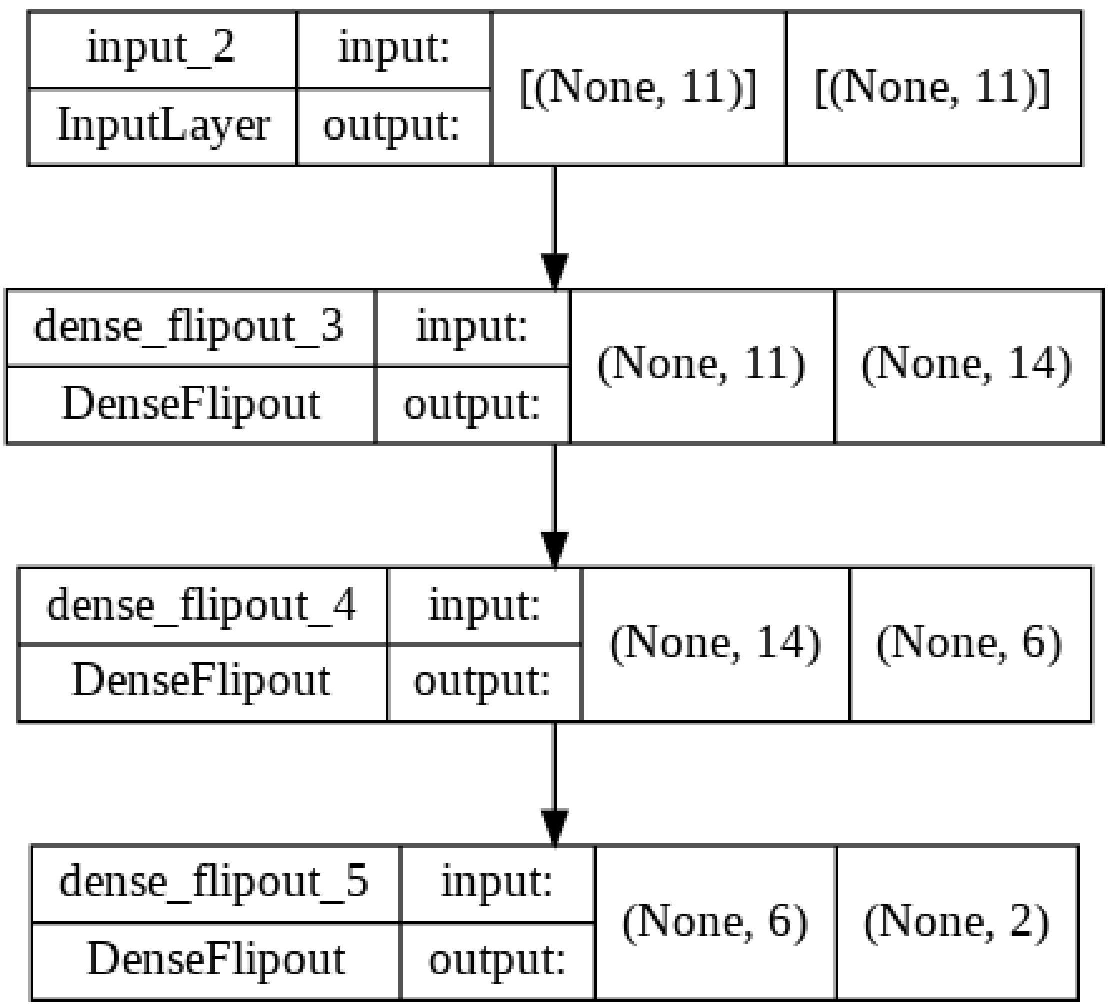
```
Collecting ann_visualizer
  Downloading ann_visualizer-2.5.tar.gz (4.7 kB)
Building wheels for collected packages: ann-visualizer
  Building wheel for ann-visualizer (setup.py) … done
  Created wheel for ann-visualizer: filename=ann_visualizer-2.5-py3-none-any.whl
size=4168
sha256=b077497bbac09fbb5f71616102570cb3f8a48d43d5df1a330b144c0d1818c72d
  Stored in directory: /root/.cache/pip/wheels/1b/fc/58/2ab1c3b30350105929308bec
ddda4fb59b1358e54f985e1f4a
```

```
Successfully built ann-visualizer
Installing collected packages: ann-visualizer
Successfully installed ann-visualizer-2.5
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-
packages (0.10.1)
```

[ ]:
```python
from ann_visualizer.visualize import ann_viz;

#ann_viz(new_model, title="My first neural network")
```

[ ]:
```python
from keras.utils.vis_utils import plot_model
#trying to save model architecture as an image.
# tried with different one but its not supporting the tfp layers, so just only this one.
plot_model(new_model, to_file='model_plot1.png', show_shapes=True, show_layer_names=True)
```

[ ]:



v2

```
[ ]: dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_tfp_v2 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06# 0.005
model_tfp_v2.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])

history = model_tfp_v2.fit(X_train, y_train,␣
 ↪epochs=80)#,batch_size=1,validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
test_loss, test_acc = model_tfp_v2.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)

# TRY REMOVING THE VALIDATION PART FROM THE FIT
# validation_data = (np.asarray(X_test), np.asarray(y_test))
#history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=50,validation_split=0.2, shuffle=True)
# to see history:
# list all data in history
print(history.history.keys())
# summarize history for accuracy

model_tfp_v2.save('model_tfp_v2.h5')
model_tfp_v2.save('saved_model/model_tfp_v2')
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
```

```
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
plt.show()
plot_model(model_tfp_v2, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)

model_tfp_v2.summary()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

```
Epoch 1/80
219/219 [==============================] - 2s 3ms/step - loss: 6.2867 -
accuracy: 0.8781
Epoch 2/80
219/219 [==============================] - 1s 3ms/step - loss: 6.2511 -
accuracy: 0.8799
Epoch 3/80
219/219 [==============================] - 1s 3ms/step - loss: 6.2813 -
accuracy: 0.8864
Epoch 4/80
219/219 [==============================] - 1s 2ms/step - loss: 6.1645 -
accuracy: 0.8680
Epoch 5/80
219/219 [==============================] - 1s 3ms/step - loss: 6.1916 -
accuracy: 0.8777
Epoch 6/80
219/219 [==============================] - 1s 3ms/step - loss: 6.1933 -
accuracy: 0.8753
Epoch 7/80
219/219 [==============================] - 1s 3ms/step - loss: 6.1128 -
accuracy: 0.8589
Epoch 8/80
```

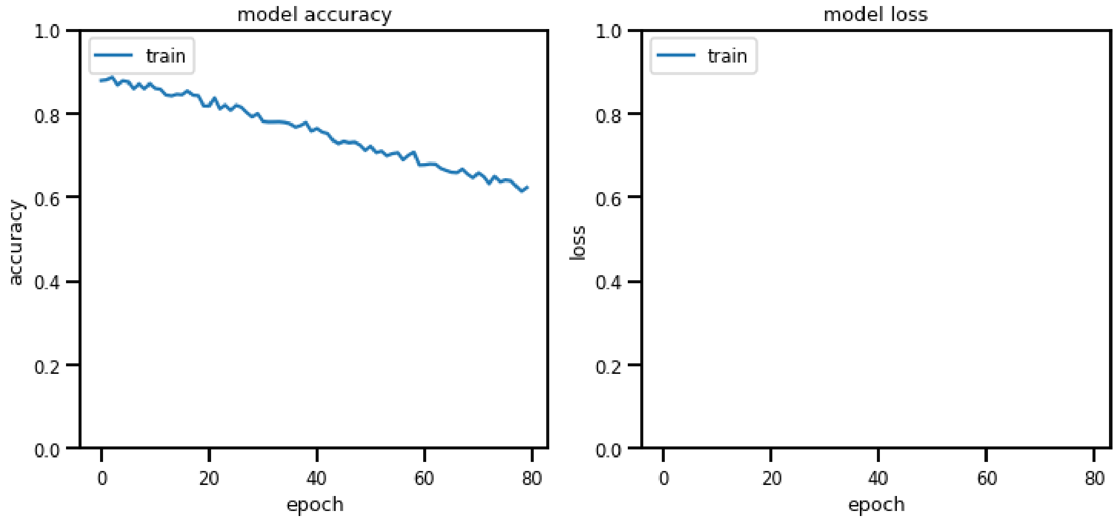
```
219/219 [==============================] - 1s 2ms/step - loss: 6.1383 -
accuracy: 0.8704
Epoch 9/80
219/219 [==============================] - 1s 2ms/step - loss: 6.0257 -
accuracy: 0.8589
Epoch 10/80
219/219 [==============================] - 1s 2ms/step - loss: 6.0200 -
accuracy: 0.8709
Epoch 11/80
219/219 [==============================] - 1s 3ms/step - loss: 6.0165 -
accuracy: 0.8599
Epoch 12/80
219/219 [==============================] - 1s 3ms/step - loss: 5.9660 -
accuracy: 0.8579
Epoch 13/80
219/219 [==============================] - 1s 3ms/step - loss: 5.8620 -
accuracy: 0.8440
Epoch 14/80
219/219 [==============================] - 1s 3ms/step - loss: 5.8566 -
accuracy: 0.8417
Epoch 15/80
219/219 [==============================] - 1s 3ms/step - loss: 5.8994 -
accuracy: 0.8456
Epoch 16/80
219/219 [==============================] - 1s 3ms/step - loss: 5.8752 -
accuracy: 0.8444
Epoch 17/80
219/219 [==============================] - 1s 4ms/step - loss: 5.8391 -
accuracy: 0.8539
Epoch 18/80
219/219 [==============================] - 1s 5ms/step - loss: 5.8254 -
accuracy: 0.8434
Epoch 19/80
219/219 [==============================] - 1s 5ms/step - loss: 5.7850 -
accuracy: 0.8423
Epoch 20/80
219/219 [==============================] - 1s 5ms/step - loss: 5.6687 -
accuracy: 0.8183
Epoch 21/80
219/219 [==============================] - 1s 6ms/step - loss: 5.6504 -
accuracy: 0.8179
Epoch 22/80
219/219 [==============================] - 1s 6ms/step - loss: 5.7032 -
accuracy: 0.8369
Epoch 23/80
219/219 [==============================] - 1s 7ms/step - loss: 5.6071 -
accuracy: 0.8110
Epoch 24/80
```

```
219/219 [==============================] - 1s 7ms/step - loss: 5.6081 -
accuracy: 0.8201
Epoch 25/80
219/219 [==============================] - 1s 6ms/step - loss: 5.5794 -
accuracy: 0.8079
Epoch 26/80
219/219 [==============================] - 1s 5ms/step - loss: 5.6360 -
accuracy: 0.8196
Epoch 27/80
219/219 [==============================] - 1s 6ms/step - loss: 5.5630 -
accuracy: 0.8150
Epoch 28/80
219/219 [==============================] - 1s 6ms/step - loss: 5.5653 -
accuracy: 0.8023
Epoch 29/80
219/219 [==============================] - 2s 7ms/step - loss: 5.4478 -
accuracy: 0.7917
Epoch 30/80
219/219 [==============================] - 2s 7ms/step - loss: 5.4267 -
accuracy: 0.7997
Epoch 31/80
219/219 [==============================] - 1s 6ms/step - loss: 5.4070 -
accuracy: 0.7810
Epoch 32/80
219/219 [==============================] - 1s 7ms/step - loss: 5.4115 -
accuracy: 0.7796
Epoch 33/80
219/219 [==============================] - 2s 7ms/step - loss: 5.3812 -
accuracy: 0.7797
Epoch 34/80
219/219 [==============================] - 1s 6ms/step - loss: 5.3153 -
accuracy: 0.7800
Epoch 35/80
219/219 [==============================] - 1s 6ms/step - loss: 5.2641 -
accuracy: 0.7787
Epoch 36/80
219/219 [==============================] - 1s 5ms/step - loss: 5.2675 -
accuracy: 0.7757
Epoch 37/80
219/219 [==============================] - 2s 7ms/step - loss: 5.2714 -
accuracy: 0.7671
Epoch 38/80
219/219 [==============================] - 1s 7ms/step - loss: 5.2656 -
accuracy: 0.7713
Epoch 39/80
219/219 [==============================] - 1s 5ms/step - loss: 5.2436 -
accuracy: 0.7784
Epoch 40/80
```

```
219/219 [==============================] - 1s 4ms/step - loss: 5.2184 -
accuracy: 0.7577
Epoch 41/80
219/219 [==============================] - 1s 6ms/step - loss: 5.1765 -
accuracy: 0.7643
Epoch 42/80
219/219 [==============================] - 1s 6ms/step - loss: 5.1540 -
accuracy: 0.7554
Epoch 43/80
219/219 [==============================] - 1s 5ms/step - loss: 5.0500 -
accuracy: 0.7520
Epoch 44/80
219/219 [==============================] - 1s 5ms/step - loss: 5.0609 -
accuracy: 0.7356
Epoch 45/80
219/219 [==============================] - 1s 5ms/step - loss: 5.0547 -
accuracy: 0.7279
Epoch 46/80
219/219 [==============================] - 1s 5ms/step - loss: 5.0463 -
accuracy: 0.7333
Epoch 47/80
219/219 [==============================] - 1s 5ms/step - loss: 5.0088 -
accuracy: 0.7300
Epoch 48/80
219/219 [==============================] - 1s 5ms/step - loss: 5.0331 -
accuracy: 0.7314
Epoch 49/80
219/219 [==============================] - 1s 5ms/step - loss: 4.9963 -
accuracy: 0.7247
Epoch 50/80
219/219 [==============================] - 1s 6ms/step - loss: 4.9486 -
accuracy: 0.7117
Epoch 51/80
219/219 [==============================] - 1s 6ms/step - loss: 4.9179 -
accuracy: 0.7220
Epoch 52/80
219/219 [==============================] - 1s 6ms/step - loss: 4.8708 -
accuracy: 0.7063
Epoch 53/80
219/219 [==============================] - 1s 5ms/step - loss: 4.8973 -
accuracy: 0.7100
Epoch 54/80
219/219 [==============================] - 1s 4ms/step - loss: 4.8066 -
accuracy: 0.6986
Epoch 55/80
219/219 [==============================] - 1s 3ms/step - loss: 4.8338 -
accuracy: 0.7036
Epoch 56/80
```

```
219/219 [==============================] - 1s 3ms/step - loss: 4.7968 -
accuracy: 0.7061
Epoch 57/80
219/219 [==============================] - 1s 3ms/step - loss: 4.7918 -
accuracy: 0.6889
Epoch 58/80
219/219 [==============================] - 1s 3ms/step - loss: 4.7320 -
accuracy: 0.6997
Epoch 59/80
219/219 [==============================] - 1s 3ms/step - loss: 4.7334 -
accuracy: 0.7074
Epoch 60/80
219/219 [==============================] - 1s 3ms/step - loss: 4.6935 -
accuracy: 0.6769
Epoch 61/80
219/219 [==============================] - 1s 3ms/step - loss: 4.6884 -
accuracy: 0.6774
Epoch 62/80
219/219 [==============================] - 1s 3ms/step - loss: 4.7875 -
accuracy: 0.6793
Epoch 63/80
219/219 [==============================] - 1s 3ms/step - loss: 4.6492 -
accuracy: 0.6787
Epoch 64/80
219/219 [==============================] - 1s 3ms/step - loss: 4.7007 -
accuracy: 0.6687
Epoch 65/80
219/219 [==============================] - 1s 3ms/step - loss: 4.5833 -
accuracy: 0.6639
Epoch 66/80
219/219 [==============================] - 1s 3ms/step - loss: 4.5888 -
accuracy: 0.6594
Epoch 67/80
219/219 [==============================] - 1s 4ms/step - loss: 4.6061 -
accuracy: 0.6584
Epoch 68/80
219/219 [==============================] - 1s 3ms/step - loss: 4.6037 -
accuracy: 0.6671
Epoch 69/80
219/219 [==============================] - 1s 3ms/step - loss: 4.5637 -
accuracy: 0.6549
Epoch 70/80
219/219 [==============================] - 1s 3ms/step - loss: 4.5775 -
accuracy: 0.6460
Epoch 71/80
219/219 [==============================] - 1s 2ms/step - loss: 4.5473 -
accuracy: 0.6579
Epoch 72/80
```

```
219/219 [==============================] - 1s 2ms/step - loss: 4.5030 -
accuracy: 0.6486
Epoch 73/80
219/219 [==============================] - 1s 3ms/step - loss: 4.3652 -
accuracy: 0.6321
Epoch 74/80
219/219 [==============================] - 1s 3ms/step - loss: 4.4360 -
accuracy: 0.6497
Epoch 75/80
219/219 [==============================] - 1s 3ms/step - loss: 4.4118 -
accuracy: 0.6366
Epoch 76/80
219/219 [==============================] - 1s 3ms/step - loss: 4.3829 -
accuracy: 0.6409
Epoch 77/80
219/219 [==============================] - 1s 3ms/step - loss: 4.3912 -
accuracy: 0.6390
Epoch 78/80
219/219 [==============================] - 1s 3ms/step - loss: 4.3847 -
accuracy: 0.6261
Epoch 79/80
219/219 [==============================] - 1s 2ms/step - loss: 4.3971 -
accuracy: 0.6140
Epoch 80/80
219/219 [==============================] - 1s 2ms/step - loss: 4.3793 -
accuracy: 0.6231
94/94 [==============================] - 1s 2ms/step - loss: 4.3410 - accuracy:
0.6287

Test accuracy: 0.6286666393280029

Test loss: 4.340991020202637
dict_keys(['loss', 'accuracy'])
INFO:tensorflow:Assets written to: saved_model/model_tfp_v2/assets
```

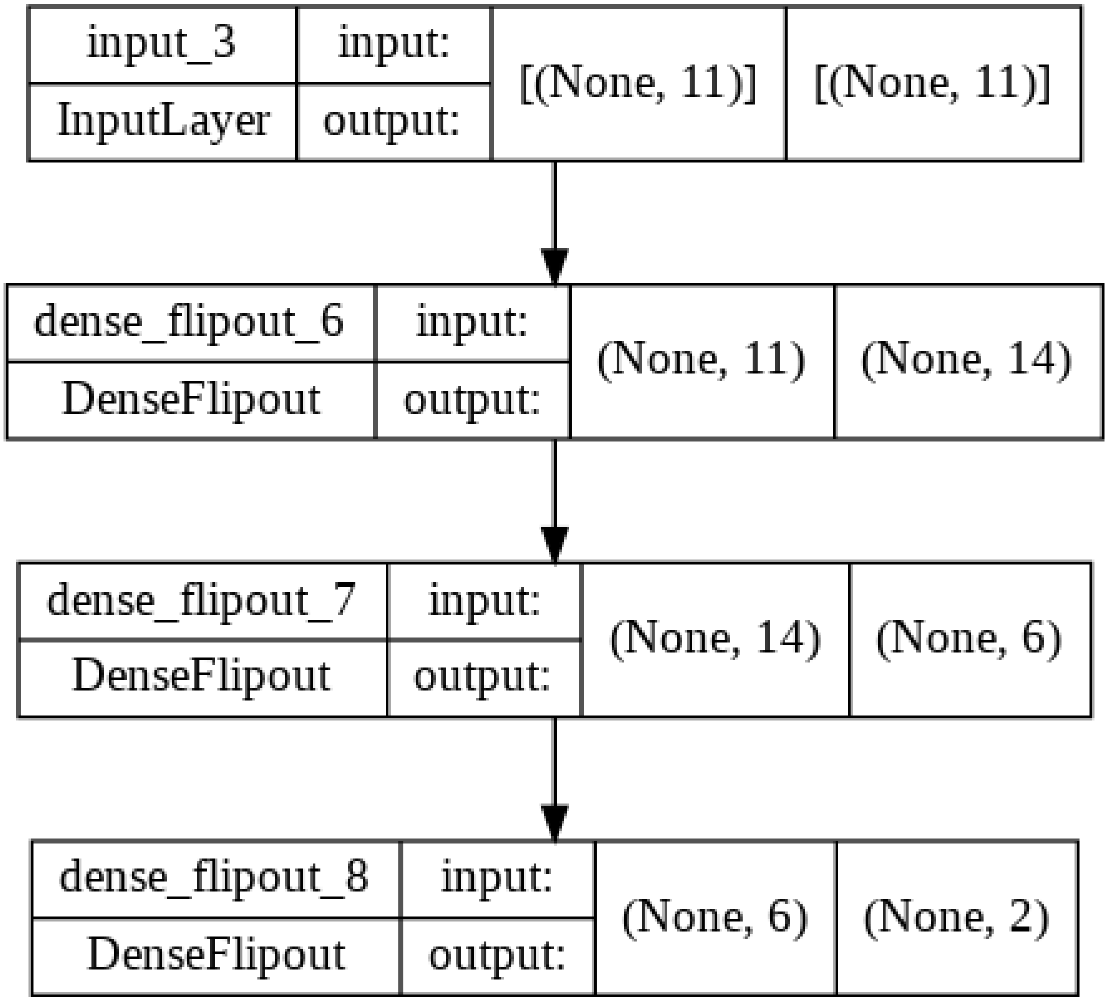



```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_6 (DenseFlipo  (None, 14)               322
 ut)

 dense_flipout_7 (DenseFlipo  (None, 6)                174
 ut)

 dense_flipout_8 (DenseFlipo  (None, 2)                26
 ut)

=================================================================
Total params: 522
Trainable params: 522
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
#ann_viz(model_tfp_v2, title="My Second neural network")
```

[ ]:
```
from keras.utils.vis_utils import plot_model
plot_model(model_tfp_v2, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[ ]:



visualize BNN

[ ]:
```
!pip3 install keras
!pip3 install ann_visualizer
!pip install graphviz
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages
(2.8.0)
Requirement already satisfied: ann_visualizer in /usr/local/lib/python3.7/dist-
packages (2.5)
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-
packages (0.10.1)
```

```
Experiment 3: probabilistic Bayesian neural network: not needed
```

0.0.3 DIFFERENT BNN'S

```
1. NORMAL BNN
2. BNN WITH DIFFERENT DROPOUTS
3. BNN WITH DIFFERENT EARLY STOPS
4. BNN WITH DIFFERENT REGULARIZERS
5. SIR mentioned to work on transformers also
6. MIXING OF THE ABOVE VARIANTS AND COMPARING WITH THE NORMAL ANN
```

`PLOT THE UNCERTAINITIES FOR ALL THESE MODELS`

1. NORMAL BNN

[ ]:
```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

normal_bnn_model = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(16, kernel_divergence_fn=kl_divergence_function␣
 ↪),#activation=tf.nn.relu),
    tfp.layers.DenseFlipout(6,␣
 ↪kernel_divergence_fn=kl_divergence_function,activation=tf.nn.relu),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06#0.001
normal_bnn_model.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:
```
# TRY REMOVING THE VALIDATION PART FROM THE FIT
# validation_data = (np.asarray(X_test), np.asarray(y_test))
#history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=50,validation_split=0.2, shuffle=True)
# to see history:
```

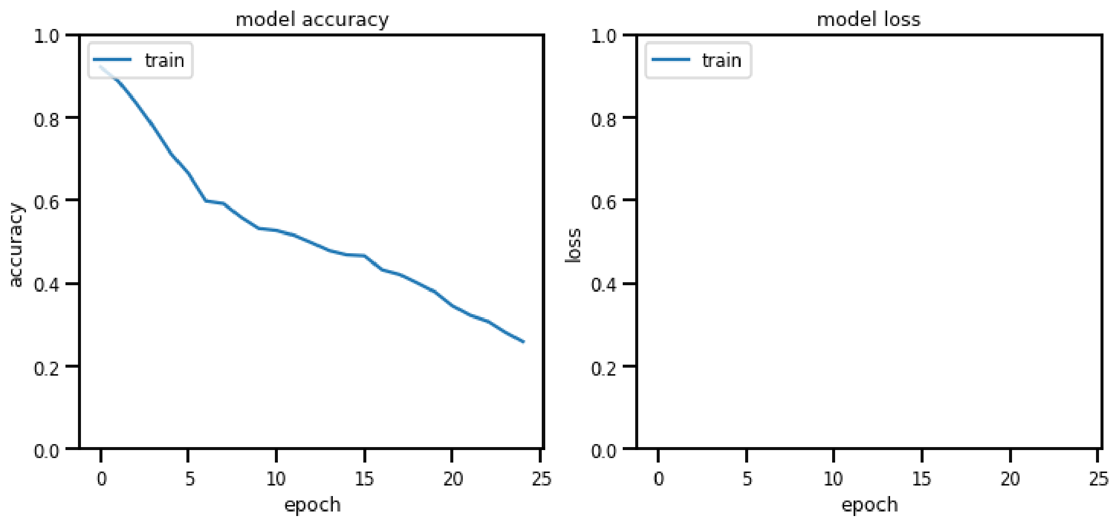
```
history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=25, batch_size=1,validation_data = (np.
 ↪asarray(X_test), np.asarray(y_test)),verbose=0)
# list all data in history
print(history.history.keys())
# summarize history for accuracy
test_loss, test_acc = normal_bnn_model.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)

normal_bnn_model.save('normal_bnn_model.h5')
normal_bnn_model.save('saved_model/normal_bnn_model')
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
94/94 [==============================] - 1s 3ms/step - loss: 0.9998 - accuracy:
0.2473

Test accuracy: 0.2473333328962326

Test loss: 0.9997942447662354
INFO:tensorflow:Assets written to: saved_model/normal_bnn_model/assets
```

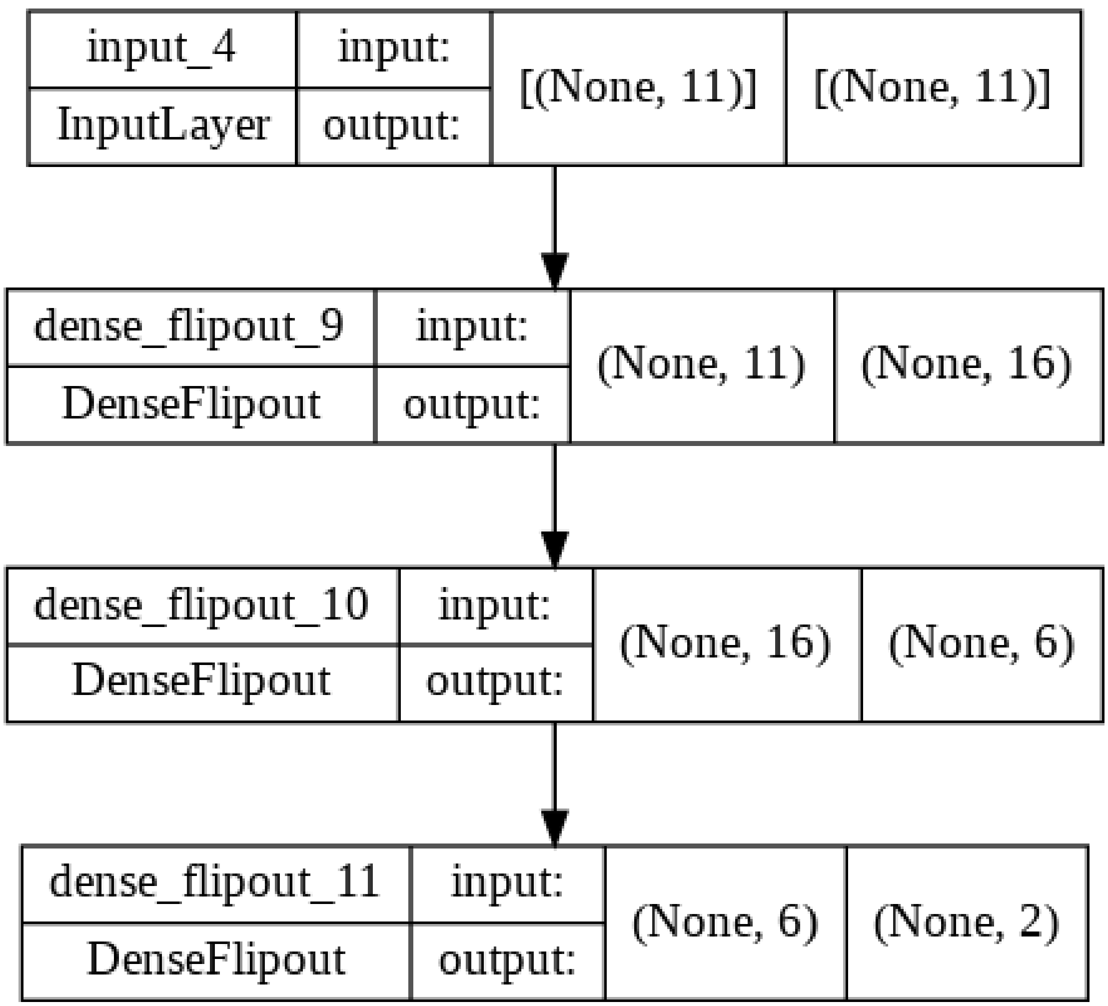
[ ]:
```
print(normal_bnn_model.predict([[2,299.1,309.5,1600,47.8,80,0,0,0,0,0]]))
```

```
[[0.36724856 0.6327514 ]]
```

[ ]:
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
plt.show()
plot_model(normal_bnn_model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```


model accuracy
train
accuracy
epoch
model loss
train
loss
epoch

[ ]:


input_4
InputLayer
input:
output:
[(None, 11)]
[(None, 11)]
dense_flipout_9
DenseFlipout
input:
output:
(None, 11)
(None, 16)
dense_flipout_10
DenseFlipout
input:
output:
(None, 16)
(None, 6)
dense_flipout_11
DenseFlipout
input:
output:
(None, 6)
(None, 2)

New Section

NORMAL BNN2

[ ]:
```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

normal_bnn2_model = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    #Dense(units = 22, activation = 'relu'),
    tfp.layers.DenseFlipout(22,␣
 ↪kernel_divergence_fn=kl_divergence_function),#activation=tf.nn.relu),
    tfp.layers.DenseFlipout(12,␣
 ↪kernel_divergence_fn=kl_divergence_function),#activation=tf.nn.relu),
    tfp.layers.DenseFlipout(4, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06 #0.00065
normal_bnn2_model.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:
```
# TRY REMOVING THE VALIDATION PART FROM THE FIT
# validation_data = (np.asarray(X_test), np.asarray(y_test))
#history = normal_bnn2_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=100,validation_split=0.3, shuffle=True)
# to see history:
history = normal_bnn2_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=15, batch_size=1,validation_data = (np.
 ↪asarray(X_test), np.asarray(y_test)),verbose=0)
# list all data in history
```

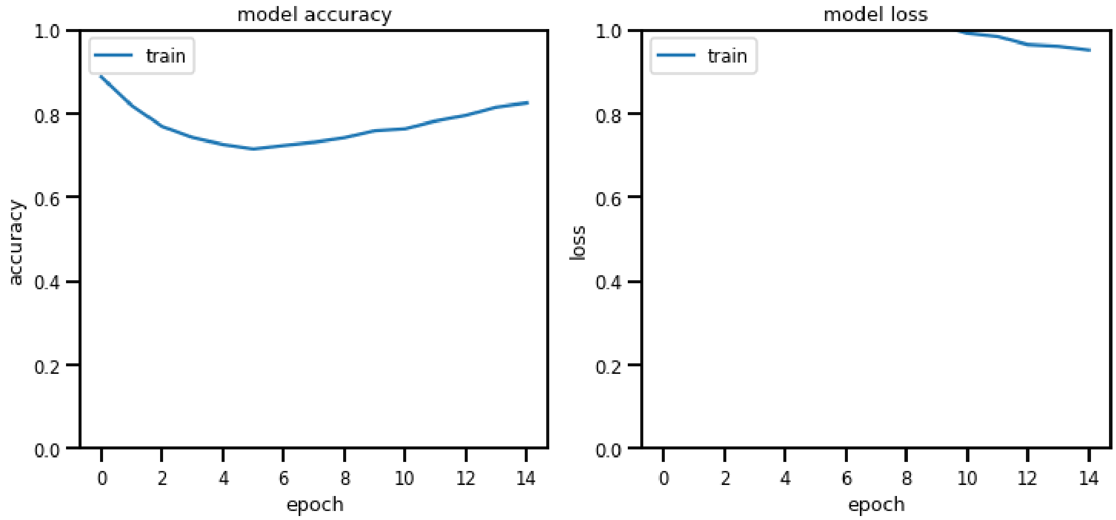
```
print(history.history.keys())
# summarize history for accuracy
test_loss, test_acc = normal_bnn2_model.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
94/94 [==============================] - 1s 3ms/step - loss: 0.9489 - accuracy:
0.8447

Test accuracy: 0.8446666598320007

Test loss: 0.948859691619873
```

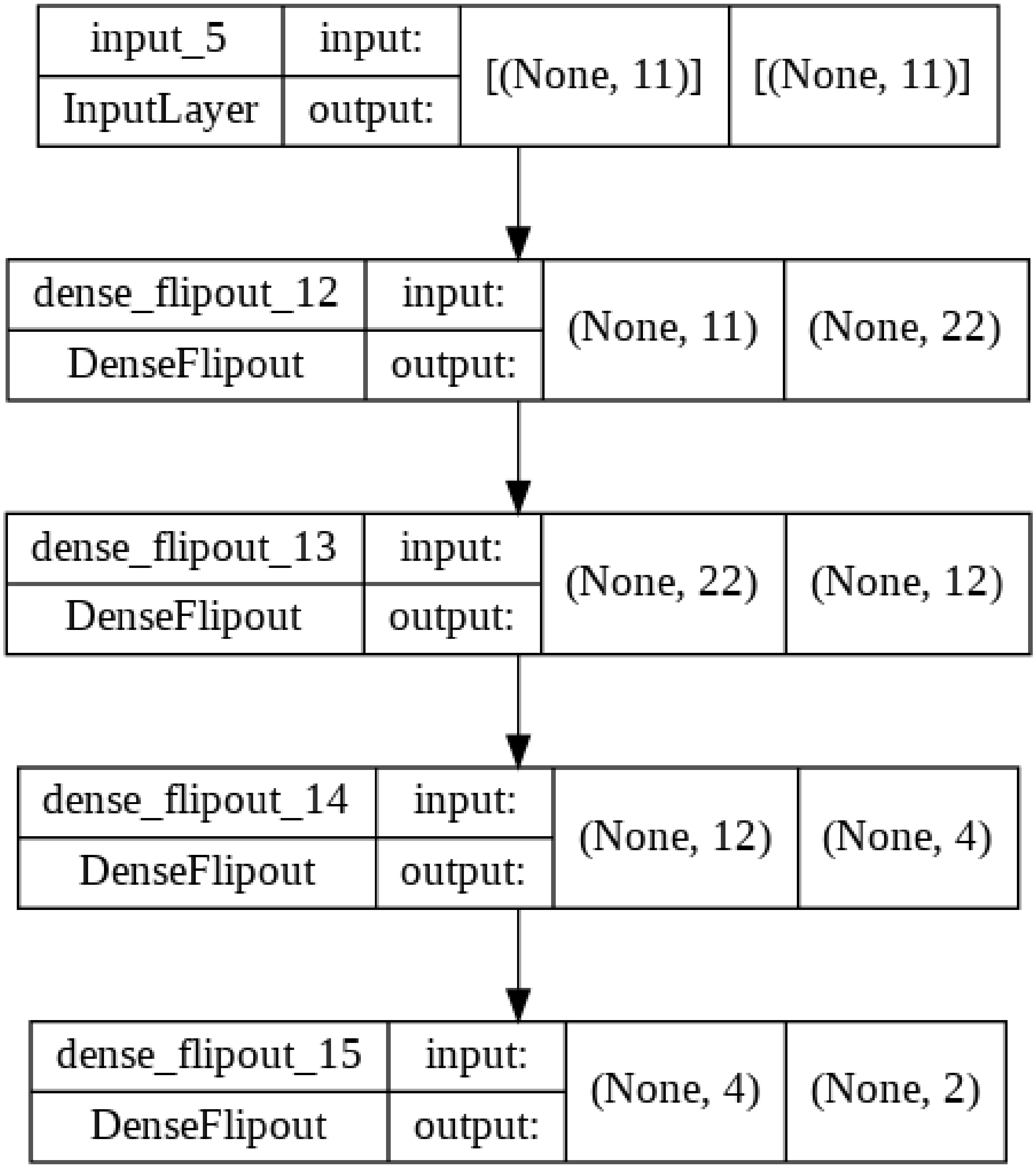
[ ]:
```
print(normal_bnn2_model.predict([[1.0,299.1,309.5,1800.0,47.8,200.0,1.0,1.0,0.
 ↪0,0.0,30.0]]))
```

```
[[0.52816063 0.47183937]]
```

[ ]:
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
# summarize history for loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1)
plt.show()
plot_model(normal_bnn2_model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```


model accuracy
train
accuracy
epoch
model loss
train
loss
epoch

[ ]:



[ ]:
```
normal_bnn2_model.save('normal_bnn2_model.h5')
normal_bnn2_model.save('saved_model/normal_bnn2_model')
```

```
INFO:tensorflow:Assets written to: saved_model/normal_bnn2_model/assets
```

2. BNN WITH DIFFERENT DROPOUT VALUES `MC Dropout write description here!`

[ ]:
```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_dropout_v1 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06 #0.005
model_dropout_v1.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:
```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_dropout_v2 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.35),
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])
```

```
learning_rate = 1e-06 #0.005
model_dropout_v2.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:

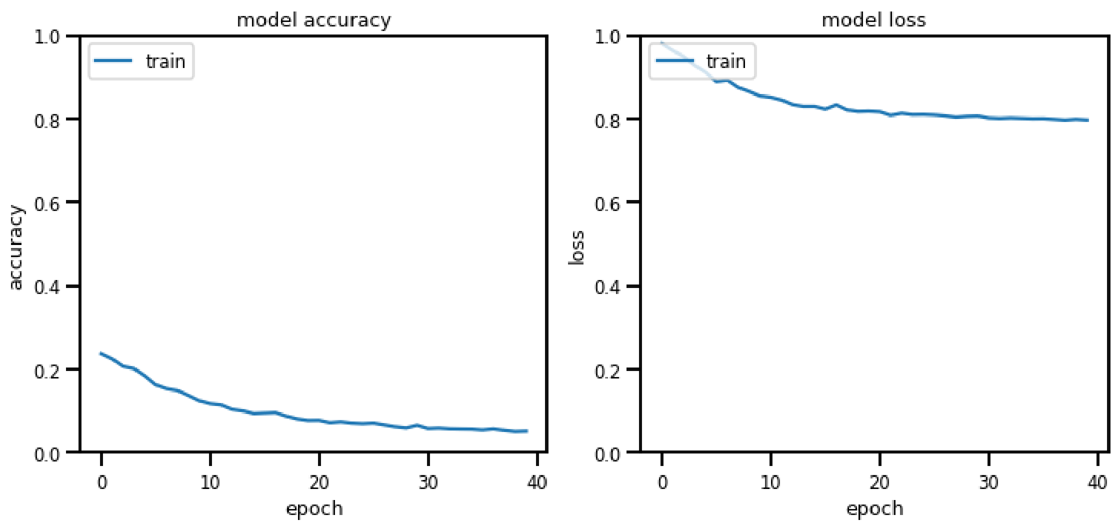
```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_dropout_v3 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.5),
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06 #0.005
model_dropout_v3.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

[ ]:
```python
from sklearn.metrics import classification_report

models = [normal_bnn_model,normal_bnn2_model,model_dropout_v1,␣
 ↪model_dropout_v2, model_dropout_v3]

models_acc = []
models_loss = []
i = 1
for p_model in models:
    #history = p_model.fit(X_train, y_train,␣
 ↪epochs=40)#,batch_size=1,validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
    history = p_model.fit(np.asarray(X_train), np.asarray(y_train),epochs=40,␣
 ↪batch_size=1,validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
    #history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=100, batch_size=1,validation_data = (np.
 ↪asarray(X_test), np.asarray(y_test)),verbose=0)
    test_loss, test_acc = p_model.evaluate(X_test, y_test)
    y_pred = p_model.predict(X_test)
    print('\nTest accuracy:', test_acc)
    print('\nTest loss:', test_loss)
    models_acc.append(test_acc)
    models_loss.append(test_loss)
    #history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=100, batch_size=1,verbose=0)
    # to see history:
    # list all data in history
    print(history.history.keys())
    p_model.save('%s.h5' %('p_model'+' '+str(i)))
    p_model.save('saved_model/%s' %('p_model'+' '+str(i)))
    i = i+1
    # summarize history for accuracy
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'])
    #plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.ylim(0, 1)
    # summarize history for loss
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    #plt.plot(history.history['val_loss'])
    plt.title('model loss')
```

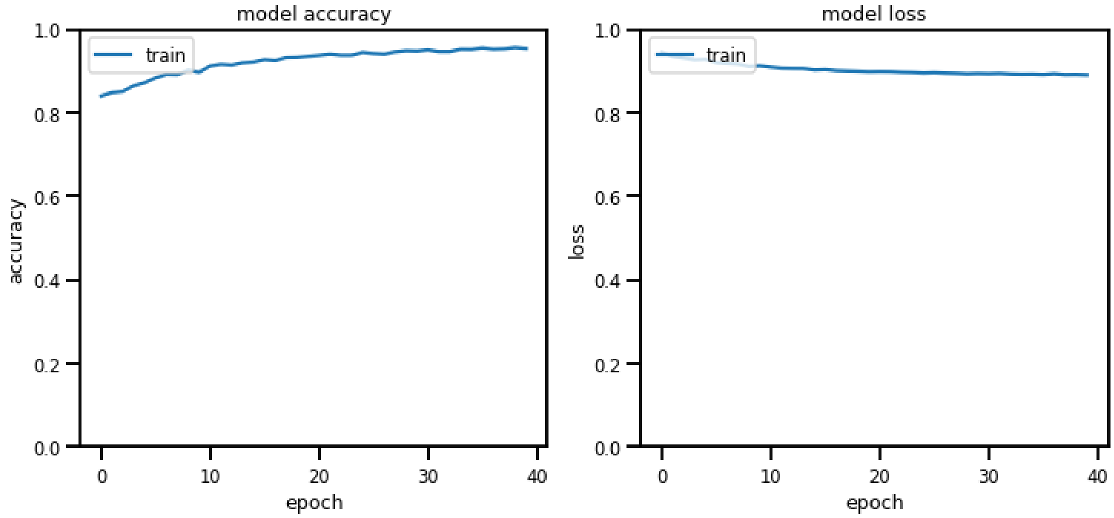
```
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.ylim(0, 1)
    plt.show()
    plot_model(p_model, to_file='model_plotss.png', show_shapes=True,␣
↪show_layer_names=True)
    '''index = 0
    for i in y_pred:
        if i<0.5:
            y_pred[index] = 0
        else:
            y_pred[index] = 1

    print(classification_report(y_test, y_pred))'''
```

```
94/94 [==============================] - 0s 2ms/step - loss: 0.7981 - accuracy:
0.0460

Test accuracy: 0.04600000008940697

Test loss: 0.7980939149856567
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 1/assets
```

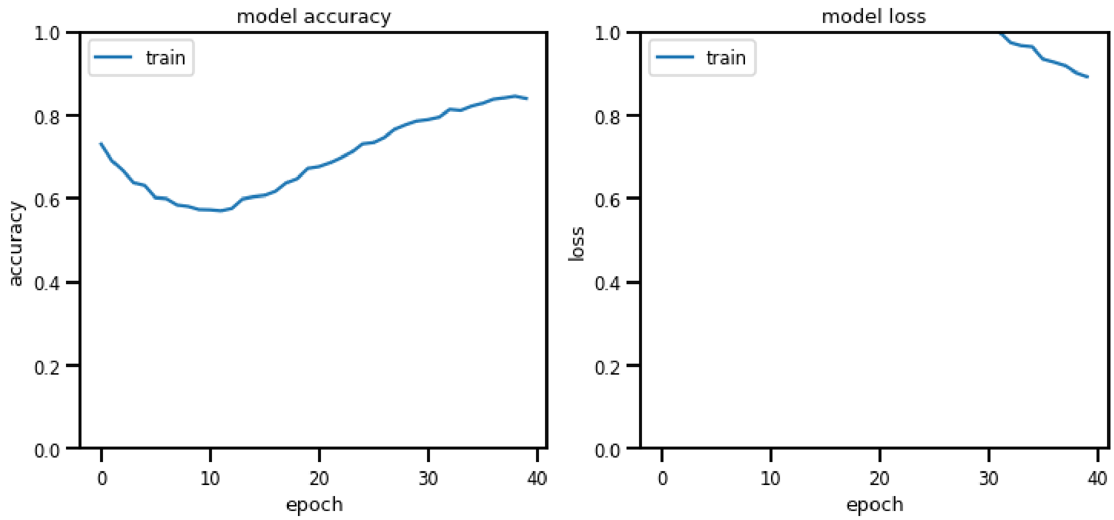



```
94/94 [==============================] - 0s 3ms/step - loss: 0.8895 - accuracy:
0.9600

Test accuracy: 0.9599999785423279
```

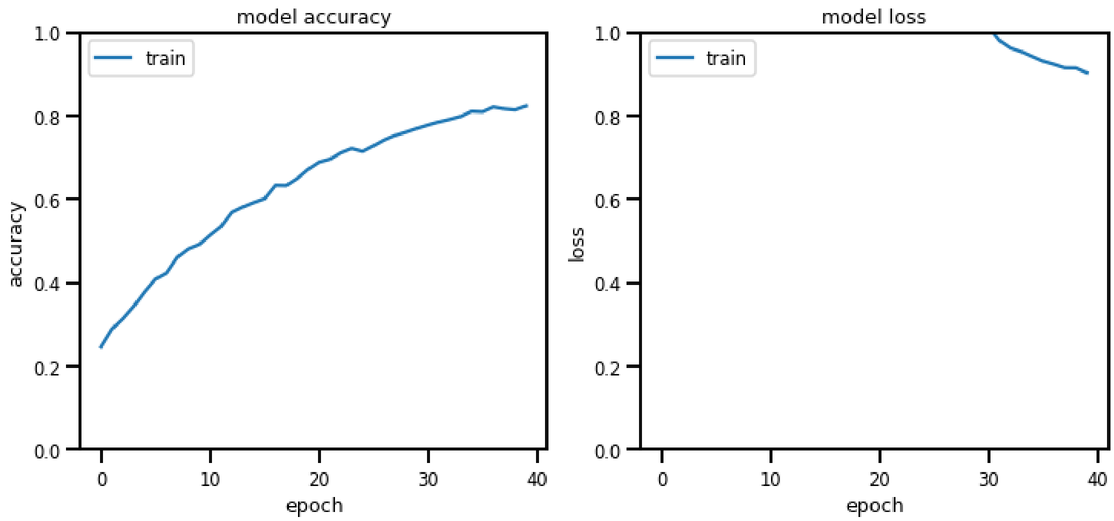
```
Test loss: 0.8895038366317749
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 2/assets
```



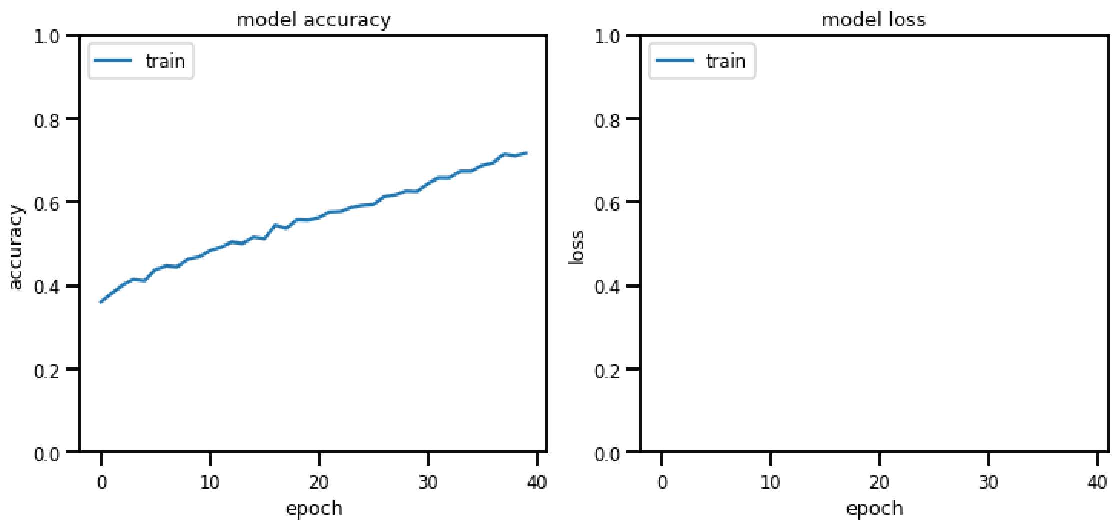

```
94/94 [==============================] - 1s 2ms/step - loss: 0.7902 - accuracy:
0.9357

Test accuracy: 0.9356666803359985

Test loss: 0.7901726961135864
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 3/assets
```



```
94/94 [==============================] - 1s 3ms/step - loss: 0.7885 - accuracy:
0.9517

Test accuracy: 0.9516666531562805

Test loss: 0.7884607911109924
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 4/assets
```




```
94/94 [==============================] - 1s 2ms/step - loss: 0.8642 - accuracy:
0.8343

Test accuracy: 0.8343333601951599

Test loss: 0.8642163276672363
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 5/assets
```




```
[ ]: print(models_acc)
     print(models_loss)
```

[0.04600000008940697, 0.9599999785423279, 0.9356666803359985, 0.9516666531562805, 0.8343333601951599]
[0.7980939149856567, 0.8895038366317749, 0.7901726961135864, 0.7884607911109924, 0.8642163276672363]

3. BNN WITH DIFFERENT EARLY STOPS

```
[104]: callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

#callbacks=[callback]
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_callback_v1 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    #tf.keras.layers.Dropout(0.5)
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])
```

```python
learning_rate = 1e-06 #0.005
model_callback_v1.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
history = model_callback_v1.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=50, batch_size=1,␣
 ↪callbacks=[callback],validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
len(history.history['loss'])

#model_tfp_v2.fit(X_train, y_train, epochs=80)
test_loss, test_acc = model_callback_v1.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)

model_callback_v1.summary()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

```
94/94 [==============================] - 1s 3ms/step - loss: 0.7877 - accuracy:
0.0573

Test accuracy: 0.0573333315551281

Test loss: 0.7876830101013184
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_28 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_29 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_30 (DenseFlip  (None, 2)                26
 out)

=================================================================
Total params: 522
```

```
Trainable params: 522
Non-trainable params: 0
_________________________________________________________________
```

[105]:
```python
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=4)

#callbacks=[callback]
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_callback_v2 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    #tf.keras.layers.Dropout(0.5)
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06 #0.005
model_callback_v2.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
history = model_callback_v2.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=10, batch_size=1,␣
 ↪callbacks=[callback],validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
len(history.history['loss'])

test_loss, test_acc = model_callback_v2.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
model_callback_v2.summary()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

```
94/94 [==============================] - 1s 3ms/step - loss: 0.9962 - accuracy:
0.3607

Test accuracy: 0.36066666245460510

Test loss: 0.9961703419685364
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_31 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_32 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_33 (DenseFlip  (None, 2)                26
 out)

=================================================================
Total params: 522
Trainable params: 522
Non-trainable params: 0
_________________________________________________________________
```

[106]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)

#callbacks=[callback]
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

model_callback_v3 = tf.keras.Sequential([
    tf.keras.Input(X_train.shape[1]),
    tfp.layers.DenseFlipout(14, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu),
    #tf.keras.layers.Dropout(0.5)
    tfp.layers.DenseFlipout(6, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.relu ),
    tfp.layers.DenseFlipout(2, kernel_divergence_fn=kl_divergence_function,␣
 ↪activation=tf.nn.softmax),
])

learning_rate = 1e-06 #0.005
model_callback_v3.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate),loss='binary_crossentropy',metrics=['accuracy'])
```

```
history = model_callback_v3.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=10, batch_size=1,␣
 ↪callbacks=[callback],validation_data = (np.asarray(X_test), np.
 ↪asarray(y_test)),verbose=0)
len(history.history['loss'])

test_loss, test_acc = model_callback_v3.evaluate(X_test,  y_test)
print('\nTest accuracy:', test_acc)
print('\nTest loss:', test_loss)
model_callback_v3.summary()
```

```
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:102: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
/usr/local/lib/python3.7/dist-
packages/tensorflow_probability/python/layers/util.py:112: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  trainable=trainable)
```

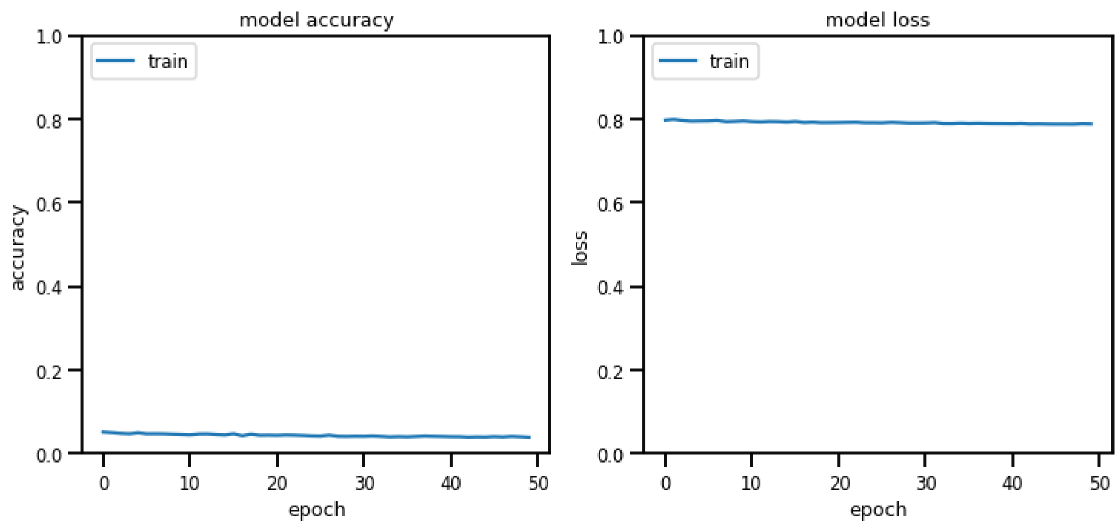
```
94/94 [==============================] - 1s 3ms/step - loss: 2.2261 - accuracy:
0.5263

Test accuracy: 0.5263333320617676

Test loss: 2.2260825634002686
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_34 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_35 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_36 (DenseFlip  (None, 2)                26
 out)

=================================================================
Total params: 522
Trainable params: 522
Non-trainable params: 0
_________________________________________________________________
```

[107]:

```
from sklearn.metrics import classification_report

models = [normal_bnn_model,normal_bnn2_model,model_callback_v1,␣
↪model_callback_v2, model_callback_v3]

models_acc = []
models_loss = []
i = 6
for p_model in models:
    #history = p_model.fit(X_train, y_train,␣
↪epochs=40)#,batch_size=1,validation_data = (np.asarray(X_test), np.
↪asarray(y_test)),verbose=0)
    history = p_model.fit(np.asarray(X_train), np.asarray(y_train),epochs=50,␣
↪batch_size=1,validation_data = (np.asarray(X_test), np.
↪asarray(y_test)),verbose=0)
    #history = normal_bnn_model.fit(np.asarray(X_train), np.
↪asarray(y_train),epochs=100, batch_size=1,validation_data = (np.
↪asarray(X_test), np.asarray(y_test)),verbose=0)
    test_loss, test_acc = p_model.evaluate(X_test, y_test)
    y_pred = p_model.predict(X_test)
    print('\nTest accuracy:', test_acc)
    print('\nTest loss:', test_loss)
    models_acc.append(test_acc)
    models_loss.append(test_loss)
    #history = normal_bnn_model.fit(np.asarray(X_train), np.
↪asarray(y_train),epochs=100, batch_size=1,verbose=0)
    # to see history:
    # list all data in history
    print(history.history.keys())
    p_model.save('%s.h5' %('callp_model'+' '+str(i)))
    p_model.save('saved_model/%s' %('callp_model'+' '+str(i)))
    i = i+1
    # summarize history for accuracy
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'])
    #plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.ylim(0, 1)
    # summarize history for loss
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    #plt.plot(history.history['val_loss'])
    plt.title('model loss')
```

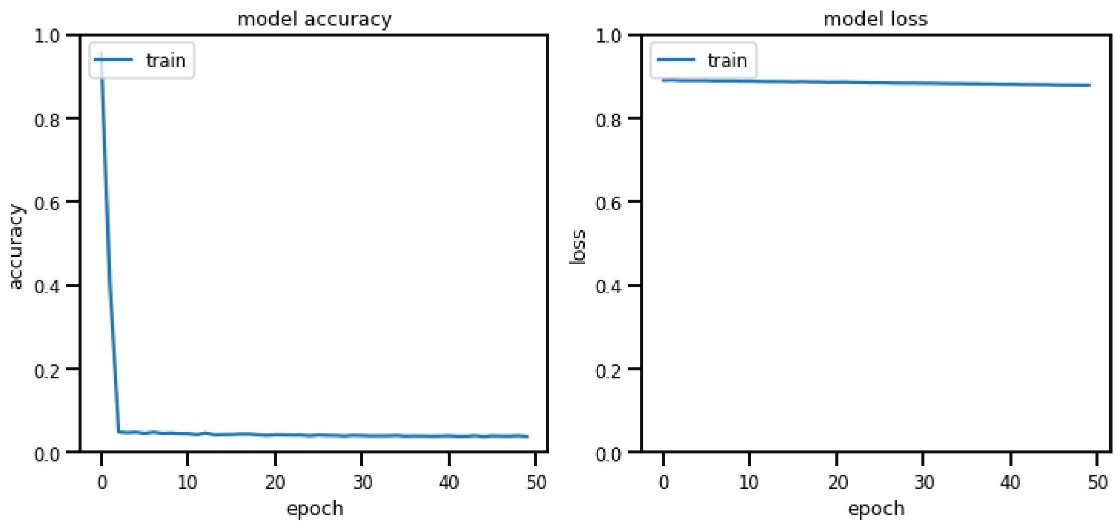
```
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.ylim(0, 1)
    plt.show()
    plot_model(p_model, to_file='model_plotsss.png', show_shapes=True,␣
↪show_layer_names=True)
    '''index = 0
    for i in y_pred:
        if i<0.5:
            y_pred[index] = 0
        else:
            y_pred[index] = 1

    print(classification_report(y_test, y_pred))'''
```

```
94/94 [==============================] - 0s 3ms/step - loss: 0.7896 - accuracy:
0.0360

Test accuracy: 0.035999998450279236

Test loss: 0.789599597454071
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 6/assets
```

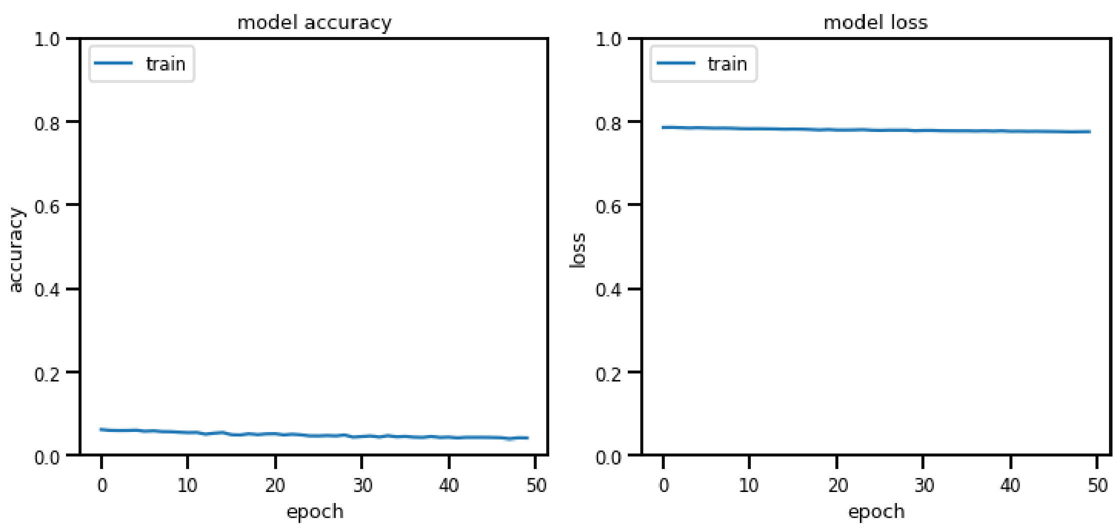



```
94/94 [==============================] - 0s 3ms/step - loss: 0.8770 - accuracy:
0.0340

Test accuracy: 0.03400000184774399
```

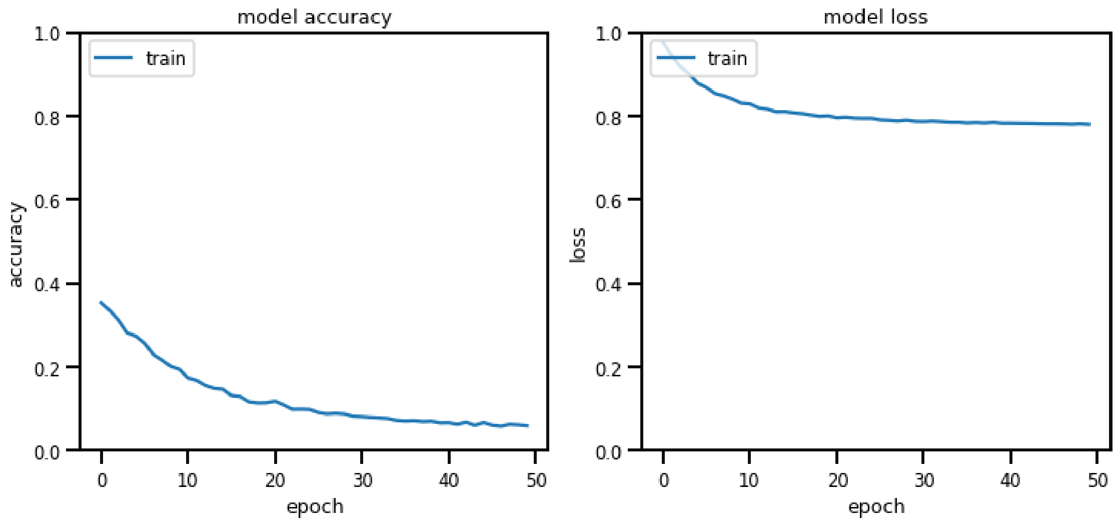
```
Test loss: 0.8769914507865906
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 7/assets
```



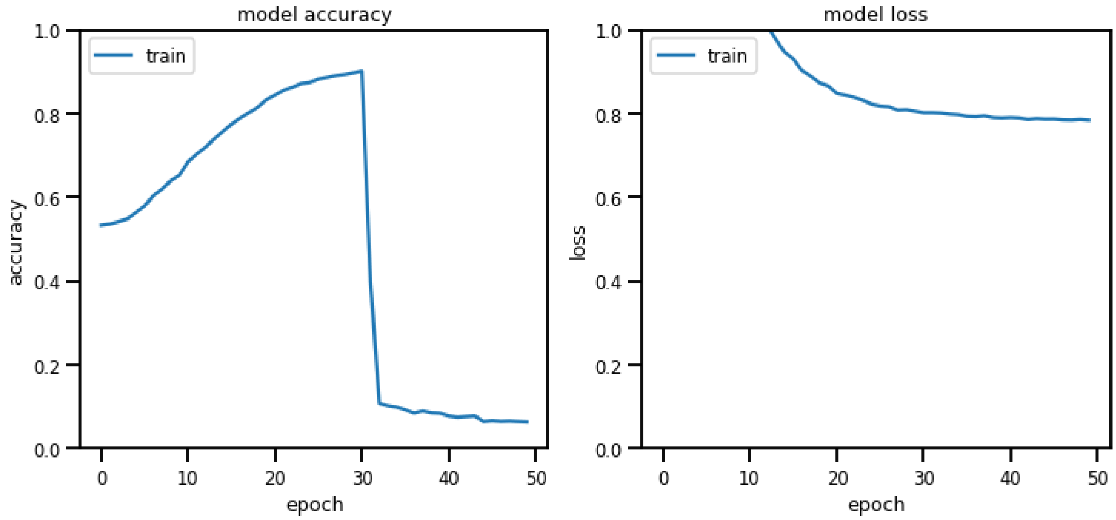

```
94/94 [==============================] - 0s 3ms/step - loss: 0.7733 - accuracy:
0.0340

Test accuracy: 0.03400000184774399

Test loss: 0.7732999920845032
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 8/assets
```



```
94/94 [==============================] - 0s 2ms/step - loss: 0.7792 - accuracy:
0.0593

Test accuracy: 0.059333331882953644

Test loss: 0.7792478799819946
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 9/assets
```




```
94/94 [==============================] - 0s 2ms/step - loss: 0.7828 - accuracy:
0.0580

Test accuracy: 0.057999998331069946

Test loss: 0.7828440070152283
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 10/assets
```



4. BNN WITH DIFFERENT REGULARIZERS `TRANSFORMERS`

6. MIXING OF THE ABOVE VARIANTS AND COMPARING WITH THE NORMAL ANN

w and b site `streamlit for gui`

[107]:

```

```

0.0.4 WEEKLY OUTPUT PDFS

convert notebook to pdf `for weekly progrss submission`

[107]:

```

```

[108]:

```
%cd /content/drive/MyDrive/Colab Notebooks/MTP
!pwd
!ls
```

```
/content/drive/MyDrive/Colab Notebooks/MTP
/content/drive/MyDrive/Colab Notebooks/MTP
 3rd_sem1.pdf                         material
 3rd_sem.pdf                          models.zip
 4th_sem_MARCH.pdf                    model_tfp1v1.pkl
 4th_sem_mid_FINAL_all.pdf            model_tfp_v1.h5
 4th_sem_mid_FINAL_alls.pdf           MTP_BNN.ipynb
 4th_sem_mid_FINAL.pdf                MTP_BNN.pdf
```

```
 4th_sem_mid.pdf                        MTP_Data_Visualization.ipynb
 4th_sem_mid_plots.pdf                 'p-2 mid'
 4th_sem_mid_plots_sir.pdf              READ.md
 4th_sem.pdf                            README.md
'Copy of 4th_sem_mid.pdf'               saved_model
'Copy of 4th_sem_mid_plots.pdf'         w1.pdf
'Copy of 4th_sem_mid_plots_sir.pdf'     w2.pdf
 datasets                              'web app'
 dec.pdf
```

[ ]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-generic-recommended
```

```
Reading package lists… Done
Building dependency tree
Reading state information… Done
The following packages were automatically installed and are no longer required:
  libnvidia-common-460 nsight-compute-2020.2.0
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-texgyre
  javascript-common libcupsfilters1 libcupsimage2 libgs9 libgs9-common
  libijs-0.35 libjbig2dec0 libjs-jquery libkpathsea6 libpotrace0 libptexenc1
  libruby2.5 libsynctex1 libtexlua52 libtexluajit2 libzzip-0-13 lmodern
  poppler-data preview-latex-style rake ruby ruby-did-you-mean ruby-minitest
  ruby-net-telnet ruby-power-assert ruby-test-unit ruby2.5
  rubygems-integration t1utils tex-common tex-gyre texlive-base
  texlive-binaries texlive-latex-base texlive-latex-extra
  texlive-latex-recommended texlive-pictures texlive-plain-generic tipa
Suggested packages:
  fonts-noto apache2 | lighttpd | httpd poppler-utils ghostscript
  fonts-japanese-mincho | fonts-ipafont-mincho fonts-japanese-gothic
  | fonts-ipafont-gothic fonts-arphic-ukai fonts-arphic-uming fonts-nanum ri
  ruby-dev bundler debhelper gv | postscript-viewer perl-tk xpdf-reader
  | pdf-viewer texlive-fonts-recommended-doc texlive-latex-base-doc
  python-pygments icc-profiles libfile-which-perl
  libspreadsheet-parseexcel-perl texlive-latex-extra-doc
  texlive-latex-recommended-doc texlive-pstricks dot2tex prerex ruby-tcltk
  | libtcltk-ruby texlive-pictures-doc vprerex
The following NEW packages will be installed:
  fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-texgyre
  javascript-common libcupsfilters1 libcupsimage2 libgs9 libgs9-common
  libijs-0.35 libjbig2dec0 libjs-jquery libkpathsea6 libpotrace0 libptexenc1
  libruby2.5 libsynctex1 libtexlua52 libtexluajit2 libzzip-0-13 lmodern
  poppler-data preview-latex-style rake ruby ruby-did-you-mean ruby-minitest
  ruby-net-telnet ruby-power-assert ruby-test-unit ruby2.5
  rubygems-integration t1utils tex-common tex-gyre texlive-base
  texlive-binaries texlive-fonts-recommended texlive-generic-recommended
```

```
  texlive-latex-base texlive-latex-extra texlive-latex-recommended
  texlive-pictures texlive-plain-generic texlive-xetex tipa
0 upgraded, 47 newly installed, 0 to remove and 42 not upgraded.
Need to get 146 MB of archives.
After this operation, 460 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-droid-fallback
all 1:6.0.1r16-1.1 [1,805 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-lato all 2.0-2
[2,698 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic/main amd64 poppler-data all
0.4.8-2 [1,479 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/main amd64 tex-common all 6.09
[33.0 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-lmodern all
2.004.5-3 [4,551 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-noto-mono all
20171026-2 [75.5 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic/universe amd64 fonts-texgyre all
20160520-1 [8,761 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic/main amd64 javascript-common all
11 [6,066 B]
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcupsfilters1
amd64 1.20.2-0ubuntu3.1 [108 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcupsimage2
amd64 2.2.7-1ubuntu2.8 [18.6 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/main amd64 libijs-0.35 amd64
0.35-13 [15.5 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/main amd64 libjbig2dec0 amd64
0.13-6 [55.9 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libgs9-common
all 9.26~dfsg+0-0ubuntu0.18.04.16 [5,093 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libgs9 amd64
9.26~dfsg+0-0ubuntu0.18.04.16 [2,265 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic/main amd64 libjs-jquery all
3.2.1-1 [152 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libkpathsea6
amd64 2017.20170613.44572-8ubuntu0.1 [54.9 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic/main amd64 libpotrace0 amd64
1.14-2 [17.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libptexenc1
amd64 2017.20170613.44572-8ubuntu0.1 [34.5 kB]
Get:19 http://archive.ubuntu.com/ubuntu bionic/main amd64 rubygems-integration
all 1.11 [4,994 B]
Get:20 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 ruby2.5 amd64
2.5.1-1ubuntu1.11 [48.6 kB]
Get:21 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby amd64 1:2.5.1
[5,712 B]
Get:22 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 rake all
```

```
12.3.1-1ubuntu0.1 [44.9 kB]
Get:23 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-did-you-mean all
1.2.0-2 [9,700 B]
Get:24 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-minitest all
5.10.3-1 [38.6 kB]
Get:25 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-net-telnet all
0.1.1-2 [12.6 kB]
Get:26 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-power-assert all
0.3.0-1 [7,952 B]
Get:27 http://archive.ubuntu.com/ubuntu bionic/main amd64 ruby-test-unit all
3.2.5-1 [61.1 kB]
Get:28 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libruby2.5
amd64 2.5.1-1ubuntu1.11 [3,072 kB]
Get:29 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libsynctex1
amd64 2017.20170613.44572-8ubuntu0.1 [41.4 kB]
Get:30 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libtexlua52
amd64 2017.20170613.44572-8ubuntu0.1 [91.2 kB]
Get:31 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libtexluajit2
amd64 2017.20170613.44572-8ubuntu0.1 [230 kB]
Get:32 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libzzip-0-13
amd64 0.13.62-3.1ubuntu0.18.04.1 [26.0 kB]
Get:33 http://archive.ubuntu.com/ubuntu bionic/main amd64 lmodern all 2.004.5-3
[9,631 kB]
Get:34 http://archive.ubuntu.com/ubuntu bionic/main amd64 preview-latex-style
all 11.91-1ubuntu1 [185 kB]
Get:35 http://archive.ubuntu.com/ubuntu bionic/main amd64 t1utils amd64 1.41-2
[56.0 kB]
Get:36 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tex-gyre all
20160520-1 [4,998 kB]
Get:37 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 texlive-
binaries amd64 2017.20170613.44572-8ubuntu0.1 [8,179 kB]
Get:38 http://archive.ubuntu.com/ubuntu bionic/main amd64 texlive-base all
2017.20180305-1 [18.7 MB]
Get:39 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-fonts-
recommended all 2017.20180305-1 [5,262 kB]
Get:40 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-plain-
generic all 2017.20180305-2 [23.6 MB]
Get:41 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-generic-
recommended all 2017.20180305-1 [15.9 kB]
Get:42 http://archive.ubuntu.com/ubuntu bionic/main amd64 texlive-latex-base all
2017.20180305-1 [951 kB]
Get:43 http://archive.ubuntu.com/ubuntu bionic/main amd64 texlive-latex-
recommended all 2017.20180305-1 [14.9 MB]
Get:44 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-pictures
all 2017.20180305-1 [4,026 kB]
Get:45 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-latex-
extra all 2017.20180305-2 [10.6 MB]
Get:46 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tipa all 2:1.3-20
```

```
[2,978 kB]
Get:47 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-xetex all
2017.20180305-1 [10.7 MB]
Fetched 146 MB in 6s (25.8 MB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76,
<> line 47.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package fonts-droid-fallback.
(Reading database … 155203 files and directories currently installed.)
Preparing to unpack …/00-fonts-droid-fallback_1%3a6.0.1r16-1.1_all.deb …
Unpacking fonts-droid-fallback (1:6.0.1r16-1.1) …
Selecting previously unselected package fonts-lato.
Preparing to unpack …/01-fonts-lato_2.0-2_all.deb …
Unpacking fonts-lato (2.0-2) …
Selecting previously unselected package poppler-data.
Preparing to unpack …/02-poppler-data_0.4.8-2_all.deb …
Unpacking poppler-data (0.4.8-2) …
Selecting previously unselected package tex-common.
Preparing to unpack …/03-tex-common_6.09_all.deb …
Unpacking tex-common (6.09) …
Selecting previously unselected package fonts-lmodern.
Preparing to unpack …/04-fonts-lmodern_2.004.5-3_all.deb …
Unpacking fonts-lmodern (2.004.5-3) …
```

[ ]:
```
!jupyter nbconvert --to pdf --output "4th_sem_FINAL_may" MTP_BNN.ipynb
```

[ ]:

[ ]:

[ ]:
```
# should have saved plots as files for download
```

[ ]:
```
from google.colab import files
!zip -r /content/models.zip /content/saved_model
files.download("/content/models.zip")
!zip -r /content/content.zip /content/*.h5
files.download("/content/contenth5.zip")
!zip -r /content/content.zip /content/*.png
files.download("/content/contentpng.zip")
```

comparison of models.

```
[ ]: from bokeh.plotting import figure, output_file, show
```

```
[ ]: # normal vs dropout
'''import numpy as np
import matplotlib.pyplot as plt

train_loss = [0.4377, 0.7227, 0.7029, 0.7039, 0.7060]
train_accuracy = [0.9687, 0.9693, 0.9690 , 0.0313, 0.9687]
test_accuracy = [0.968666672706604,0.9693333506584167 ,0.968999981880188 ,0.
↪031333334743976590 ,0.968666672706604]
test_loss = [0.43769827485084534, 0.7226769924163818,0.7028810977935791, 0.
↪703898012638092, 0.7059929370880127]
labels = ['normal_bnn1','normal_bnn1', 'dropout_1', 'dropout_2', 'dropout_3' ]

plot_df = pd.DataFrame({"train_loss":train_loss,"train_accuracy":
↪train_accuracy,"test_accuracy":test_accuracy,"test_loss":test_loss})

#plot_df['train_loss'] = train_loss
#plot_df['train_accuracy'] = train_accuracy
#plot_df['test_accuracy'] = test_accuracy
#plot_df['test_loss'] = test_loss

plot_df.plot_bokeh(kind='bar',x = train_accuracy,title = "ta")
plt.bar([train_loss,train_accuracy,test_loss,test_accuracy],labels)

plt.xlabel("models")
plt.ylabel("parameters")
plt.title("normal bnn models vs dropout bnn models")
plt.show()'''
```