MTP_BNN

May 26, 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.
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
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=8b55dc28443a64c18c99111832f45e7e282c60b5068ead6ddf386de7261cc385
  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
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: tensorflow-probability in
/usr/local/lib/python3.7/dist-packages (0.16.0)
Requirement already satisfied: cloudpickle>=1.3 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-probability) (1.3.0)
Requirement already satisfied: gast>=0.3.2 in /usr/local/lib/python3.7/dist-
packages (from tensorflow-probability) (0.5.3)
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages
```

```
(from tensorflow-probability) (1.0.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from tensorflow-probability) (4.4.2)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-packages
(from tensorflow-probability) (0.1.7)
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)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-
packages (5.6.1)
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: testpath in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (0.6.0)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages
(from nbconvert) (5.0.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (5.1.1)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.7/dist-packages (from nbconvert) (1.5.0)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.7/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (2.11.3)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (4.10.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (2.6.1)
Requirement already satisfied: nbformat>=4.4 in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (5.4.0)
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: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-
packages (from nbformat>=4.4->nbconvert) (4.3.3)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.7/dist-
packages (from nbformat>=4.4->nbconvert) (2.15.3)
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-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: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from
```

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

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

0.0.1 DATA

import data

[ ]:
```python
#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
```

[ ]:
```python
# 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
```

[ ]:
```
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
```

[ ]:
```
# 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)
```

[ ]:
```
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
```

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

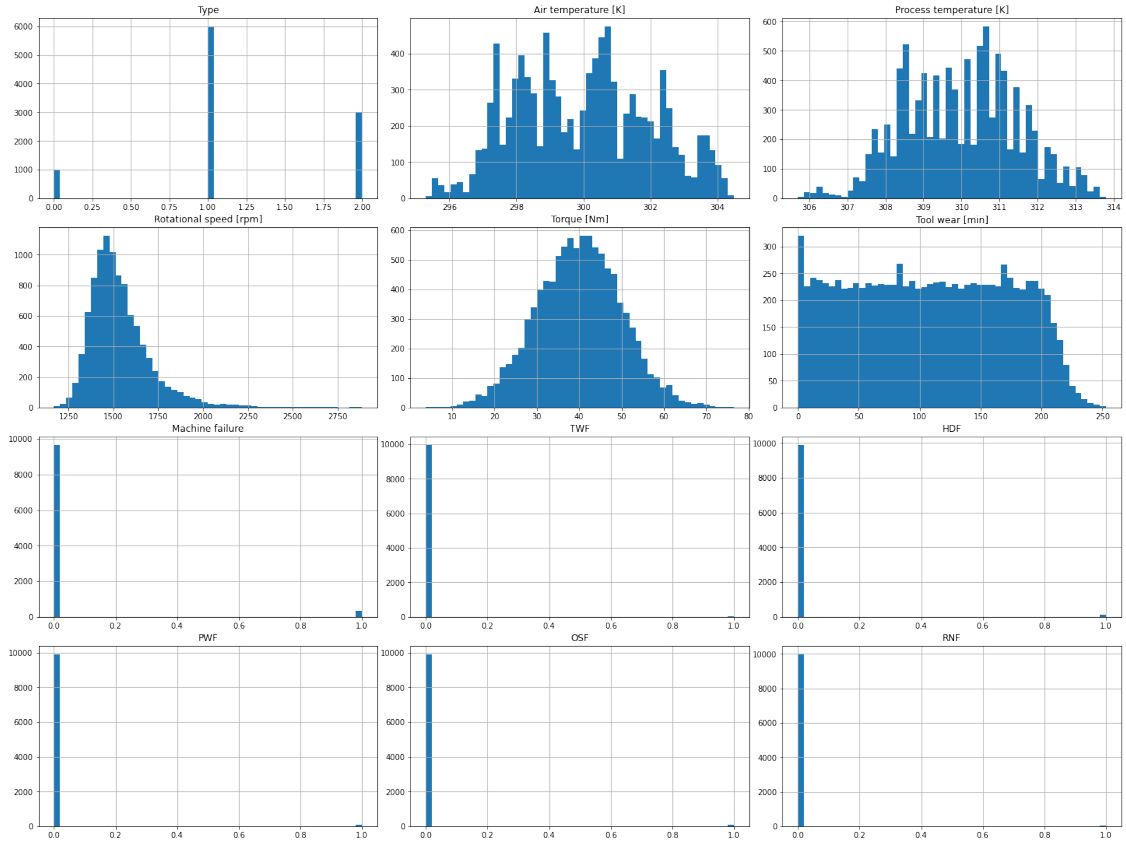
mitosheet visualization code

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

[ ]:
```
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>
```

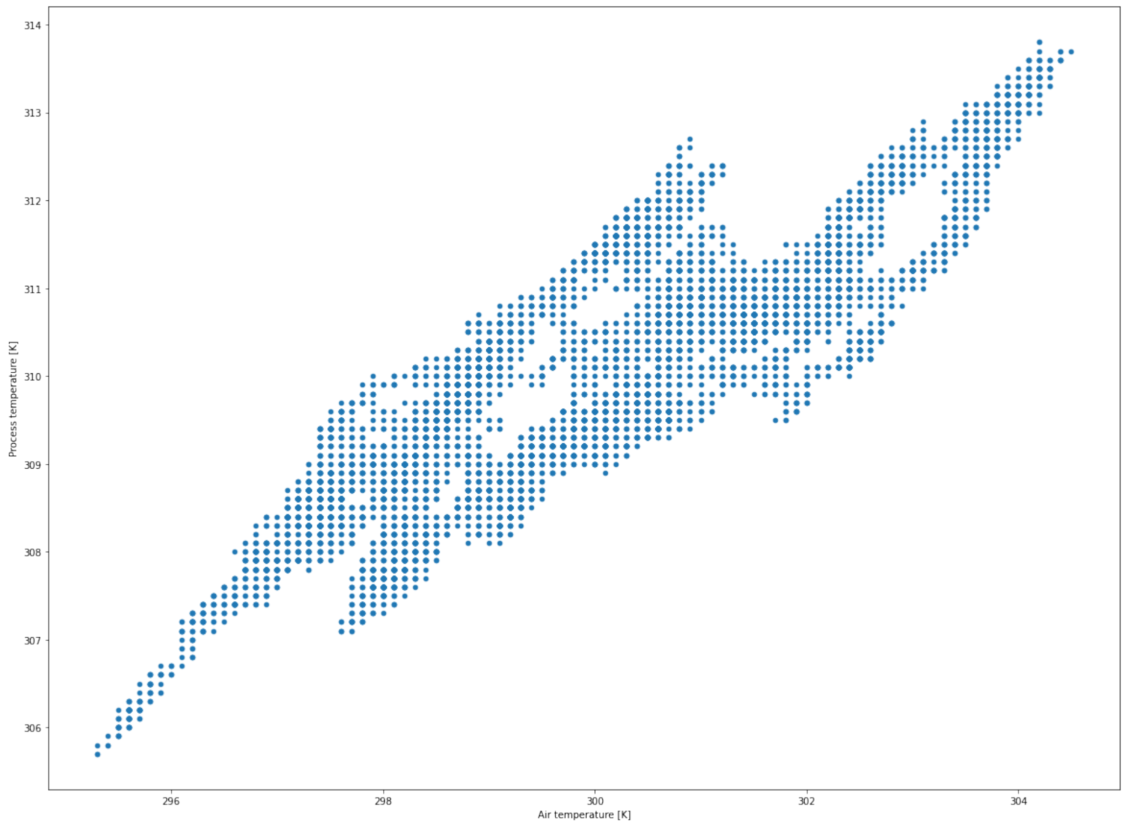


[ ]:
```
df.plot.scatter(y = 'Type',x='Air temperature [K]', figsize=(20,15))
plt.show()
```

<IPython.core.display.Javascript object>

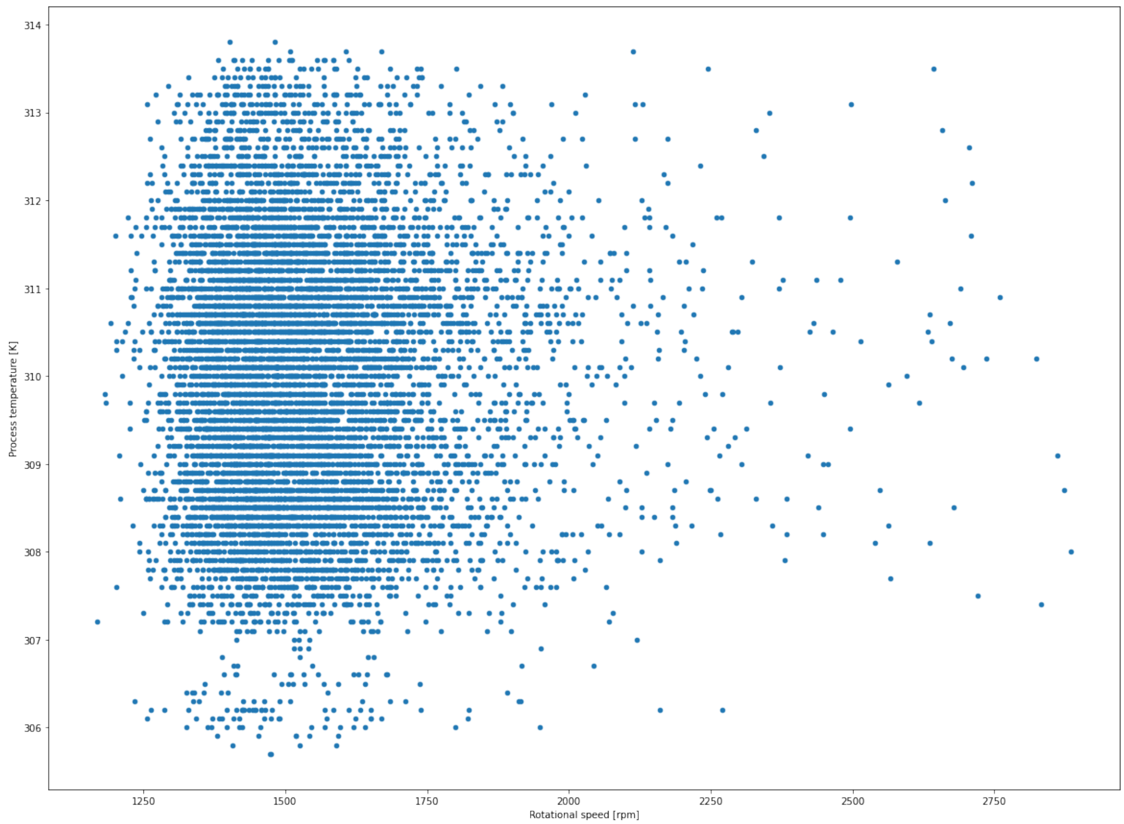
[ ]:
```
df.plot.scatter(y = 'Process temperature [K]',x='Air temperature [K]',␣
↪figsize=(20,15))
plt.show()
```

```
<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

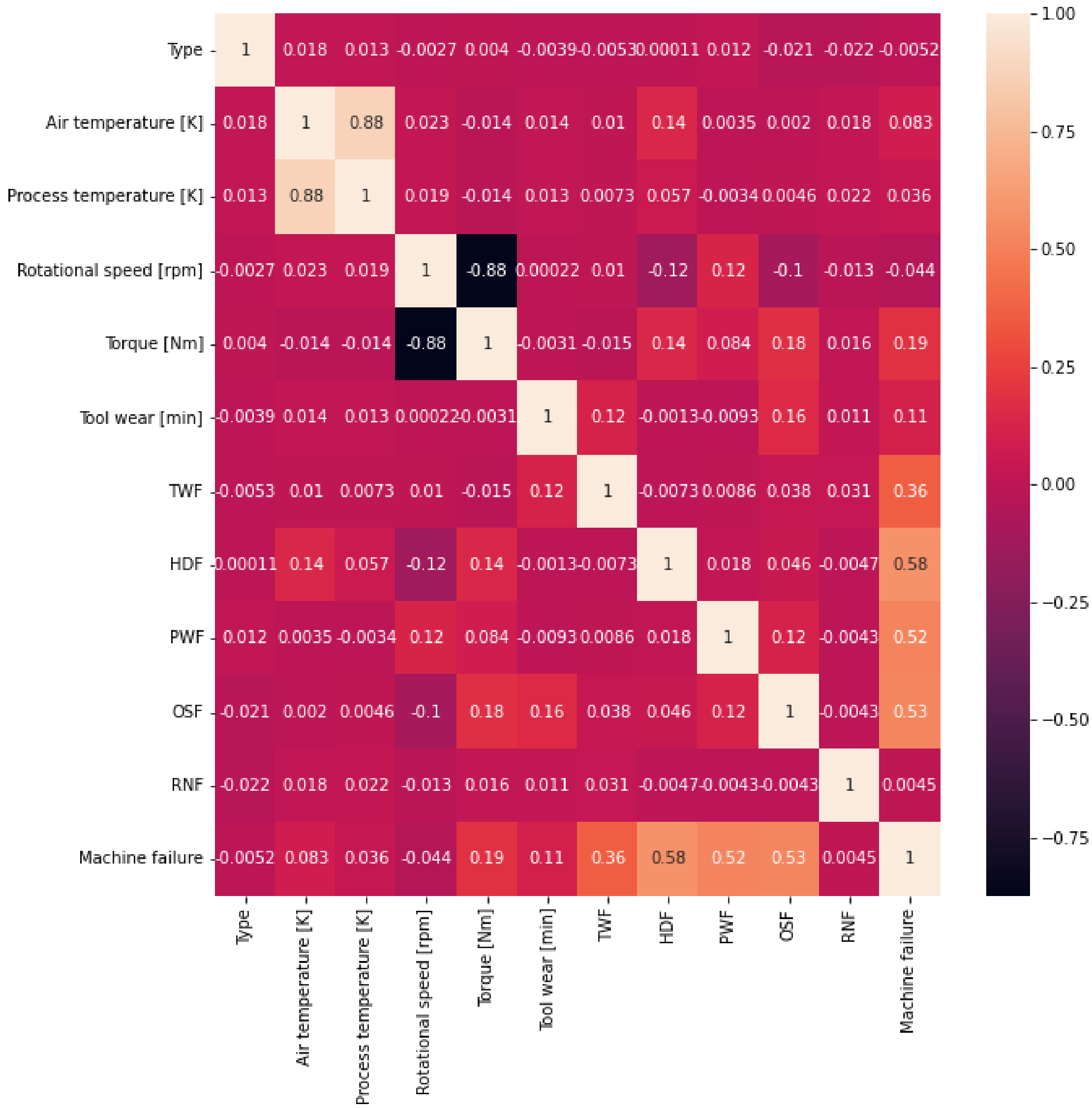
	Torque [Nm]	Tool wear [min]	TWF	HDF \
Type	0.004011	-0.003930	-0.005349	0.000108
Air temperature [K]	-0.013778	0.013853	0.009955	0.137831
Process temperature [K]	-0.014061	0.013488	0.007315	0.056933
Rotational speed [rpm]	-0.875027	0.000223	0.010389	-0.121241
Torque [Nm]	1.000000	-0.003093	-0.014662	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

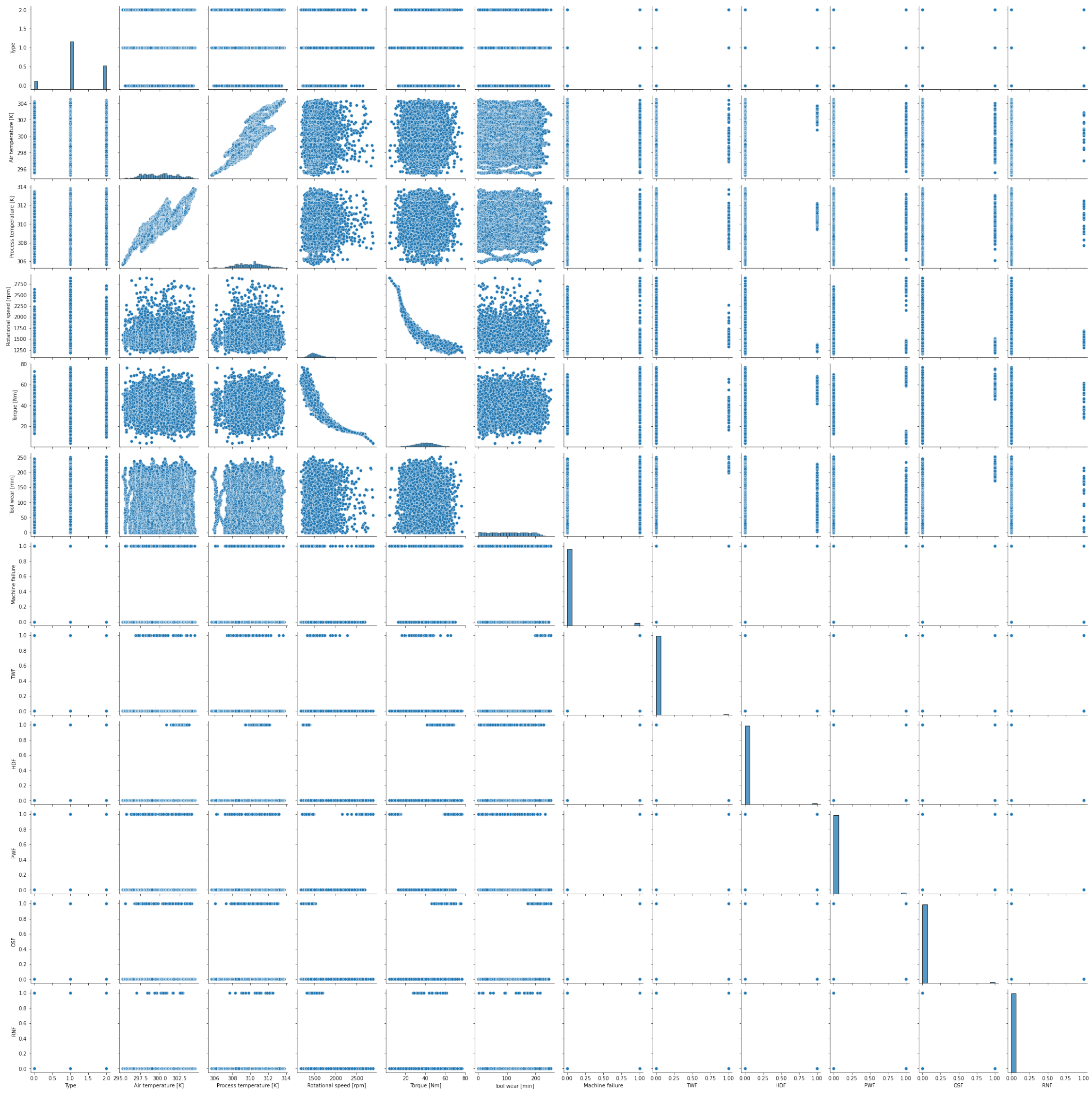
```
Machine failure          0.522812  0.531083  0.004516          1.000000
```

<IPython.core.display.Javascript object>

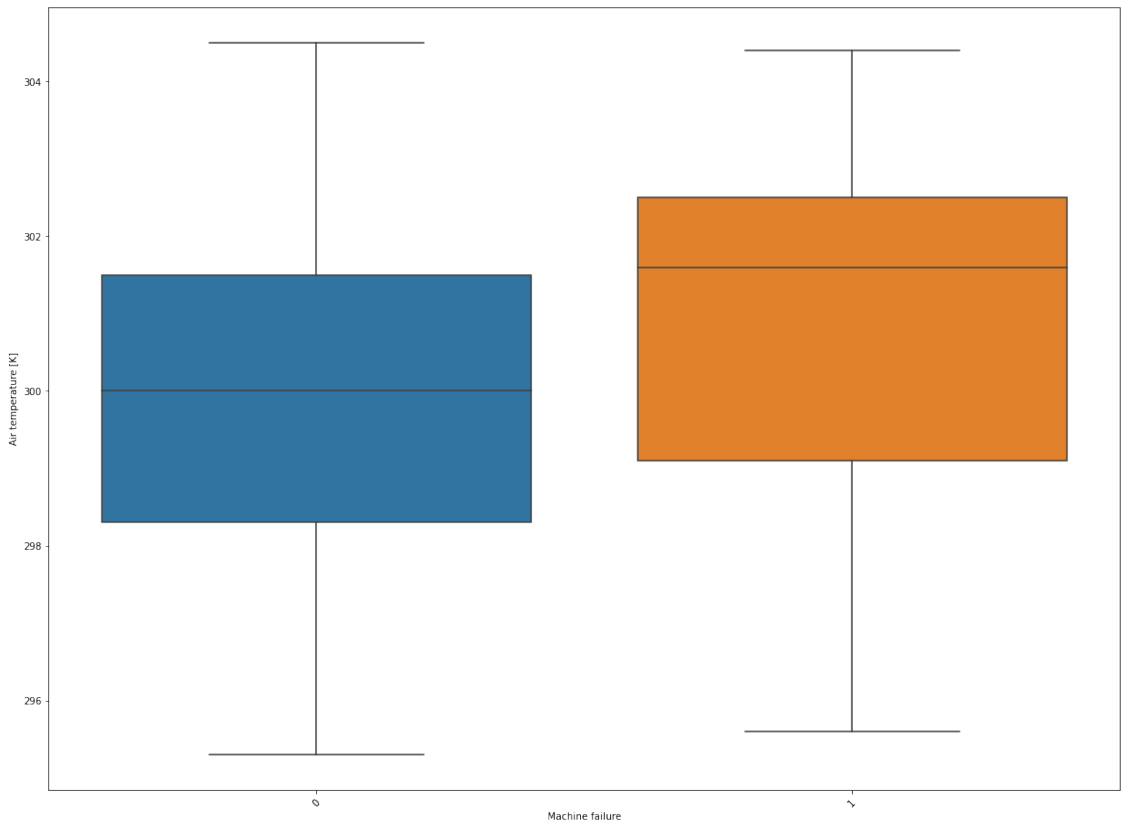
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9e2fd5b290>



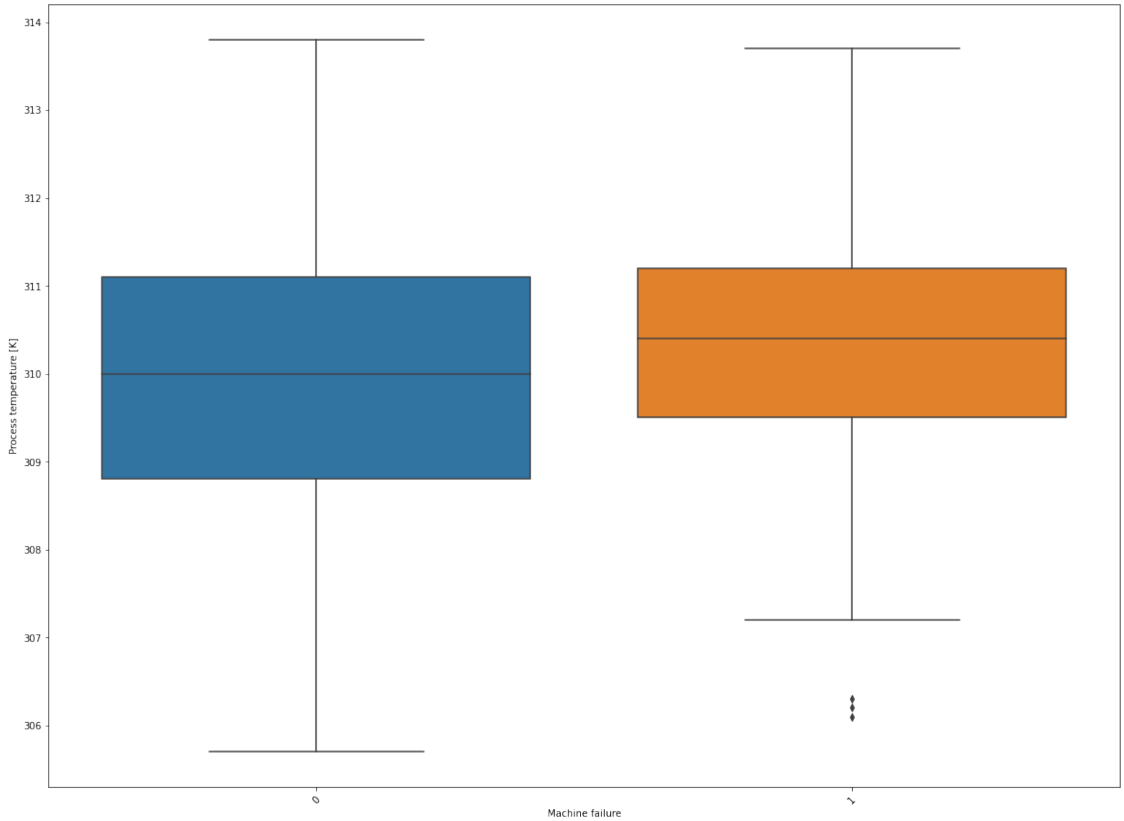

```python
import matplotlib.pyplot as plt
import seaborn as sns
sns.pairplot(df, kind="scatter")
plt.show()
```



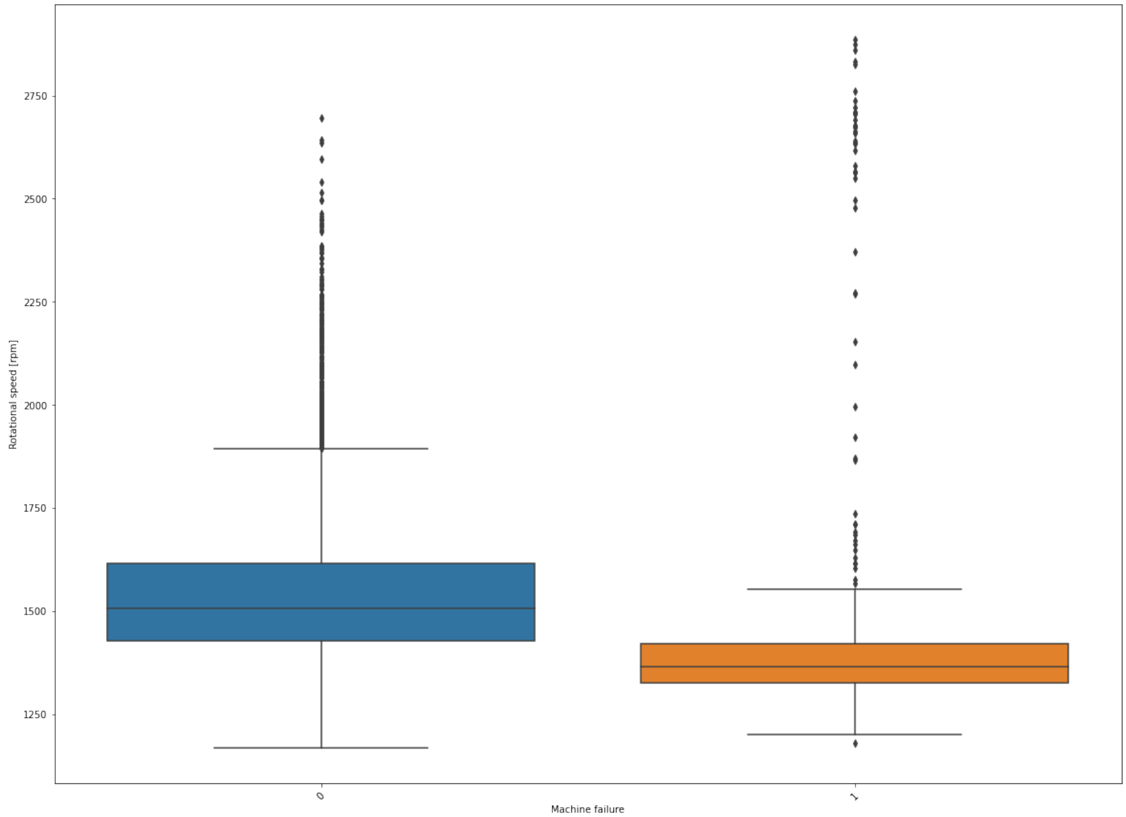
```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Air temperature [K]', x = 'Machine failure');
```



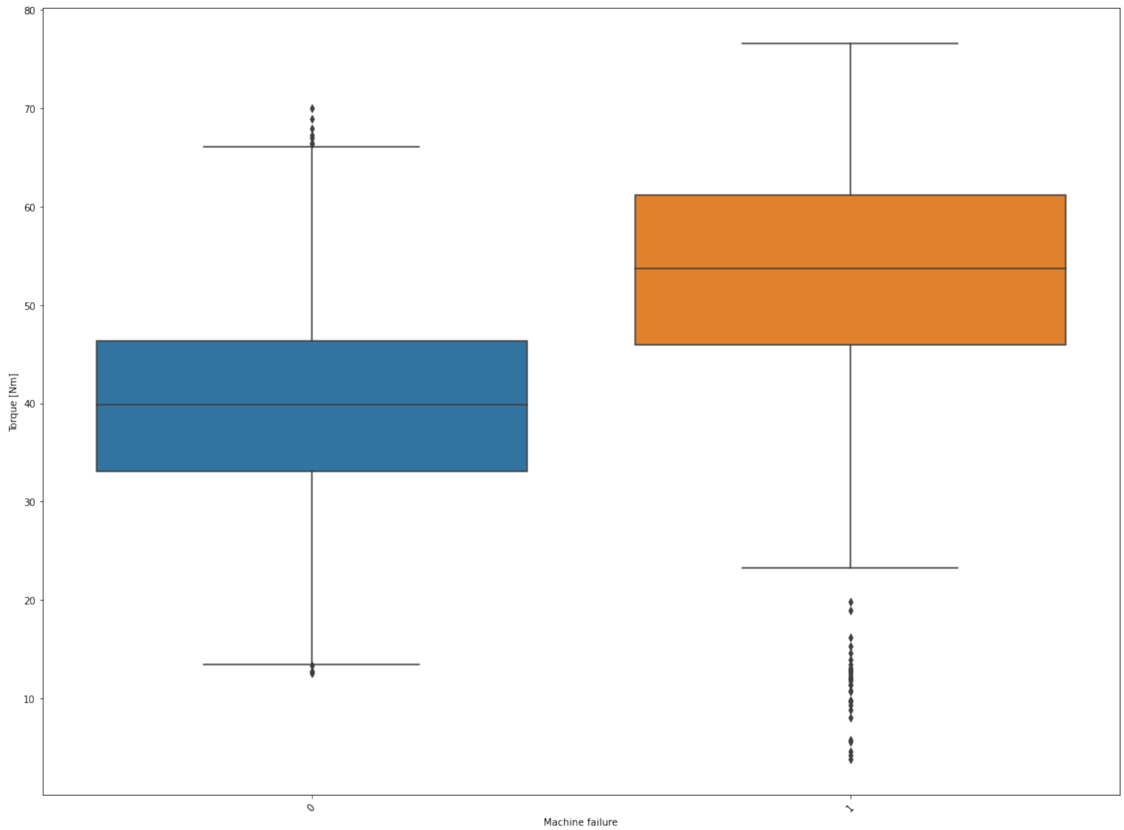

```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Process temperature [K]', x = 'Machine failure');
```



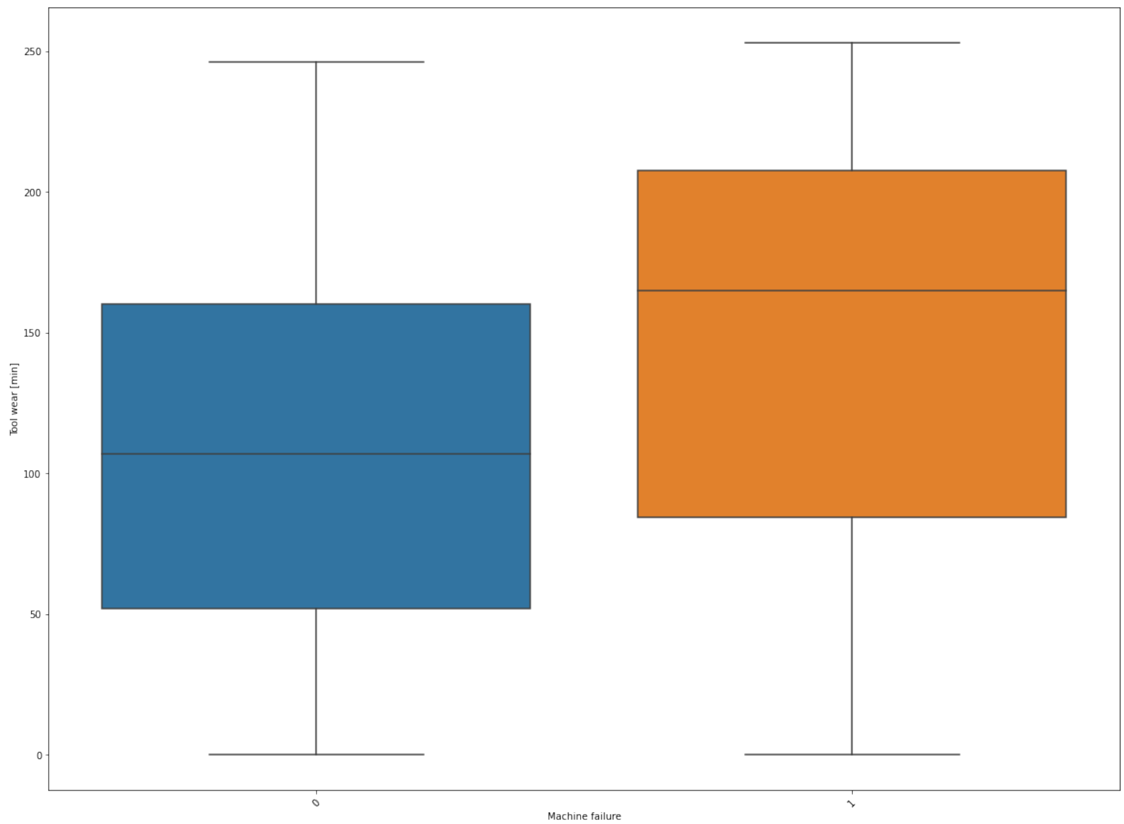

```
[ ]: plt.figure(figsize=(20,15))
     plt.xticks(rotation=45)
     sns.boxplot(data = df, y = 'Rotational speed [rpm]', x = 'Machine failure');
```



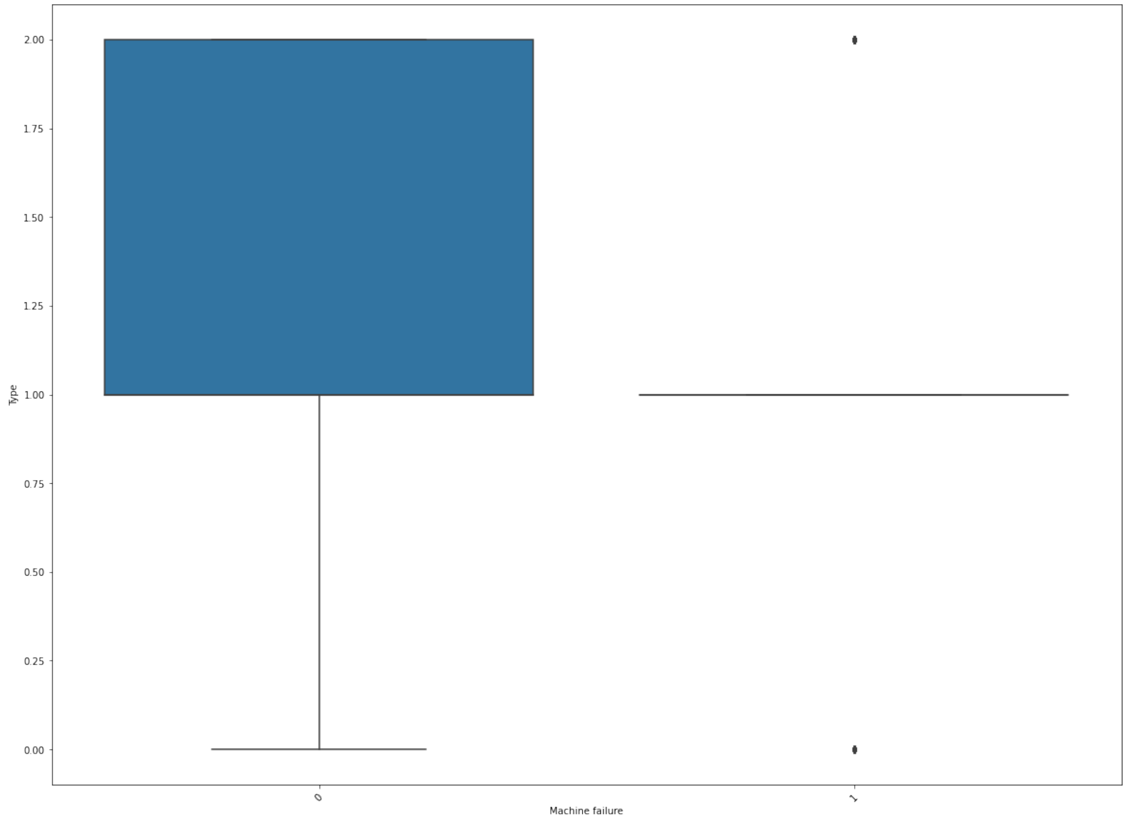
[ ]:
```
plt.figure(figsize=(20,15))
plt.xticks(rotation=45)
sns.boxplot(data = df, y = 'Torque [Nm]', x = 'Machine failure');
```



[ ]:
```
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');
```



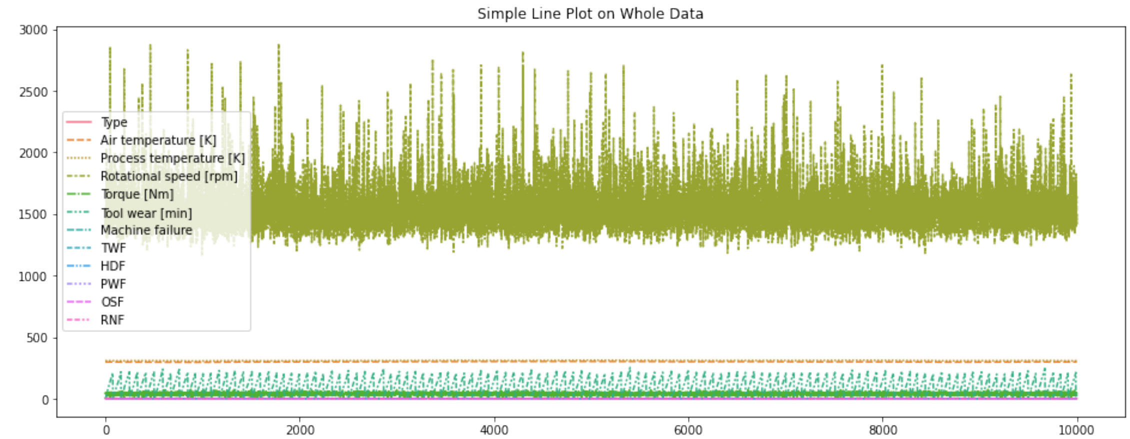
0.0.2 BNN

```
[ ]: #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

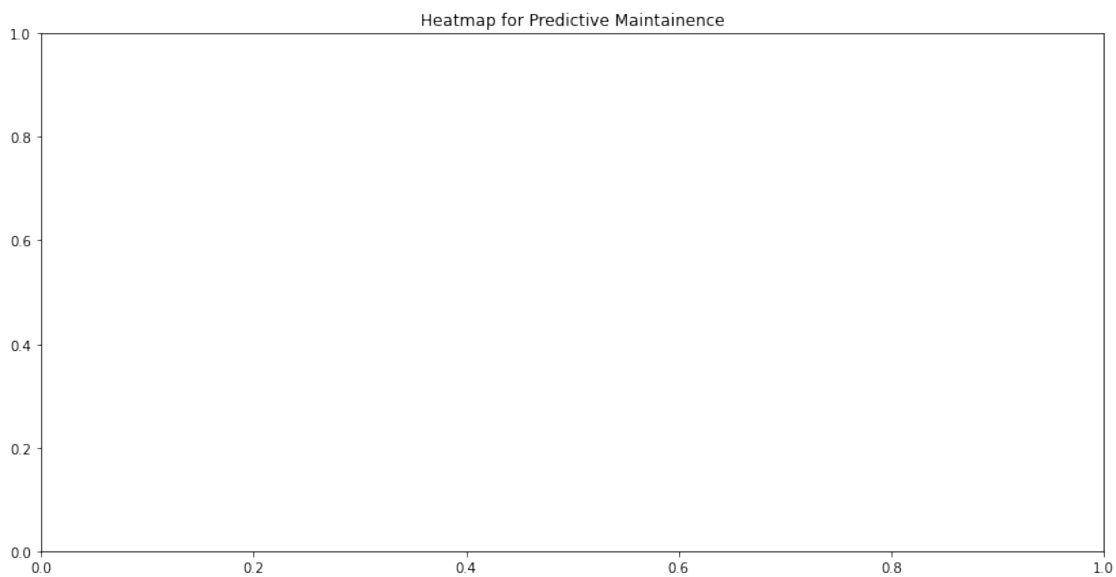
```
[ ]: import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

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

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9e27419b90>



[ ]:
```
#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")
```

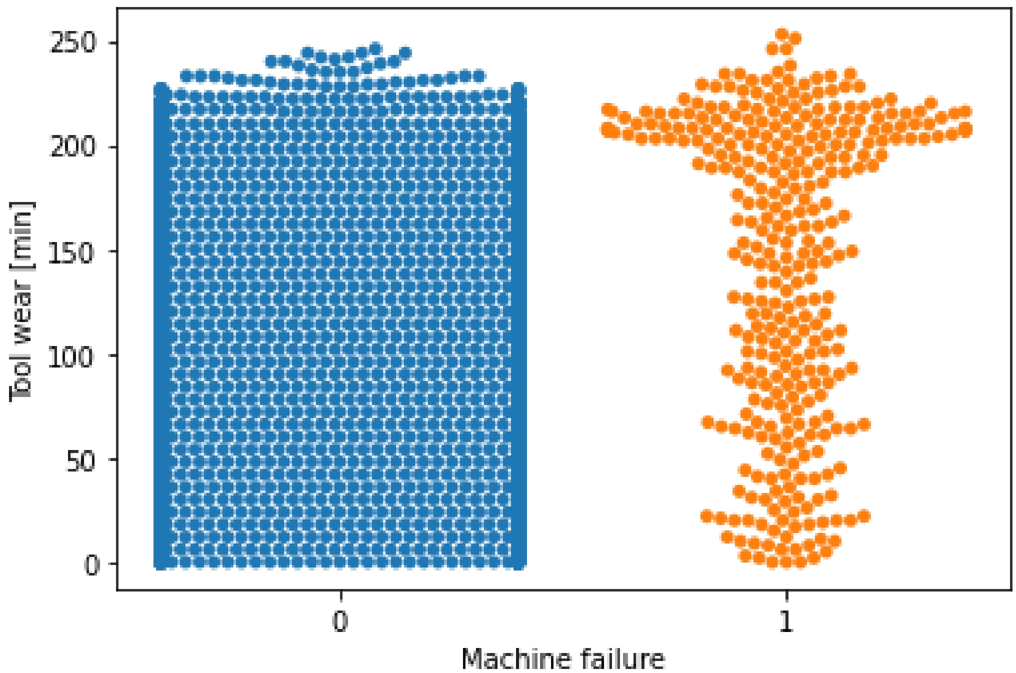
[ ]: Text(0.5, 1.0, 'Heatmap for Predictive Maintainence')



[ ]:
```
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)
```

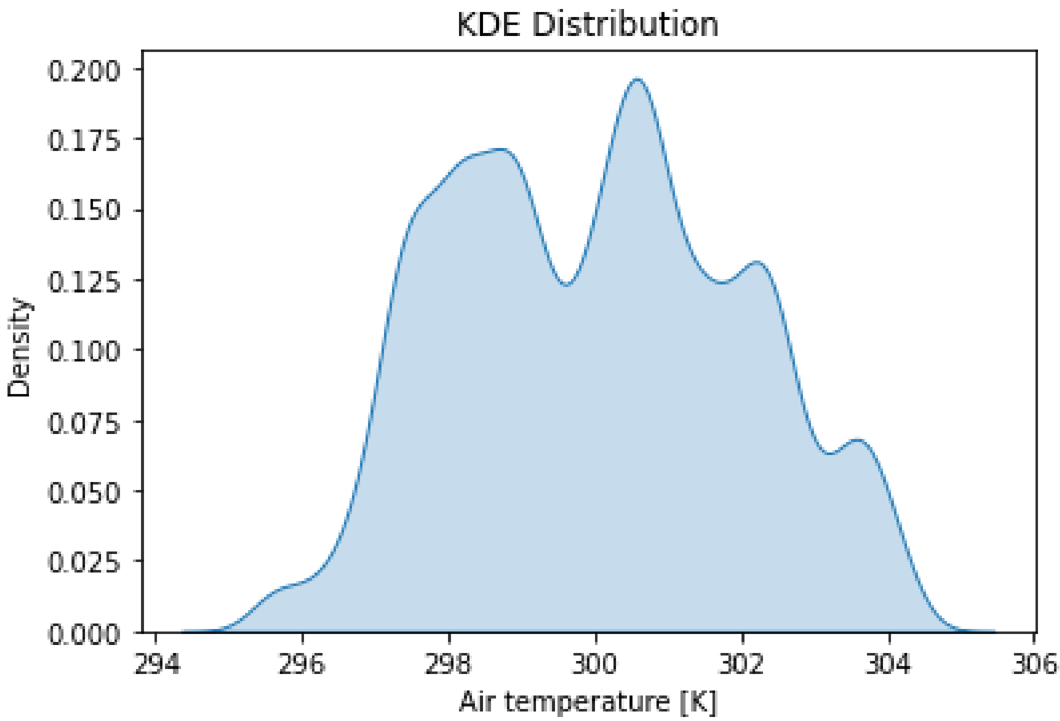
[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9dc0b369d0>
```



[ ]:
```
#stripplot
```

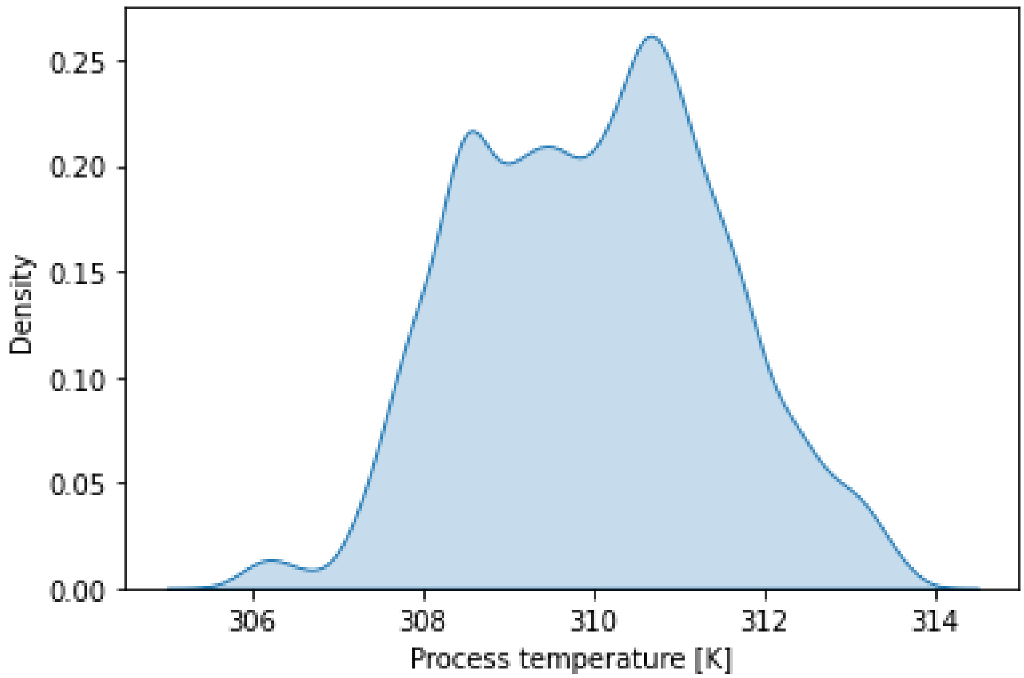
[ ]:
```
#distribution
#for i in df:
sns.kdeplot(data=df['Air temperature [K]'], label='Air temperature [K]',␣
↪shade=True)

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

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



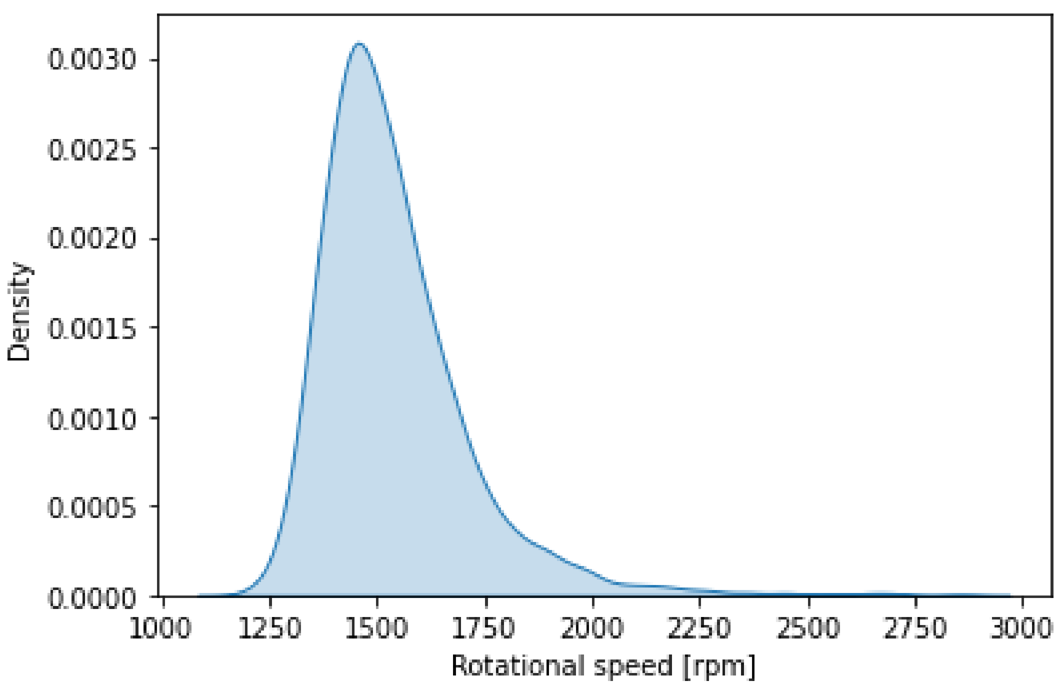

```
[ ]: sns.kdeplot(data=df['Process temperature [K]'], label='Process temperature␣
      ↪[K]', shade=True)
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc0a3d4d0>
```



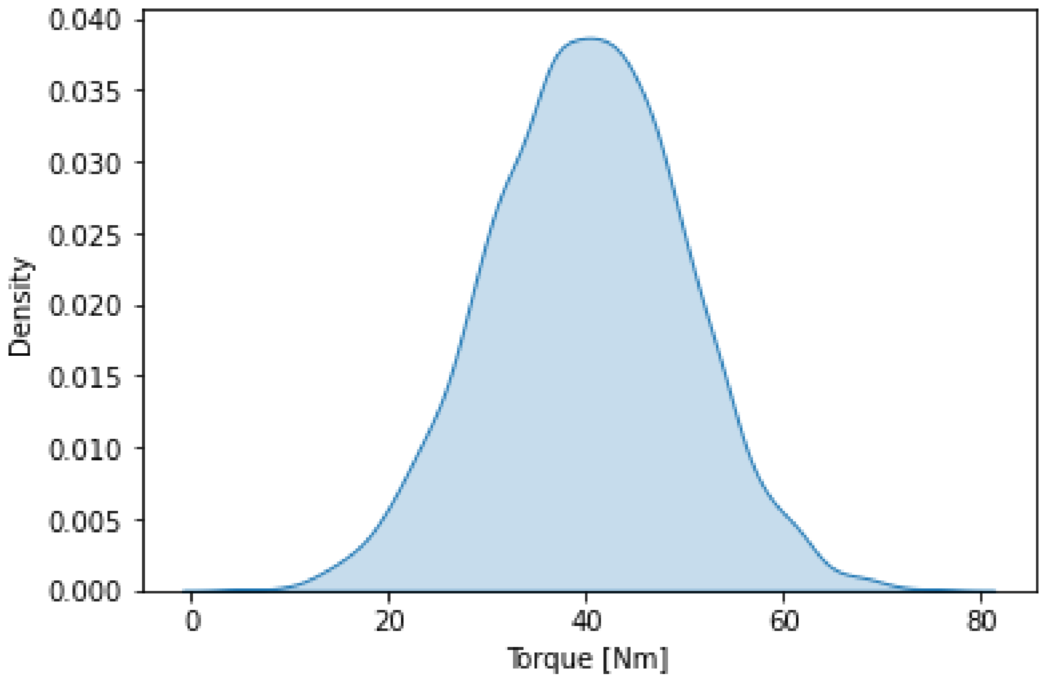

```
[ ]: sns.kdeplot(data=df['Rotational speed [rpm]'], label='Rotational speed [rpm]',␣
     ↪shade=True)
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc0a456d0>
```



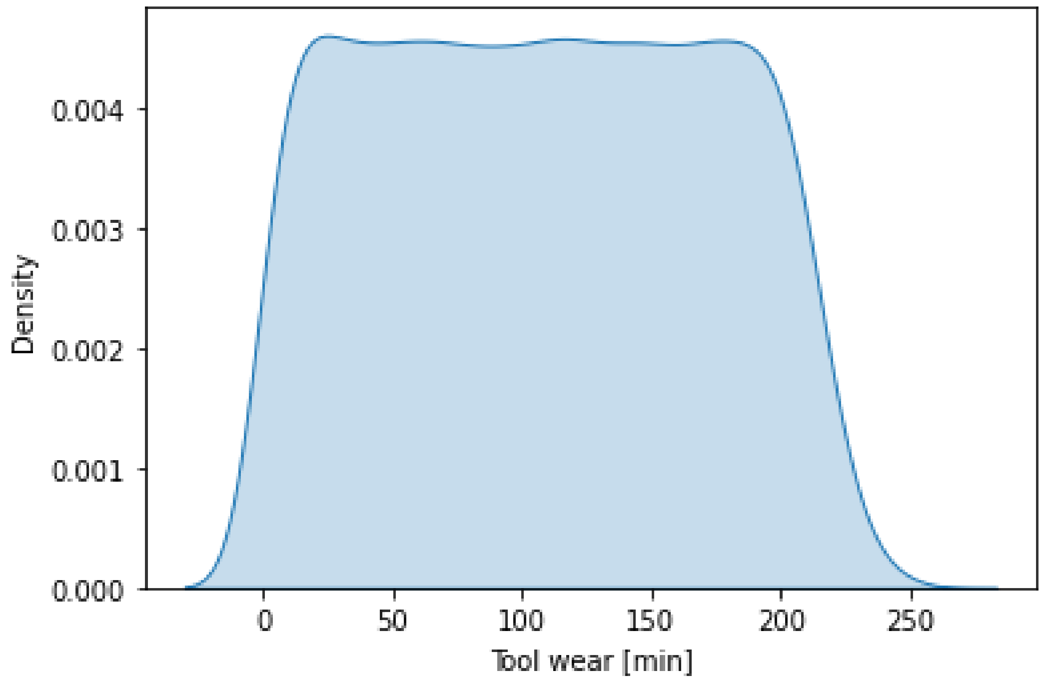
[ ]:
```
sns.kdeplot(data=df['Torque [Nm]'], label='Torque [Nm]', shade=True)
```

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc0996bd0>



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

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc09db1d0>



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)
```

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

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

[ ]:
```
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
```

[ ]:
```
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: 11.4100 -
accuracy: 0.8780
Epoch 2/50
219/219 [==============================] - 0s 2ms/step - loss: 4.2059 -
accuracy: 0.9361
Epoch 3/50
219/219 [==============================] - 0s 2ms/step - loss: 3.3513 -
accuracy: 0.9380
Epoch 4/50
219/219 [==============================] - 0s 2ms/step - loss: 2.4733 -
accuracy: 0.9393
Epoch 5/50
219/219 [==============================] - 0s 2ms/step - loss: 1.9160 -
```

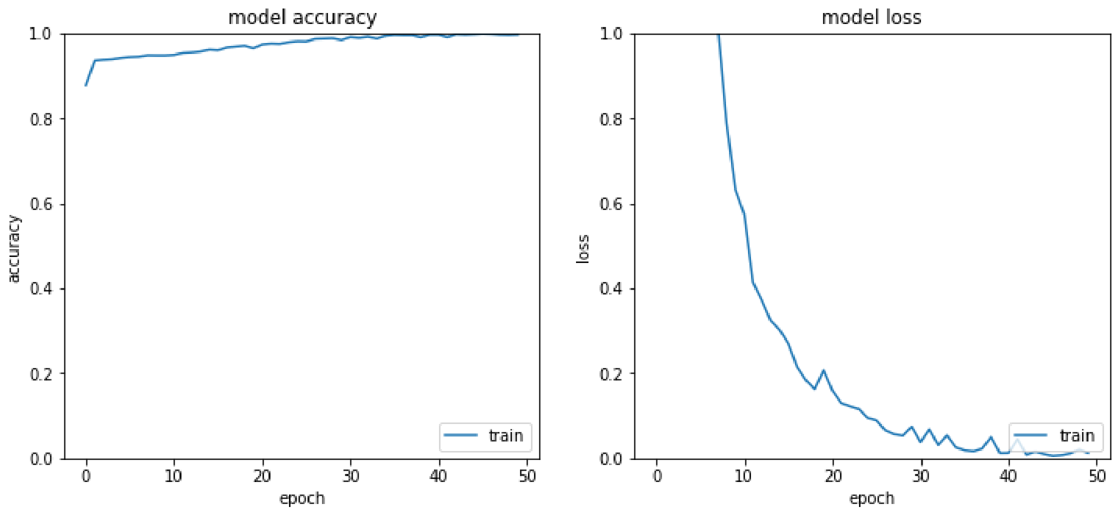
```
accuracy: 0.9424
Epoch 6/50
219/219 [==============================] - 0s 2ms/step - loss: 1.5904 -
accuracy: 0.9444
Epoch 7/50
219/219 [==============================] - 0s 2ms/step - loss: 1.2752 -
accuracy: 0.9453
Epoch 8/50
219/219 [==============================] - 0s 2ms/step - loss: 1.0173 -
accuracy: 0.9484
Epoch 9/50
219/219 [==============================] - 0s 2ms/step - loss: 0.7898 -
accuracy: 0.9481
Epoch 10/50
219/219 [==============================] - 0s 2ms/step - loss: 0.6325 -
accuracy: 0.9481
Epoch 11/50
219/219 [==============================] - 0s 2ms/step - loss: 0.5740 -
accuracy: 0.9494
Epoch 12/50
219/219 [==============================] - 0s 2ms/step - loss: 0.4138 -
accuracy: 0.9541
Epoch 13/50
219/219 [==============================] - 0s 2ms/step - loss: 0.3714 -
accuracy: 0.9553
Epoch 14/50
219/219 [==============================] - 0s 2ms/step - loss: 0.3239 -
accuracy: 0.9574
Epoch 15/50
219/219 [==============================] - 0s 2ms/step - loss: 0.3030 -
accuracy: 0.9619
Epoch 16/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2702 -
accuracy: 0.9606
Epoch 17/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2148 -
accuracy: 0.9670
Epoch 18/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1837 -
accuracy: 0.9691
Epoch 19/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1628 -
accuracy: 0.9710
Epoch 20/50
219/219 [==============================] - 0s 2ms/step - loss: 0.2073 -
accuracy: 0.9651
Epoch 21/50
219/219 [==============================] - 1s 5ms/step - loss: 0.1595 -
```

```
accuracy: 0.9741
Epoch 22/50
219/219 [==============================] - 1s 3ms/step - loss: 0.1300 -
accuracy: 0.9761
Epoch 23/50
219/219 [==============================] - 1s 6ms/step - loss: 0.1223 -
accuracy: 0.9754
Epoch 24/50
219/219 [==============================] - 0s 2ms/step - loss: 0.1159 -
accuracy: 0.9793
Epoch 25/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0953 -
accuracy: 0.9814
Epoch 26/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0891 -
accuracy: 0.9810
Epoch 27/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0659 -
accuracy: 0.9873
Epoch 28/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0569 -
accuracy: 0.9880
Epoch 29/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0532 -
accuracy: 0.9887
Epoch 30/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0743 -
accuracy: 0.9844
Epoch 31/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0382 -
accuracy: 0.9914
Epoch 32/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0683 -
accuracy: 0.9894
Epoch 33/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0311 -
accuracy: 0.9923
Epoch 34/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0537 -
accuracy: 0.9879
Epoch 35/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0259 -
accuracy: 0.9944
Epoch 36/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0188 -
accuracy: 0.9964
Epoch 37/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0165 -
```

```
accuracy: 0.9959
Epoch 38/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0229 -
accuracy: 0.9959
Epoch 39/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0500 -
accuracy: 0.9910
Epoch 40/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0124 -
accuracy: 0.9969
Epoch 41/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0127 -
accuracy: 0.9969
Epoch 42/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0454 -
accuracy: 0.9911
Epoch 43/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0083 -
accuracy: 0.9979
Epoch 44/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0157 -
accuracy: 0.9969
Epoch 45/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0101 -
accuracy: 0.9977
Epoch 46/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0056 -
accuracy: 0.9990
Epoch 47/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0073 -
accuracy: 0.9983
Epoch 48/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0116 -
accuracy: 0.9969
Epoch 49/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0209 -
accuracy: 0.9964
Epoch 50/50
219/219 [==============================] - 0s 2ms/step - loss: 0.0127 -
accuracy: 0.9971
94/94 - 0s - loss: 7.1267e-04 - accuracy: 0.9997 - 271ms/epoch - 3ms/step

Test accuracy: 0.999666690826416

Test loss: 0.0007126674754545093
```

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

[ ]:
```python
# 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()
```



[ ]:
```python
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 = 0.001 #1e-06 #
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 [==============================] - 3s 3ms/step - loss: 1.0757 -
accuracy: 0.8729
Epoch 2/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7937 -
accuracy: 0.9574
Epoch 3/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7908 -
accuracy: 0.9594
Epoch 4/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7865 -
accuracy: 0.9621
Epoch 5/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7826 -
accuracy: 0.6240
Epoch 6/50
219/219 [==============================] - 1s 2ms/step - loss: 0.7791 -
accuracy: 0.3433
Epoch 7/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7766 -
accuracy: 0.4560
Epoch 8/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7728 -
accuracy: 0.3937
Epoch 9/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7701 -
```

```
accuracy: 0.6831
Epoch 10/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7670 -
accuracy: 0.5551
Epoch 11/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7642 -
accuracy: 0.7180
Epoch 12/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7614 -
accuracy: 0.6241
Epoch 13/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7588 -
accuracy: 0.4771
Epoch 14/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7564 -
accuracy: 0.6461
Epoch 15/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7542 -
accuracy: 0.6146
Epoch 16/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7522 -
accuracy: 0.4636
Epoch 17/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7508 -
accuracy: 0.7271
Epoch 18/50
219/219 [==============================] - 1s 4ms/step - loss: 0.7488 -
accuracy: 0.6497
Epoch 19/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7479 -
accuracy: 0.5296
Epoch 20/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7462 -
accuracy: 0.4343
Epoch 21/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7450 -
accuracy: 0.5571
Epoch 22/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7438 -
accuracy: 0.5479
Epoch 23/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7431 -
accuracy: 0.4851
Epoch 24/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7417 -
accuracy: 0.5131
Epoch 25/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7406 -
```

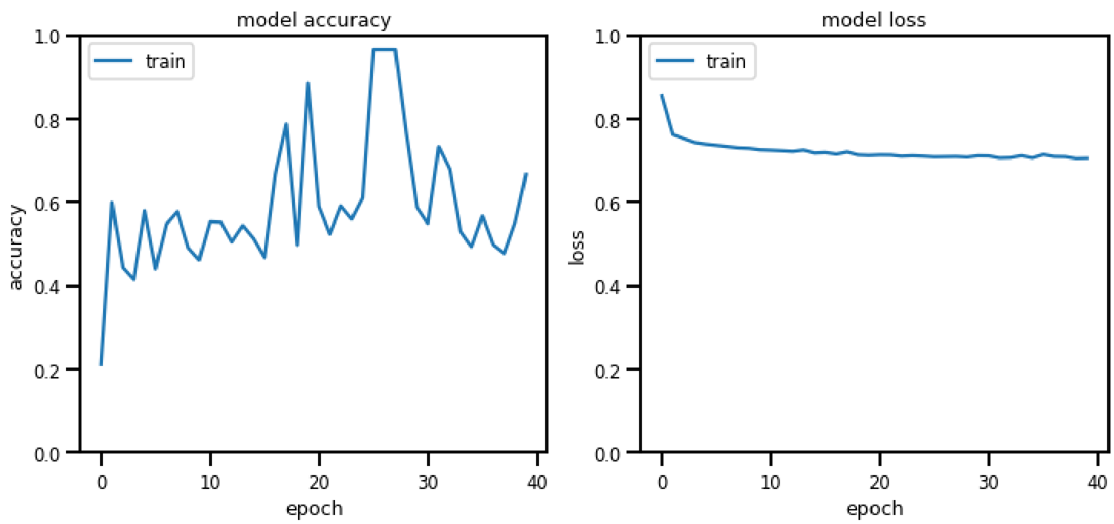
```
accuracy: 0.6360
Epoch 26/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7400 -
accuracy: 0.4933
Epoch 27/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7388 -
accuracy: 0.5763
Epoch 28/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7378 -
accuracy: 0.7509
Epoch 29/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7369 -
accuracy: 0.5059
Epoch 30/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7359 -
accuracy: 0.4609
Epoch 31/50
219/219 [==============================] - 1s 4ms/step - loss: 0.7357 -
accuracy: 0.5491
Epoch 32/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7347 -
accuracy: 0.4850
Epoch 33/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7338 -
accuracy: 0.5180
Epoch 34/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7330 -
accuracy: 0.7883
Epoch 35/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7331 -
accuracy: 0.9646
Epoch 36/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7318 -
accuracy: 0.4869
Epoch 37/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7319 -
accuracy: 0.2961
Epoch 38/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7311 -
accuracy: 0.4657
Epoch 39/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7300 -
accuracy: 0.7320
Epoch 40/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7294 -
accuracy: 0.6094
Epoch 41/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7288 -
```

```
accuracy: 0.3671
Epoch 42/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7281 -
accuracy: 0.0351
Epoch 43/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7289 -
accuracy: 0.2726
Epoch 44/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7288 -
accuracy: 0.5110
Epoch 45/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7276 -
accuracy: 0.7980
Epoch 46/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7263 -
accuracy: 0.6626
Epoch 47/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7257 -
accuracy: 0.7466
Epoch 48/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7252 -
accuracy: 0.7820
Epoch 49/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7283 -
accuracy: 0.5833
Epoch 50/50
219/219 [==============================] - 1s 3ms/step - loss: 0.7265 -
accuracy: 0.5116
94/94 [==============================] - 1s 2ms/step - loss: 0.7267 - accuracy:
0.9687

Test accuracy: 0.968666672706604

Test loss: 0.7267147898674011
```

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

[ ]:
```python
# 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()
```



[ ]:
```
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 4ms/step - loss: 2.4019 -
accuracy: 0.4121
Epoch 2/50
219/219 [==============================] - 1s 4ms/step - loss: 2.3805 -
accuracy: 0.4133
Epoch 3/50
219/219 [==============================] - 1s 4ms/step - loss: 2.3272 -
accuracy: 0.4190
Epoch 4/50
219/219 [==============================] - 1s 4ms/step - loss: 2.2809 -
accuracy: 0.4153
Epoch 5/50
219/219 [==============================] - 1s 4ms/step - loss: 2.3229 -
accuracy: 0.4161
```

```
Epoch 6/50
219/219 [==============================] - 1s 4ms/step - loss: 2.2498 -
accuracy: 0.4149
Epoch 7/50
219/219 [==============================] - 1s 4ms/step - loss: 2.3149 -
accuracy: 0.4020
Epoch 8/50
219/219 [==============================] - 1s 4ms/step - loss: 2.2349 -
accuracy: 0.4023
Epoch 9/50
219/219 [==============================] - 1s 4ms/step - loss: 2.2456 -
accuracy: 0.4134
Epoch 10/50
219/219 [==============================] - 1s 3ms/step - loss: 2.2393 -
accuracy: 0.4119
Epoch 11/50
219/219 [==============================] - 1s 4ms/step - loss: 2.2473 -
accuracy: 0.4147
Epoch 12/50
219/219 [==============================] - 1s 4ms/step - loss: 2.1595 -
accuracy: 0.4086
Epoch 13/50
219/219 [==============================] - 1s 4ms/step - loss: 2.2436 -
accuracy: 0.4179
Epoch 14/50
219/219 [==============================] - 1s 3ms/step - loss: 2.1476 -
accuracy: 0.4146
Epoch 15/50
219/219 [==============================] - 1s 3ms/step - loss: 2.1008 -
accuracy: 0.4121
Epoch 16/50
219/219 [==============================] - 1s 3ms/step - loss: 2.0665 -
accuracy: 0.4091
Epoch 17/50
219/219 [==============================] - 1s 3ms/step - loss: 2.1306 -
accuracy: 0.4167
Epoch 18/50
219/219 [==============================] - 1s 3ms/step - loss: 2.0765 -
accuracy: 0.4129
Epoch 19/50
219/219 [==============================] - 1s 3ms/step - loss: 2.0142 -
accuracy: 0.4100
Epoch 20/50
219/219 [==============================] - 1s 3ms/step - loss: 2.0783 -
accuracy: 0.4184
Epoch 21/50
219/219 [==============================] - 1s 4ms/step - loss: 1.9924 -
accuracy: 0.4064
```

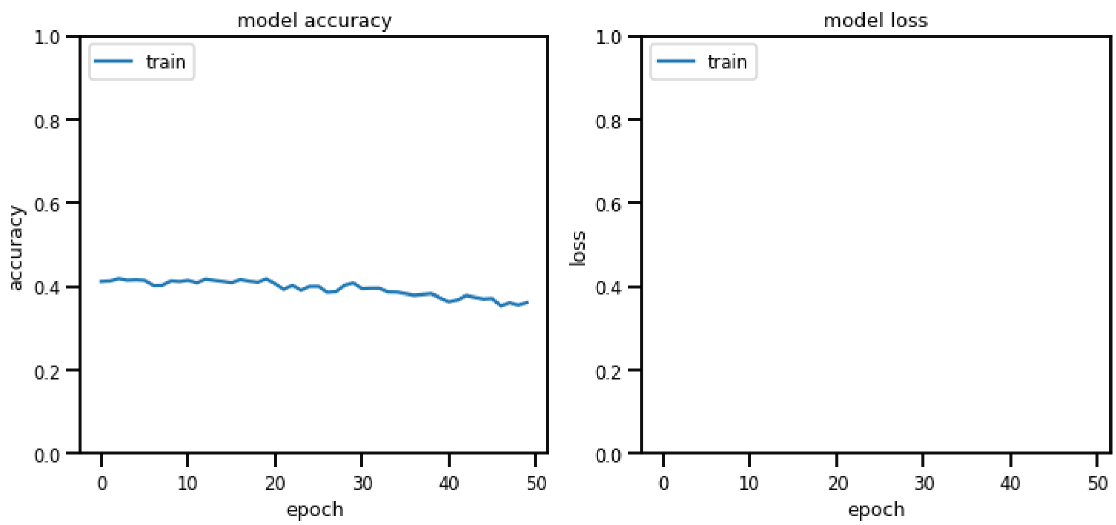
```
Epoch 22/50
219/219 [==============================] - 1s 3ms/step - loss: 1.9901 -
accuracy: 0.3927
Epoch 23/50
219/219 [==============================] - 1s 3ms/step - loss: 1.9788 -
accuracy: 0.4024
Epoch 24/50
219/219 [==============================] - 1s 3ms/step - loss: 1.9679 -
accuracy: 0.3907
Epoch 25/50
219/219 [==============================] - 1s 3ms/step - loss: 1.9494 -
accuracy: 0.4001
Epoch 26/50
219/219 [==============================] - 1s 3ms/step - loss: 1.9504 -
accuracy: 0.4001
Epoch 27/50
219/219 [==============================] - 1s 3ms/step - loss: 1.9217 -
accuracy: 0.3857
Epoch 28/50
219/219 [==============================] - 1s 4ms/step - loss: 1.8832 -
accuracy: 0.3871
Epoch 29/50
219/219 [==============================] - 1s 4ms/step - loss: 1.8649 -
accuracy: 0.4027
Epoch 30/50
219/219 [==============================] - 1s 3ms/step - loss: 1.8490 -
accuracy: 0.4084
Epoch 31/50
219/219 [==============================] - 1s 3ms/step - loss: 1.8713 -
accuracy: 0.3946
Epoch 32/50
219/219 [==============================] - 1s 3ms/step - loss: 1.7888 -
accuracy: 0.3956
Epoch 33/50
219/219 [==============================] - 1s 3ms/step - loss: 1.8289 -
accuracy: 0.3956
Epoch 34/50
219/219 [==============================] - 1s 3ms/step - loss: 1.8077 -
accuracy: 0.3866
Epoch 35/50
219/219 [==============================] - 1s 3ms/step - loss: 1.7749 -
accuracy: 0.3863
Epoch 36/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7281 -
accuracy: 0.3827
Epoch 37/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7523 -
accuracy: 0.3787
```

```
Epoch 38/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7316 -
accuracy: 0.3806
Epoch 39/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7281 -
accuracy: 0.3827
Epoch 40/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7213 -
accuracy: 0.3723
Epoch 41/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7255 -
accuracy: 0.3633
Epoch 42/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7048 -
accuracy: 0.3676
Epoch 43/50
219/219 [==============================] - 1s 4ms/step - loss: 1.6736 -
accuracy: 0.3780
Epoch 44/50
219/219 [==============================] - 1s 4ms/step - loss: 1.6481 -
accuracy: 0.3739
Epoch 45/50
219/219 [==============================] - 1s 4ms/step - loss: 1.7093 -
accuracy: 0.3699
Epoch 46/50
219/219 [==============================] - 1s 4ms/step - loss: 1.6310 -
accuracy: 0.3713
Epoch 47/50
219/219 [==============================] - 1s 4ms/step - loss: 1.5744 -
accuracy: 0.3533
Epoch 48/50
219/219 [==============================] - 1s 4ms/step - loss: 1.6090 -
accuracy: 0.3610
Epoch 49/50
219/219 [==============================] - 1s 3ms/step - loss: 1.5879 -
accuracy: 0.3554
Epoch 50/50
219/219 [==============================] - 1s 4ms/step - loss: 1.5770 -
accuracy: 0.3616
```

[ ]:

```
# 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 = 0.005 #1e-06 #
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
```

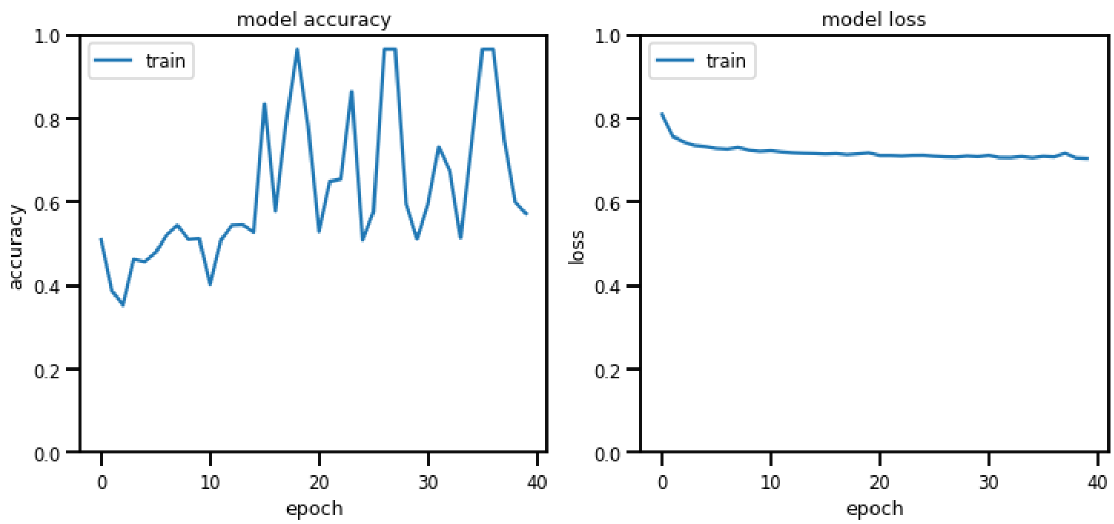
[ ]:
```
#fitting the model on the training data
history = model_tfp_v1.fit(X_train, y_train,␣
↪epochs=40)#,batch_size=1,validation_data = (np.asarray(X_test), np.
↪asarray(y_test)),verbose=0)
#evaluating the model on the test dataset
test_loss, test_acc = model_tfp_v1.evaluate(X_test,  y_test)
print('\nTest 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()
```

```python
plot_model(model_tfp_v1, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

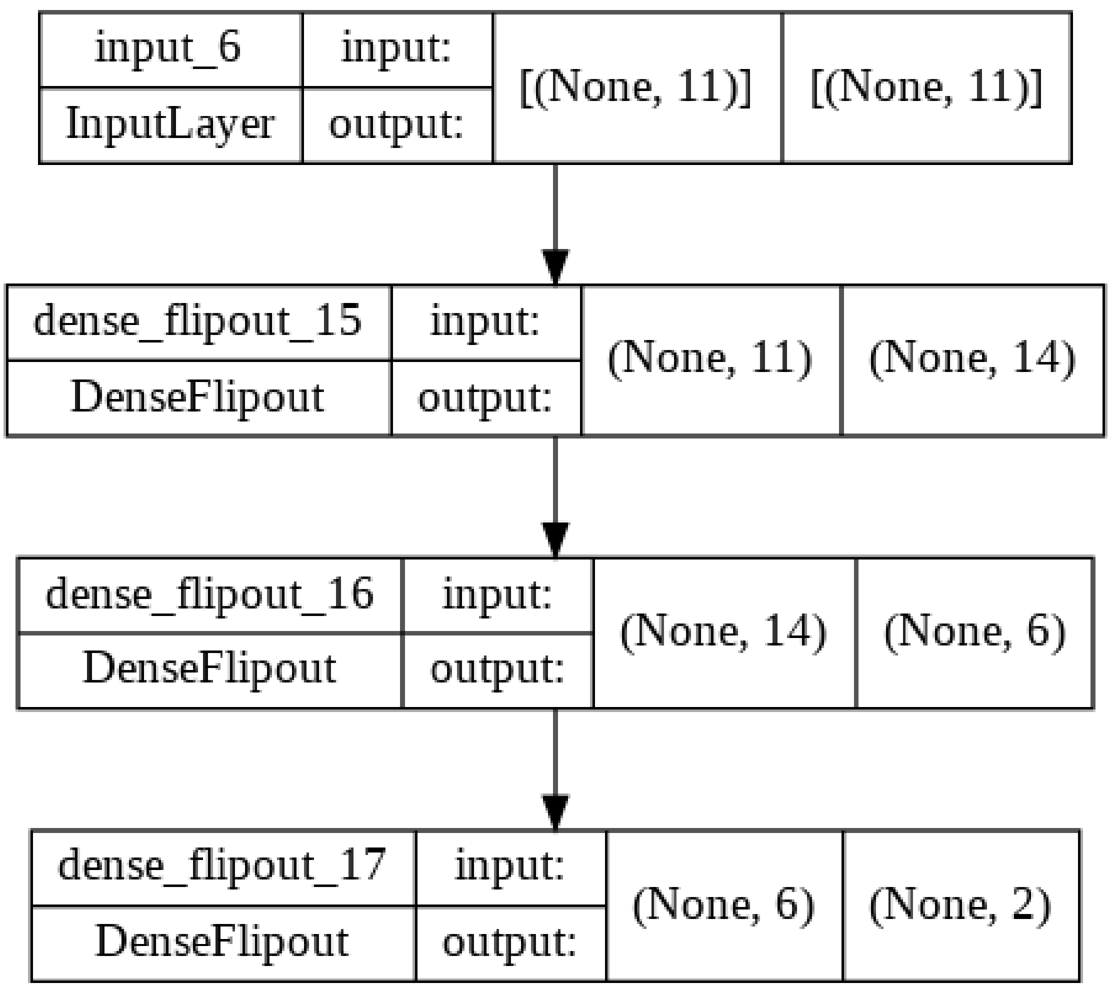
```
Epoch 1/40
219/219 [==============================] - 3s 3ms/step - loss: 0.8091 -
accuracy: 0.5089
Epoch 2/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7567 -
accuracy: 0.3874
Epoch 3/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7431 -
accuracy: 0.3524
Epoch 4/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7349 -
accuracy: 0.4621
Epoch 5/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7321 -
accuracy: 0.4561
Epoch 6/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7277 -
accuracy: 0.4784
Epoch 7/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7261 -
accuracy: 0.5200
Epoch 8/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7302 -
accuracy: 0.5444
Epoch 9/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7234 -
accuracy: 0.5099
Epoch 10/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7206 -
accuracy: 0.5121
Epoch 11/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7223 -
accuracy: 0.4017
Epoch 12/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7191 -
accuracy: 0.5080
Epoch 13/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7171 -
accuracy: 0.5443
Epoch 14/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7160 -
accuracy: 0.5454
Epoch 15/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7154 -
```

```
accuracy: 0.5271
Epoch 16/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7141 -
accuracy: 0.8340
Epoch 17/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7152 -
accuracy: 0.5776
Epoch 18/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7125 -
accuracy: 0.7929
Epoch 19/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7146 -
accuracy: 0.9647
Epoch 20/40
219/219 [==============================] - 1s 4ms/step - loss: 0.7169 -
accuracy: 0.7817
Epoch 21/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7104 -
accuracy: 0.5286
Epoch 22/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7105 -
accuracy: 0.6486
Epoch 23/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7094 -
accuracy: 0.6546
Epoch 24/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7108 -
accuracy: 0.8633
Epoch 25/40
219/219 [==============================] - 1s 5ms/step - loss: 0.7110 -
accuracy: 0.5076
Epoch 26/40
219/219 [==============================] - 1s 6ms/step - loss: 0.7089 -
accuracy: 0.5757
Epoch 27/40
219/219 [==============================] - 2s 7ms/step - loss: 0.7073 -
accuracy: 0.9649
Epoch 28/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7067 -
accuracy: 0.9649
Epoch 29/40
219/219 [==============================] - 1s 4ms/step - loss: 0.7094 -
accuracy: 0.5957
Epoch 30/40
219/219 [==============================] - 1s 4ms/step - loss: 0.7078 -
accuracy: 0.5111
Epoch 31/40
219/219 [==============================] - 1s 4ms/step - loss: 0.7109 -
```

```
accuracy: 0.5956
Epoch 32/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7056 -
accuracy: 0.7310
Epoch 33/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7053 -
accuracy: 0.6747
Epoch 34/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7081 -
accuracy: 0.5136
Epoch 35/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7049 -
accuracy: 0.7377
Epoch 36/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7085 -
accuracy: 0.9646
Epoch 37/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7071 -
accuracy: 0.9647
Epoch 38/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7158 -
accuracy: 0.7479
Epoch 39/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7046 -
accuracy: 0.6004
Epoch 40/40
219/219 [==============================] - 1s 3ms/step - loss: 0.7035 -
accuracy: 0.5716
94/94 [==============================] - 1s 2ms/step - loss: 0.7034 - accuracy:
0.9690

Test accuracy: 0.968999981880188

Test loss: 0.7033995985984802
dict_keys(['loss', 'accuracy'])
```


model accuracy
train
accuracy
epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
model loss
train
loss
epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40

[ ]:


input_6
InputLayer
input:
output:
[(None, 11)]
[(None, 11)]
dense_flipout_15
DenseFlipout
input:
output:
(None, 11)
(None, 14)
dense_flipout_16
DenseFlipout
input:
output:
(None, 14)
(None, 6)
dense_flipout_17
DenseFlipout
input:
output:
(None, 6)
(None, 2)

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

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_15 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_16 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_17 (DenseFlip  (None, 2)                26
 out)

=================================================================
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_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_15 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_16 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_17 (DenseFlip  (None, 2)                26
 out)

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

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

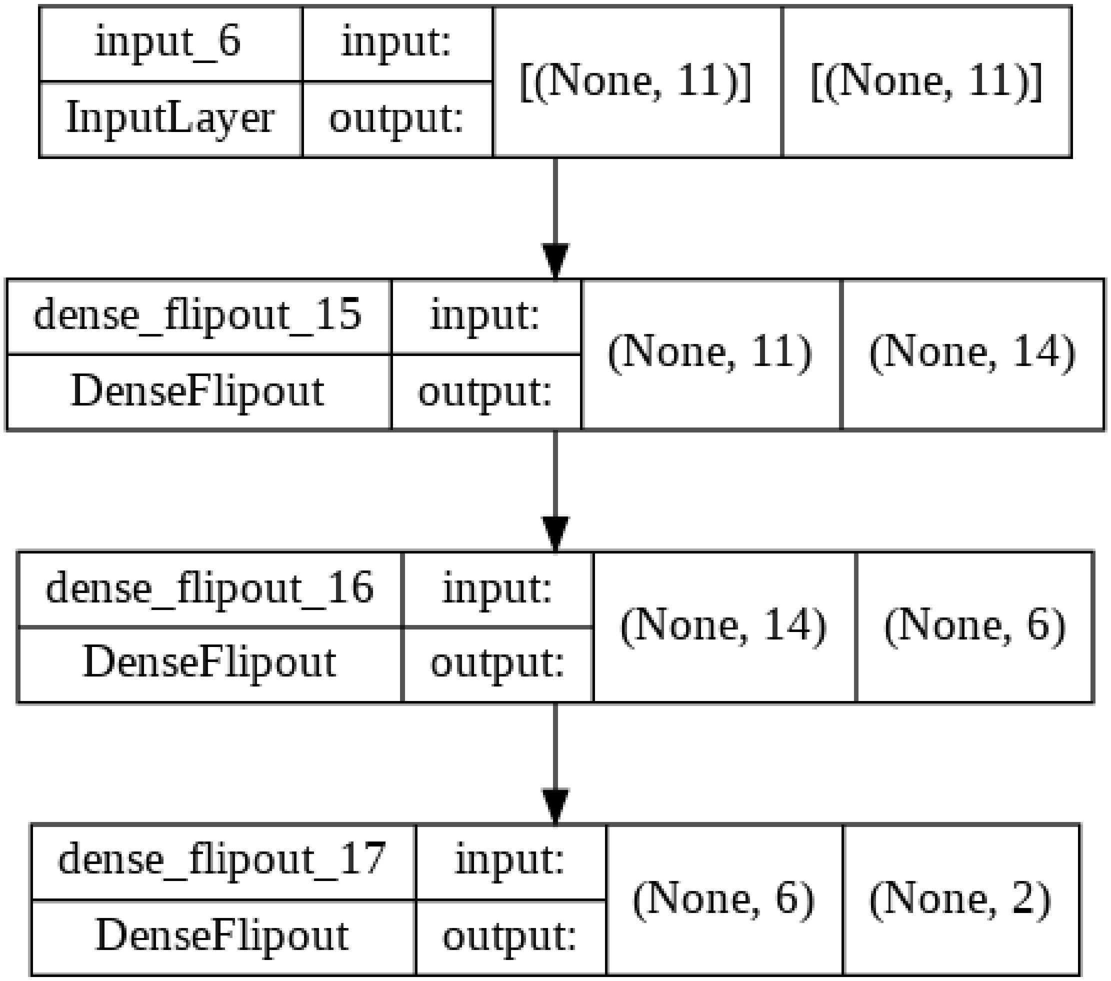
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting ann_visualizer
  Downloading ann_visualizer-2.5.tar.gz (4.7 kB)
Building wheels for collected packages: ann-visualizer
  Building wheel for ann-visualizer (setup.py) … done
  Created wheel for ann-visualizer: filename=ann_visualizer-2.5-py3-none-any.whl
size=4168
sha256=29c69b3f6d77a59efcf04d9e258a2946bf6d3b8c8b3fff153ba417c31a5da620
```

```
  Stored in directory: /root/.cache/pip/wheels/1b/fc/58/2ab1c3b30350105929308bec
dddа4fb59b1358e54f985e1f4a
Successfully built ann-visualizer
Installing collected packages: ann-visualizer
Successfully installed ann-visualizer-2.5
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
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 = 0.005 #1e-06#
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 [==============================] - 3s 3ms/step - loss: 0.9311 -
accuracy: 0.0879
Epoch 2/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7601 -
accuracy: 0.4794
Epoch 3/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7468 -
accuracy: 0.6104
Epoch 4/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7393 -
accuracy: 0.5711
Epoch 5/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7363 -
accuracy: 0.5110
Epoch 6/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7319 -
accuracy: 0.4351
Epoch 7/80
219/219 [==============================] - 1s 4ms/step - loss: 0.7288 -
accuracy: 0.4657
Epoch 8/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7265 -
accuracy: 0.4647
Epoch 9/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7240 -
accuracy: 0.4300
Epoch 10/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7233 -
accuracy: 0.4793
Epoch 11/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7216 -
accuracy: 0.5134
Epoch 12/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7197 -
accuracy: 0.5316
Epoch 13/80
```

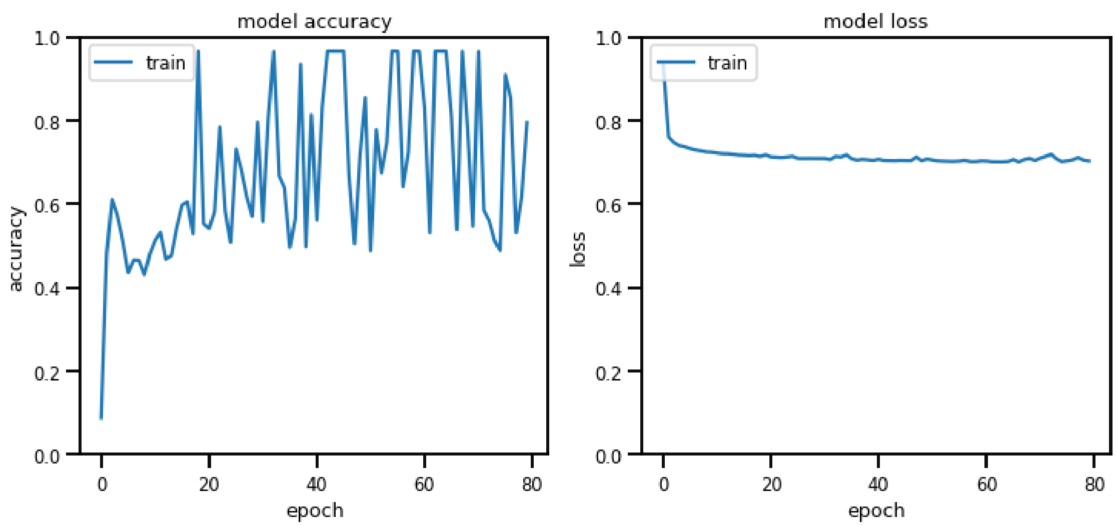
```
219/219 [==============================] - 1s 3ms/step - loss: 0.7195 -
accuracy: 0.4676
Epoch 14/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7183 -
accuracy: 0.4764
Epoch 15/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7168 -
accuracy: 0.5439
Epoch 16/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7161 -
accuracy: 0.5971
Epoch 17/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7153 -
accuracy: 0.6049
Epoch 18/80
219/219 [==============================] - 1s 4ms/step - loss: 0.7161 -
accuracy: 0.5280
Epoch 19/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7133 -
accuracy: 0.9649
Epoch 20/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7173 -
accuracy: 0.5521
Epoch 21/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7121 -
accuracy: 0.5409
Epoch 22/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7114 -
accuracy: 0.5807
Epoch 23/80
219/219 [==============================] - 1s 4ms/step - loss: 0.7106 -
accuracy: 0.7837
Epoch 24/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7119 -
accuracy: 0.5833
Epoch 25/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7136 -
accuracy: 0.5080
Epoch 26/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7088 -
accuracy: 0.7309
Epoch 27/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7085 -
accuracy: 0.6814
Epoch 28/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7087 -
accuracy: 0.6179
Epoch 29/80
```

```
219/219 [==============================] - 1s 3ms/step - loss: 0.7085 -
accuracy: 0.5709
Epoch 30/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7086 -
accuracy: 0.7956
Epoch 31/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7085 -
accuracy: 0.5579
Epoch 32/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7059 -
accuracy: 0.8029
Epoch 33/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7133 -
accuracy: 0.9640
Epoch 34/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7118 -
accuracy: 0.6674
Epoch 35/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7170 -
accuracy: 0.6384
Epoch 36/80
219/219 [==============================] - 1s 3ms/step - loss: 0.7078 -
accuracy: 0.4956
Epoch 37/80
219/219 [==============================] - 1s 6ms/step - loss: 0.7045 -
accuracy: 0.5657
Epoch 38/80
219/219 [==============================] - 1s 6ms/step - loss: 0.7065 -
accuracy: 0.9337
Epoch 39/80
219/219 [==============================] - 1s 6ms/step - loss: 0.7051 -
accuracy: 0.4970
Epoch 40/80
219/219 [==============================] - 1s 7ms/step - loss: 0.7038 -
accuracy: 0.8129
Epoch 41/80
219/219 [==============================] - 1s 6ms/step - loss: 0.7066 -
accuracy: 0.5614
Epoch 42/80
219/219 [==============================] - 1s 6ms/step - loss: 0.7037 -
accuracy: 0.8314
Epoch 43/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7034 -
accuracy: 0.9649
Epoch 44/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7031 -
accuracy: 0.9649
Epoch 45/80
```

```
219/219 [==============================] - 2s 8ms/step - loss: 0.7038 -
accuracy: 0.9649
Epoch 46/80
219/219 [==============================] - 2s 7ms/step - loss: 0.7035 -
accuracy: 0.9649
Epoch 47/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7033 -
accuracy: 0.6684
Epoch 48/80
219/219 [==============================] - 2s 7ms/step - loss: 0.7120 -
accuracy: 0.5044
Epoch 49/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7031 -
accuracy: 0.7150
Epoch 50/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7074 -
accuracy: 0.8543
Epoch 51/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7046 -
accuracy: 0.4874
Epoch 52/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7028 -
accuracy: 0.7771
Epoch 53/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7022 -
accuracy: 0.6733
Epoch 54/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7017 -
accuracy: 0.7454
Epoch 55/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7015 -
accuracy: 0.9649
Epoch 56/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7023 -
accuracy: 0.9647
Epoch 57/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7040 -
accuracy: 0.6410
Epoch 58/80
219/219 [==============================] - 2s 7ms/step - loss: 0.7011 -
accuracy: 0.7229
Epoch 59/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7009 -
accuracy: 0.9649
Epoch 60/80
219/219 [==============================] - 2s 10ms/step - loss: 0.7027 -
accuracy: 0.9646
Epoch 61/80
```

```
219/219 [==============================] - 2s 7ms/step - loss: 0.7023 -
accuracy: 0.8296
Epoch 62/80
219/219 [==============================] - 1s 7ms/step - loss: 0.7007 -
accuracy: 0.5303
Epoch 63/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7005 -
accuracy: 0.9649
Epoch 64/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7004 -
accuracy: 0.9649
Epoch 65/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7012 -
accuracy: 0.9647
Epoch 66/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7061 -
accuracy: 0.8104
Epoch 67/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7000 -
accuracy: 0.5383
Epoch 68/80
219/219 [==============================] - 2s 7ms/step - loss: 0.7058 -
accuracy: 0.9643
Epoch 69/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7086 -
accuracy: 0.7787
Epoch 70/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7035 -
accuracy: 0.5463
Epoch 71/80
219/219 [==============================] - 3s 12ms/step - loss: 0.7093 -
accuracy: 0.9641
Epoch 72/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7133 -
accuracy: 0.5841
Epoch 73/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7184 -
accuracy: 0.5600
Epoch 74/80
219/219 [==============================] - 2s 9ms/step - loss: 0.7076 -
accuracy: 0.5120
Epoch 75/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7009 -
accuracy: 0.4879
Epoch 76/80
219/219 [==============================] - 2s 7ms/step - loss: 0.7028 -
accuracy: 0.9087
Epoch 77/80
```

```
219/219 [==============================] - 2s 8ms/step - loss: 0.7053 -
accuracy: 0.8533
Epoch 78/80
219/219 [==============================] - 2s 8ms/step - loss: 0.7107 -
accuracy: 0.5307
Epoch 79/80
219/219 [==============================] - 1s 7ms/step - loss: 0.7047 -
accuracy: 0.6173
Epoch 80/80
219/219 [==============================] - 1s 4ms/step - loss: 0.7026 -
accuracy: 0.7944
94/94 [==============================] - 1s 2ms/step - loss: 0.7438 - accuracy:
0.9667

Test accuracy: 0.9666666388511658

Test loss: 0.7437806129455566
dict_keys(['loss', 'accuracy'])
INFO:tensorflow:Assets written to: saved_model/model_tfp_v2/assets
```



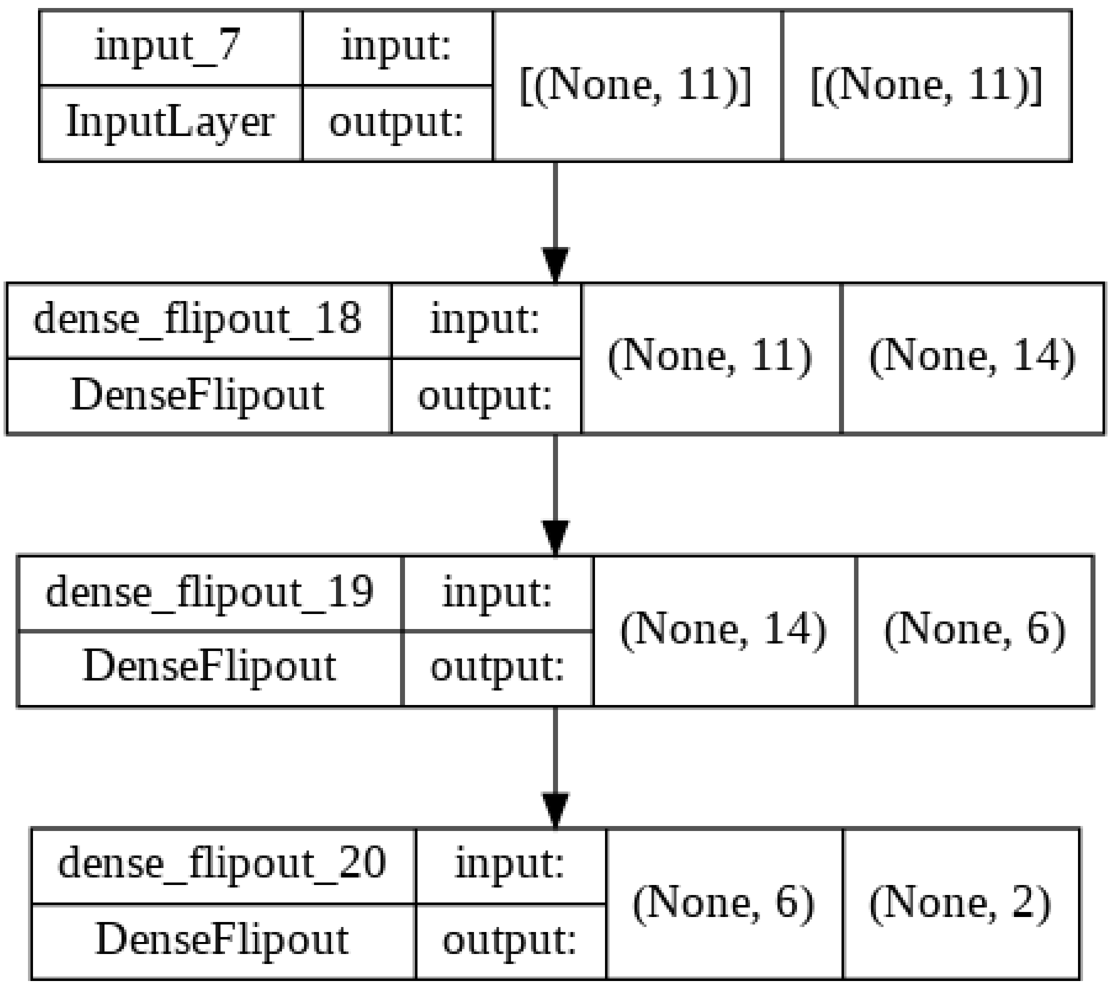

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_18 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_19 (DenseFlip  (None, 6)                174
 out)
```

```
 dense_flipout_20 (DenseFlip  (None, 2)                26
 out)

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

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

[ ]:
```python
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
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages
(2.8.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: ann_visualizer in /usr/local/lib/python3.7/dist-
packages (2.5)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
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)
```

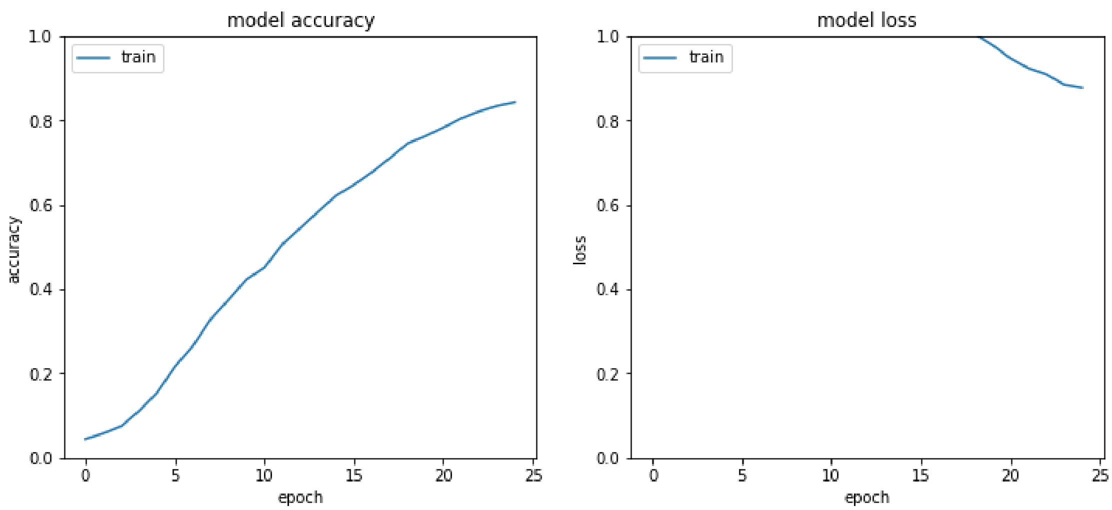
[ ]:
```
# TRY REMOVING THE VALIDATION PART FROM THE FIT
# validation_data = (np.asarray(X_test), np.asarray(y_test))
#history = normal_bnn_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=50,validation_split=0.2, shuffle=True)
# to see history:
history = normal_bnn_model.fit(np.asarray(X_train), 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 4ms/step - loss: 0.8616 - accuracy:
0.8623

Test accuracy: 0.862333357334137

Test loss: 0.8615836501121521
INFO:tensorflow:Assets written to: saved_model/normal_bnn_model/assets
```

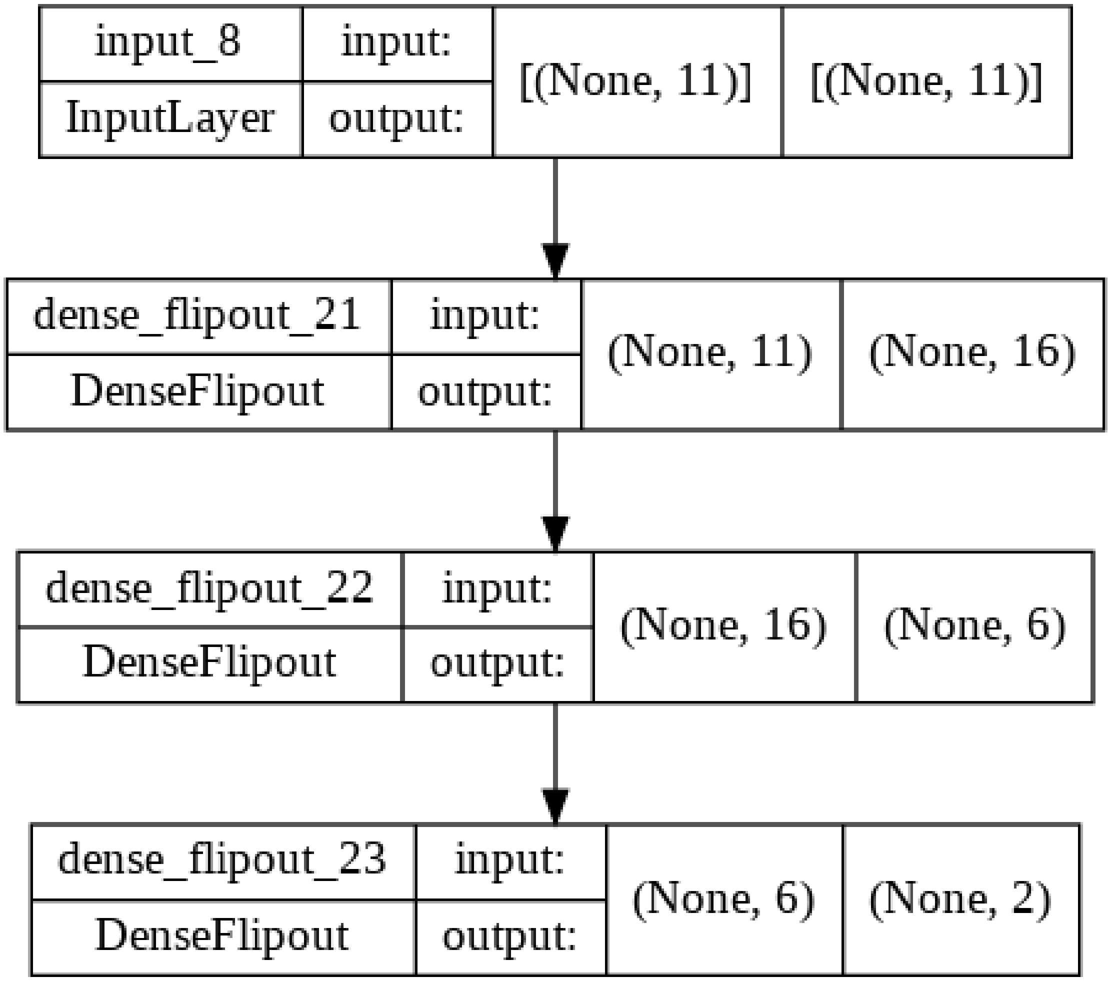
[ ]:
```
print(normal_bnn_model.predict([[2,299.1,309.5,1600,47.8,80,0,0,0,0,0]]))
```

```
[[0.54033256 0.45966744]]
```

[ ]:
```
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)
```



[ ]:



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)
```

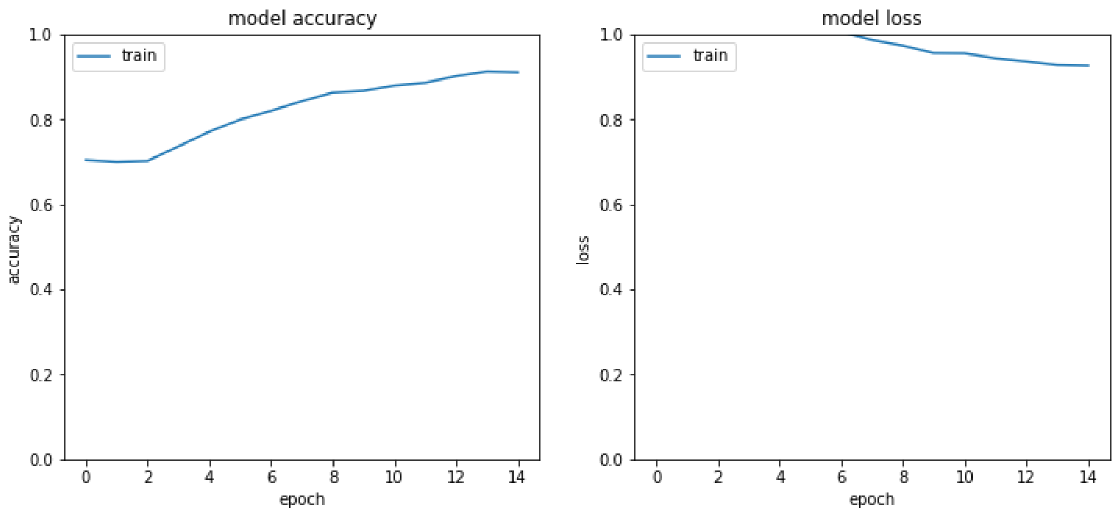
[ ]:
```
# TRY REMOVING THE VALIDATION PART FROM THE FIT
# validation_data = (np.asarray(X_test), np.asarray(y_test))
#history = normal_bnn2_model.fit(np.asarray(X_train), np.
 ↪asarray(y_train),epochs=100,validation_split=0.3, shuffle=True)
# to see history:
history = normal_bnn2_model.fit(np.asarray(X_train), 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: 0.9225 - accuracy:
0.9180

Test accuracy: 0.9179999828338623

Test loss: 0.9225282669067383
```

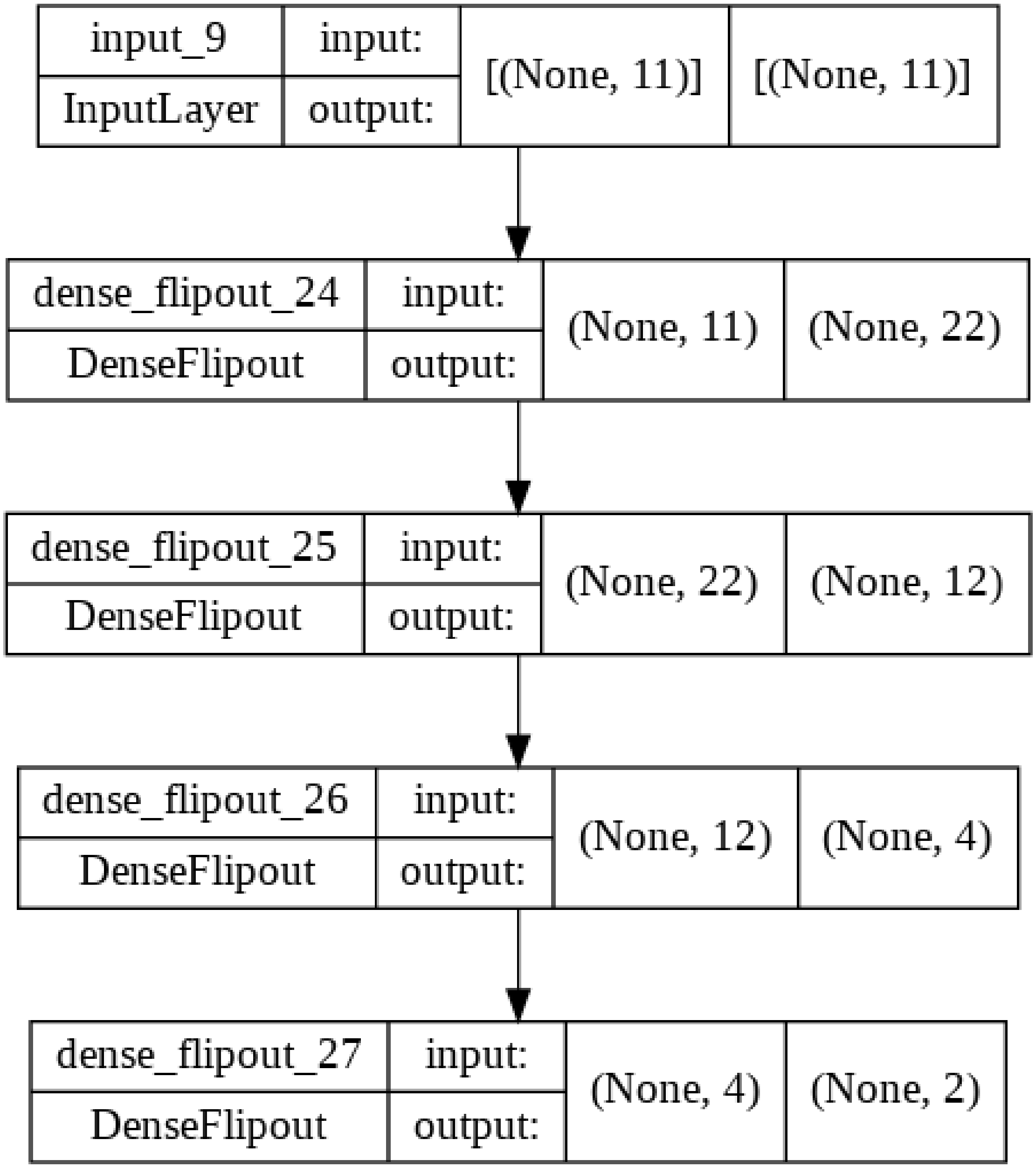
[ ]:
```
print(normal_bnn2_model.predict([[1.0,299.1,309.5,1800.0,47.8,200.0,1.0,1.0,0.
 ↪0,0.0,30.0]]))
```

```
[[0.50020003 0.49979994]]
```

[ ]:
```
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)
```



[ ]:



[ ]:
```
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)
```

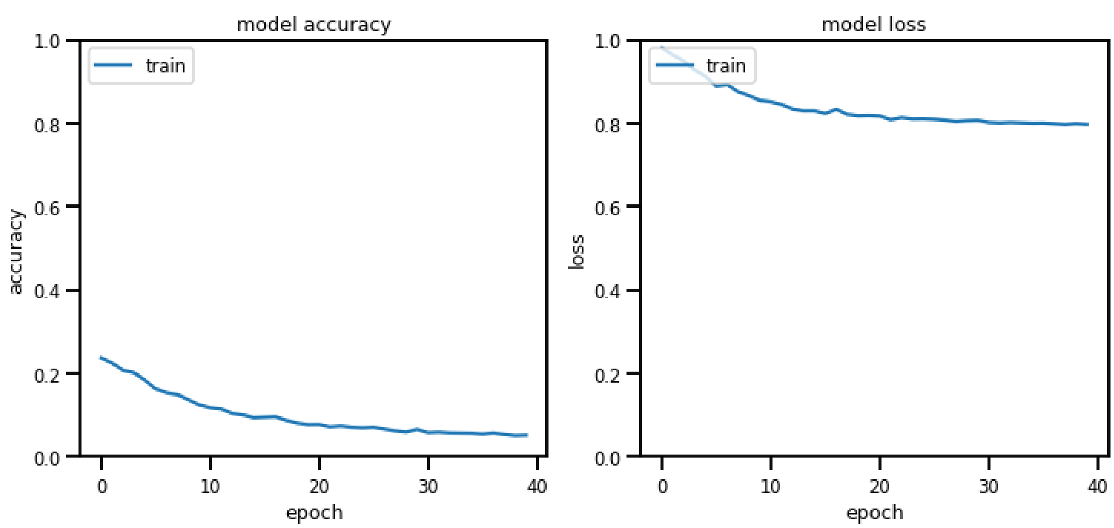
[ ]:
```
dist = tfp.distributions
dataset_size = len(X_train)
kl_divergence_function = (lambda q, p, _: dist.kl_divergence(q, p) / tf.
 ↪cast(dataset_size, dtype=tf.float32))

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)
```

```
[ ]: 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')
```

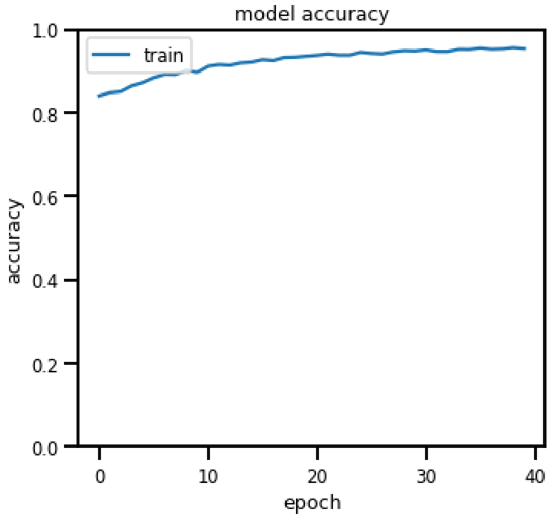
```
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.ylim(0, 1)
    plt.show()
    plot_model(p_model, to_file='model_plotss.png', show_shapes=True,␣
↪show_layer_names=True)
    '''index = 0
    for i in y_pred:
        if i<0.5:
            y_pred[index]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

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

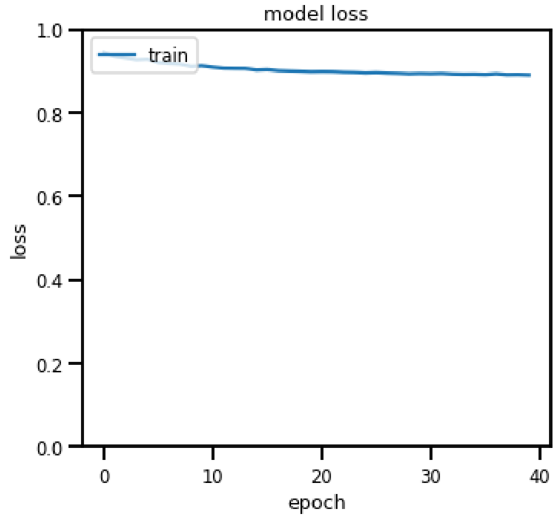
```
94/94 [==============================] - 0s 2ms/step - loss: 0.7981 - accuracy:
0.0460

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
```



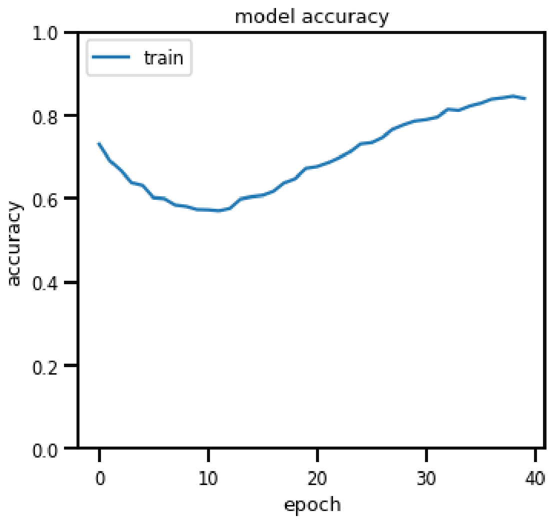

```
94/94 [==============================] - 0s 3ms/step - loss: 0.8895 - accuracy:
0.9600

Test accuracy: 0.9599999785423279
```

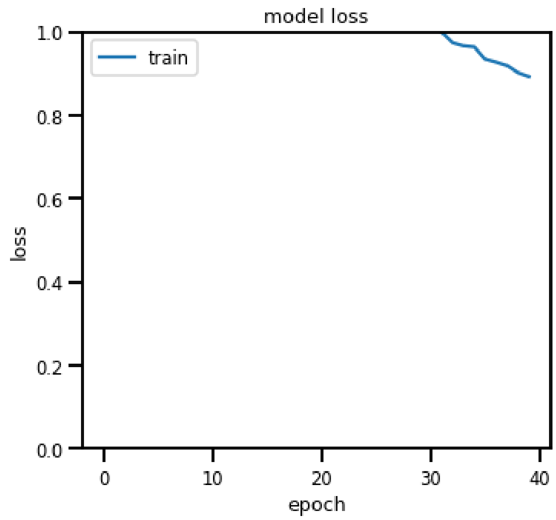
```
Test loss: 0.8895038366317749
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 2/assets
```






```
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
```

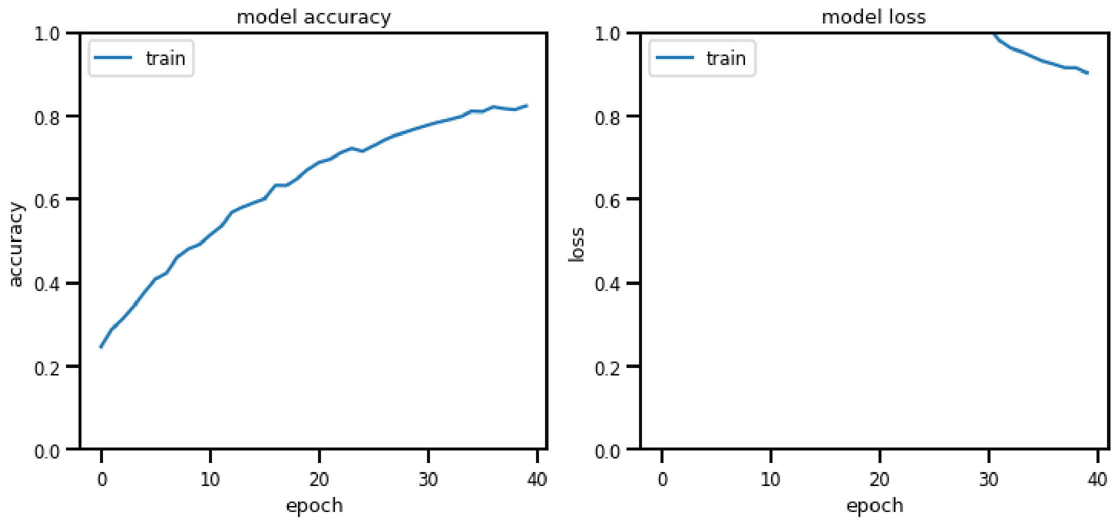




94/94 [==============================] - 1s 3ms/step - loss: 0.7885 - accuracy: 0.9517

Test accuracy: 0.9516666531562805

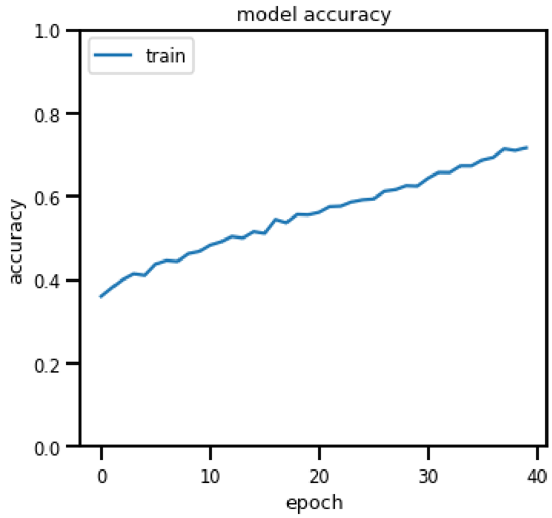
Test loss: 0.7884607911109924
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/p_model 4/assets

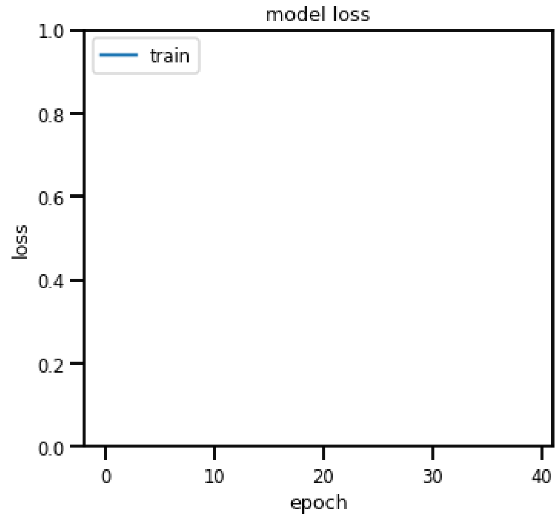


94/94 [==============================] - 1s 2ms/step - loss: 0.8642 - accuracy: 0.8343

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

[ ]:
```
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),
])
```

```
learning_rate = 0.005 #1e-06 #
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 2ms/step - loss: 1.0252 - accuracy:
0.9477

Test accuracy: 0.9476666450500488

Test loss: 1.0251801013946533
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_21 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_22 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_23 (DenseFlip  (None, 2)                26
 out)

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

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

[ ]:
```
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 = 0.005 #1e-06 #
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: 0.7097 - accuracy:
0.9680

Test accuracy: 0.9679999947547913

Test loss: 0.7096814513206482
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_24 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_25 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_26 (DenseFlip  (None, 2)                26
 out)

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

[ ]:
```
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 = 0.005 #1e-06 #
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)
```

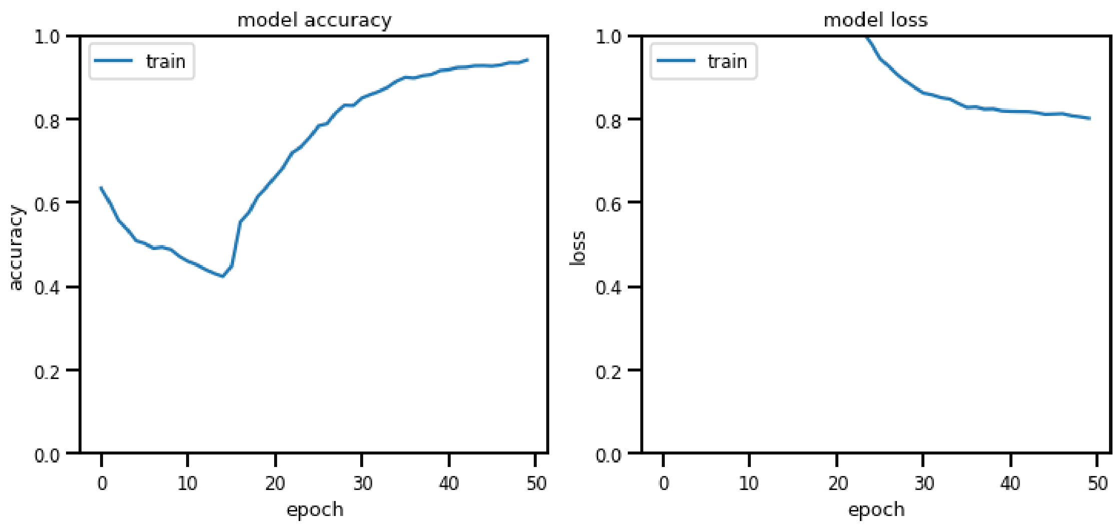
```
94/94 [==============================] - 1s 3ms/step - loss: 0.8335 - accuracy:
0.9577

Test accuracy: 0.9576666951179504

Test loss: 0.8335036039352417
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_flipout_30 (DenseFlip  (None, 14)               322
 out)

 dense_flipout_31 (DenseFlip  (None, 6)                174
 out)

 dense_flipout_32 (DenseFlip  (None, 2)                26
 out)

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

[ ]:

```
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')
```

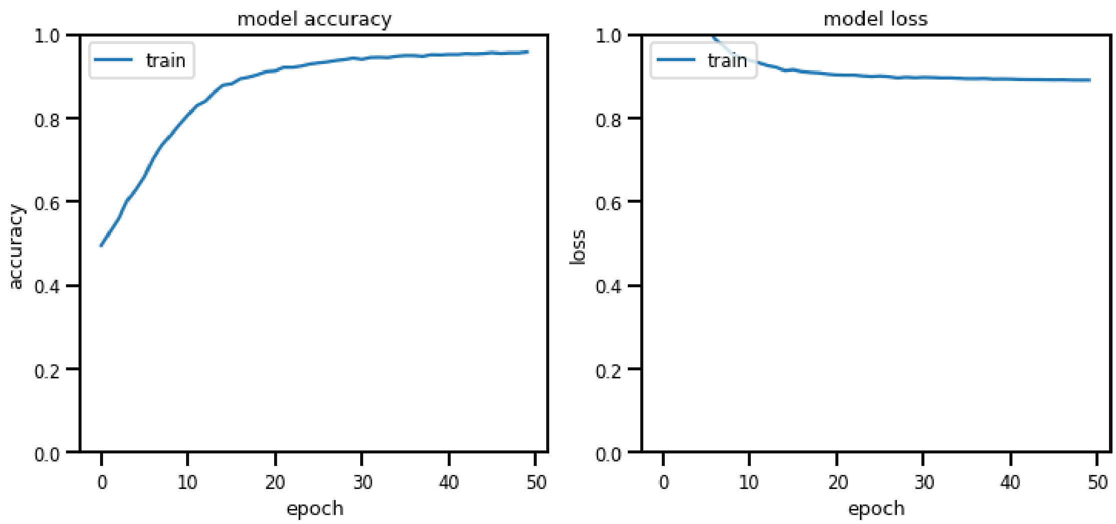
```
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.ylim(0, 1)
    plt.show()
    plot_model(p_model, to_file='model_plotsss.png', show_shapes=True,␣
↪show_layer_names=True)
    '''index = 0
    for i in y_pred:
        if i<0.5:
           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

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

```
94/94 [==============================] - 1s 3ms/step - loss: 0.8025 - accuracy:
0.9410

Test accuracy: 0.9409999847412109

Test loss: 0.8024592399597168
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 6/assets
```

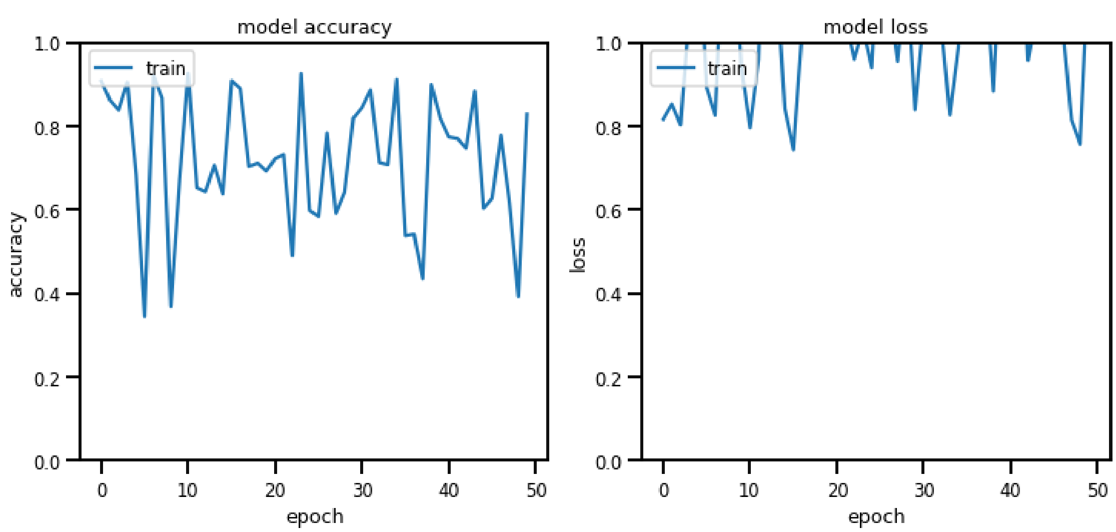



```
94/94 [==============================] - 1s 3ms/step - loss: 0.8896 - accuracy:
0.9630

Test accuracy: 0.9629999995231628
```

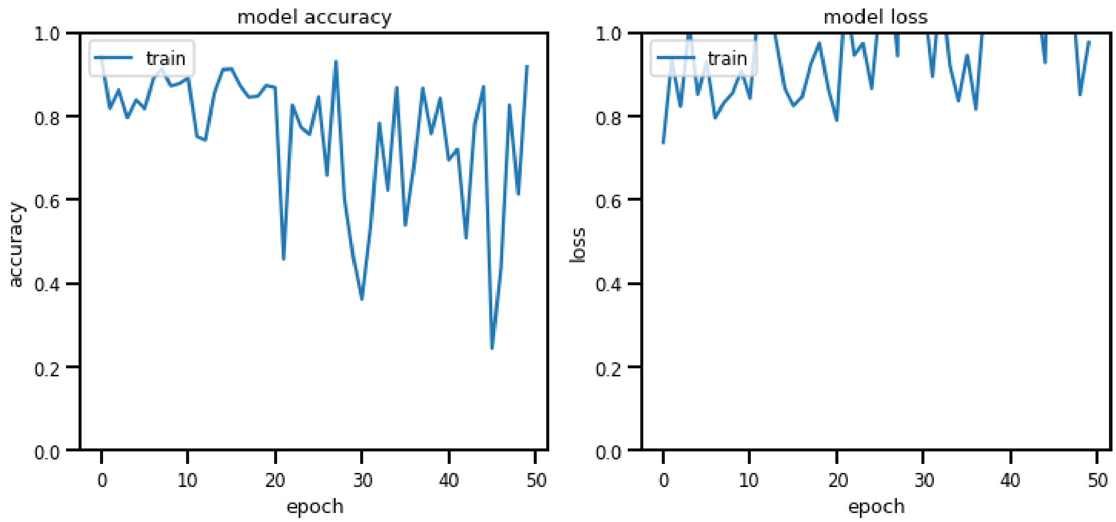
```
Test loss: 0.8896040320396423
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 7/assets
```



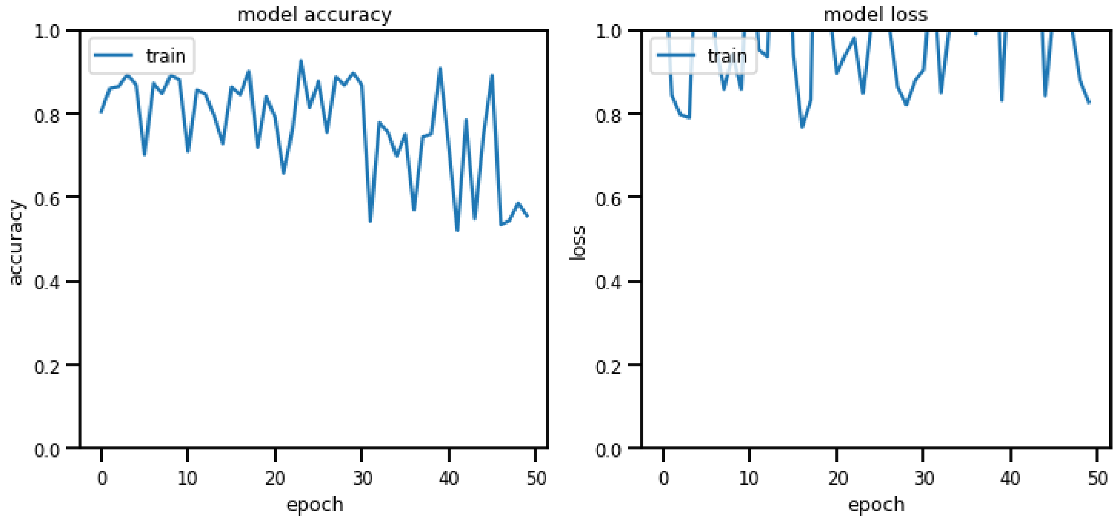

```
94/94 [==============================] - 0s 3ms/step - loss: 1.3953 - accuracy:
0.9347

Test accuracy: 0.9346666932106018

Test loss: 1.3952935934066772
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 8/assets
```



```
94/94 [==============================] - 0s 4ms/step - loss: 1.0007 - accuracy:
0.9483

Test accuracy: 0.9483333230018616

Test loss: 1.0007153749465942
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 9/assets
```




```
94/94 [==============================] - 0s 3ms/step - loss: 0.7728 - accuracy:
0.9640

Test accuracy: 0.9639999866485596

Test loss: 0.7727773189544678
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
INFO:tensorflow:Assets written to: saved_model/callp_model 10/assets
```




```
[ ]: #BNN WITH DIFFERENT REGULARIZERS TRANSFORMERS
     # MIXING OF THE ABOVE VARIANTS AND COMPARING WITH THE NORMAL ANN
```

w and b site `streamlit for gui`

```
[ ]:
```

0.0.4 WEEKLY OUTPUT PDFS

convert notebook to pdf `for weekly progrss submission`

```
[ ]:
```

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

```
/content/drive/MyDrive/Colab Notebooks/MTP
/content/drive/MyDrive/Colab Notebooks/MTP
 3rd_sem1.pdf                         dec.pdf
 3rd_sem.pdf                          material
 4th_sem_FINAL_may.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'
```

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

[ ]:
```
!jupyter nbconvert --to pdf --output "4th_sem_FINAL_Present" 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/contenth5.zip /content/*.h5
files.download("/content/contenth5.zip")
!zip -r /content/contentpng.zip /content/*.png
files.download("/content/contentpng.zip")
```

```
  adding: content/saved_model/ (stored 0%)
  adding: content/saved_model/callp_model 6/ (stored 0%)
  adding: content/saved_model/callp_model 6/variables/ (stored 0%)
  adding: content/saved_model/callp_model
6/variables/variables.data-00000-of-00001 (deflated 48%)
  adding: content/saved_model/callp_model 6/variables/variables.index (deflated
68%)
  adding: content/saved_model/callp_model 6/saved_model.pb (deflated 92%)
  adding: content/saved_model/callp_model 6/assets/ (stored 0%)
  adding: content/saved_model/callp_model 6/keras_metadata.pb (deflated 94%)
  adding: content/saved_model/model_tfp_v2/ (stored 0%)
  adding: content/saved_model/model_tfp_v2/variables/ (stored 0%)
  adding:
content/saved_model/model_tfp_v2/variables/variables.data-00000-of-00001
(deflated 54%)
  adding: content/saved_model/model_tfp_v2/variables/variables.index (deflated
68%)
  adding: content/saved_model/model_tfp_v2/saved_model.pb (deflated 92%)
  adding: content/saved_model/model_tfp_v2/assets/ (stored 0%)
  adding: content/saved_model/model_tfp_v2/keras_metadata.pb (deflated 94%)
  adding: content/saved_model/callp_model 9/ (stored 0%)
```

```
  adding: content/saved_model/callp_model 9/variables/ (stored 0%)
  adding: content/saved_model/callp_model
9/variables/variables.data-00000-of-00001 (deflated 57%)
  adding: content/saved_model/callp_model 9/variables/variables.index (deflated
68%)
  adding: content/saved_model/callp_model 9/saved_model.pb (deflated 92%)
  adding: content/saved_model/callp_model 9/assets/ (stored 0%)
  adding: content/saved_model/callp_model 9/keras_metadata.pb (deflated 94%)
  adding: content/saved_model/callp_model 10/ (stored 0%)
  adding: content/saved_model/callp_model 10/variables/ (stored 0%)
  adding: content/saved_model/callp_model
10/variables/variables.data-00000-of-00001 (deflated 56%)
  adding: content/saved_model/callp_model 10/variables/variables.index (deflated
68%)
  adding: content/saved_model/callp_model 10/saved_model.pb (deflated 92%)
  adding: content/saved_model/callp_model 10/assets/ (stored 0%)
  adding: content/saved_model/callp_model 10/keras_metadata.pb (deflated 94%)
  adding: content/saved_model/model_tfp_v1/ (stored 0%)
  adding: content/saved_model/model_tfp_v1/variables/ (stored 0%)
  adding:
content/saved_model/model_tfp_v1/variables/variables.data-00000-of-00001
(deflated 53%)
  adding: content/saved_model/model_tfp_v1/variables/variables.index (deflated
68%)
  adding: content/saved_model/model_tfp_v1/saved_model.pb (deflated 92%)
  adding: content/saved_model/model_tfp_v1/assets/ (stored 0%)
  adding: content/saved_model/model_tfp_v1/keras_metadata.pb (deflated 94%)
  adding: content/saved_model/callp_model 7/ (stored 0%)
  adding: content/saved_model/callp_model 7/variables/ (stored 0%)
  adding: content/saved_model/callp_model
7/variables/variables.data-00000-of-00001 (deflated 41%)
  adding: content/saved_model/callp_model 7/variables/variables.index (deflated
70%)
  adding: content/saved_model/callp_model 7/saved_model.pb (deflated 92%)
  adding: content/saved_model/callp_model 7/assets/ (stored 0%)
  adding: content/saved_model/callp_model 7/keras_metadata.pb (deflated 95%)
  adding: content/saved_model/callp_model 8/ (stored 0%)
  adding: content/saved_model/callp_model 8/variables/ (stored 0%)
  adding: content/saved_model/callp_model
8/variables/variables.data-00000-of-00001 (deflated 56%)
  adding: content/saved_model/callp_model 8/variables/variables.index (deflated
68%)
  adding: content/saved_model/callp_model 8/saved_model.pb (deflated 92%)
  adding: content/saved_model/callp_model 8/assets/ (stored 0%)
  adding: content/saved_model/callp_model 8/keras_metadata.pb (deflated 94%)
```

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

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

```
  adding: content/callp_model 10.h5 (deflated 86%)
  adding: content/callp_model 6.h5 (deflated 84%)
  adding: content/callp_model 7.h5 (deflated 82%)
  adding: content/callp_model 8.h5 (deflated 86%)
  adding: content/callp_model 9.h5 (deflated 87%)
  adding: content/model_tfp_v1.h5 (deflated 86%)
  adding: content/model_tfp_v2.h5 (deflated 86%)
```

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

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

```
  adding: content/model_plot1.png (deflated 13%)
  adding: content/model_plot.png (deflated 15%)
  adding: content/model_plotsss.png (deflated 13%)
```

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

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

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.
↪03133333474397659 ,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
```

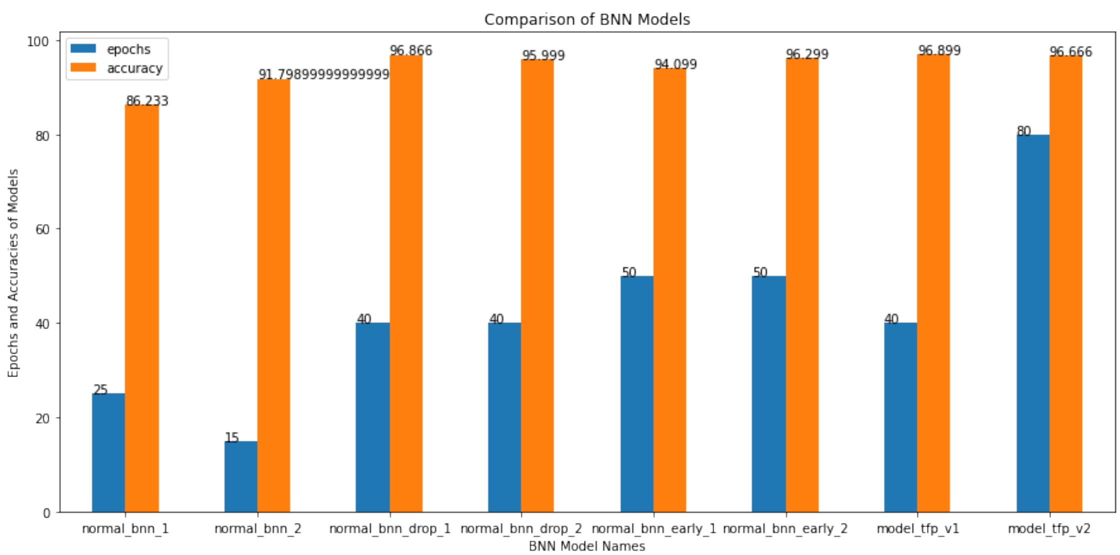
```
plot_df.plot_bokeh(kind='bar',x = train_accuracy,title = "ta")
plt.bar([train_loss,train_accuracy,test_loss,test_accuracy],labels)

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

[ ]:
```
'import numpy as np\nimport matplotlib.pyplot as plt\n\ntrain_loss = [0.4377,
0.7227, 0.7029, 0.7039, 0.7060]\ntrain_accuracy = [0.9687, 0.9693, 0.9690 ,
0.0313, 0.9687]\ntest_accuracy = [0.968666672706604,0.9693333506584167
,0.968999981880188 ,0.03133333474397659 ,0.968666672706604]\ntest_loss =
[0.43769827485084534, 0.7226769924163818,0.7028810977935791, 0.703898012638092,
0.7059929370880127]\nlabels = [\'normal_bnn1\',\'normal_bnn1\', \'dropout_1\',
\'dropout_2\', \'dropout_3\' ]\n\nplot_df = pd.DataFrame({"train_loss":train_los
s,"train_accuracy":train_accuracy,"test_accuracy":test_accuracy,"test_loss":test
_loss})\n\n#plot_df[\'train_loss\'] = train_loss\n#plot_df[\'train_accuracy\'] =
train_accuracy\n#plot_df[\'test_accuracy\'] =
test_accuracy\n#plot_df[\'test_loss\'] =
test_loss\n\nplot_df.plot_bokeh(kind=\'bar\',x = train_accuracy,title = "ta")\np
lt.bar([train_loss,train_accuracy,test_loss,test_accuracy],labels)\n\nplt.xlabel
("models")\nplt.ylabel("parameters")\nplt.title("normal bnn models vs dropout
bnn models")\nplt.show()'
```

```
Normal BNN -1:  3 - layers, epochs = 25, learning rate = 1e-06, Accuracy:  0.86233

Normal BNN -2:  4 - layers, epochs = 15, learning rate = 1e-06, Accuracy:  0.91799

Normal BNN -1:  3 - layers, epochs = 40, learning rate = 1e-06, Accuracy:  0.96866

Normal BNN -2:  4 - layers, epochs = 40, learning rate = 1e-06, Accuracy:  0.95999

Normal BNN -1:  3 - layers, epochs = 50, learning rate = 1e-06, Accuracy:  0.94099

Normal BNN -2:  4 - layers, epochs = 50, learning rate = 1e-06, Accuracy:  0.96299

Model_tfp_v1:  3 - layers, epochs = 40, learning rate = 0.002, Accuracy:  0.96899

Model_tfp_v2:  3 - layers, epochs = 80, learning rate = 0.005, Accuracy:  0.96666
```

[31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

epoch_list = [25,15,40,40,50,50,40,80]
accuracy = [0.86233,0.91799,0.96866,0.95999,0.94099,0.96299,0.96899,0.96666]
#for i in [0,len(accuracy)-1]:
 #   accuracy[i] = (accuracy[i]*100)
learning_rates = []
```

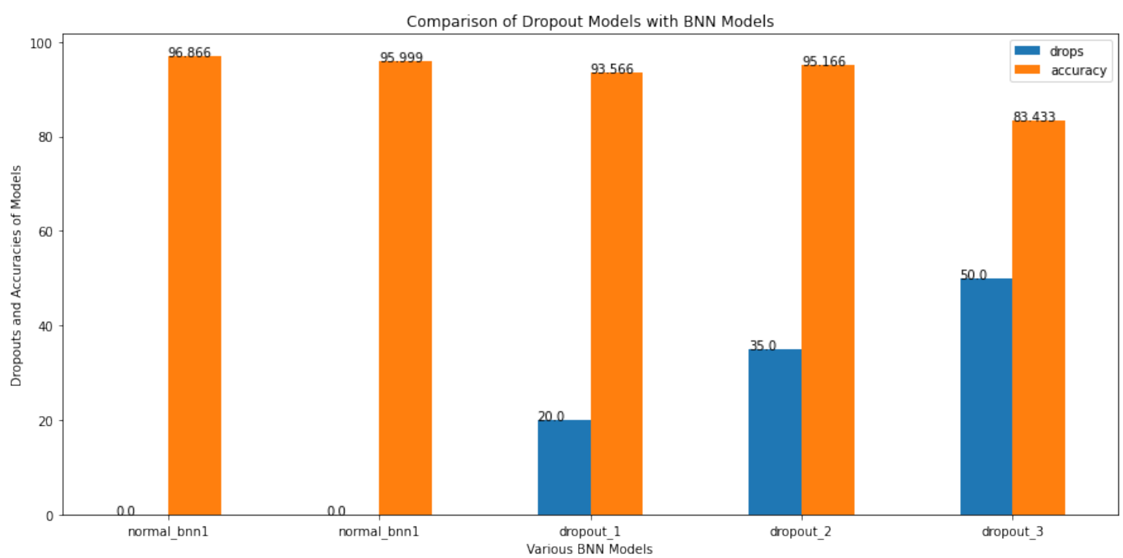
```
labels =␣
↪['normal_bnn_1','normal_bnn_2','normal_bnn_drop_1','normal_bnn_drop_2','normal_bnn_early_1'
#plt.figure(figsize = (15, 5))
df = pd.DataFrame({'epochs':epoch_list,'accuracy':accuracy},index = labels)
df['accuracy'] = df['accuracy'] * 100
ax = df.plot.bar(rot=0,figsize=(15,7),title="Comparison of BNN Models")
ax.set_xlabel("BNN Model Names")
ax.set_ylabel("Epochs and Accuracies of Models")

for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1, p.get_height() * 1))
#plt.figure(figsize = (15, 5))
#plt.bar(labels,accuracy)
#plt.title("Comparison of BNN Models")
#plt.xlabel("BNN Model Names")
#plt.ylabel("Accuracies of Models")
```




```
Monte Carlo - Dropout:  learning rate = 1e-06, epochs=40

Dropout 1 :  dropout value = 0.2, Accuracy:  0.93566

Dropout 2 :  dropout value = 0.35, Accuracy:  0.95166

Dropout 3 :  dropout value = 0.5, Accuracy:  0.83433

Normal BNN -1:  3 - layers, epochs = 40, learning rate = 1e-06, Accuracy:  0.96866

Normal BNN -2:  4 - layers, epochs = 40, learning rate = 1e-06, Accuracy:  0.95999
```

[30]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
#train_loss = [0.4377, 0.7227, 0.7029, 0.7039, 0.7060]
#train_accuracy = [0.93, 0.9, 0.9690 , 0.0313, 0.9687]
test_accuracy = [0.96866,0.95999 ,0.93566 ,0.95166 ,0.83433]
dropout_rates = [0,0,0.2,0.35,0.5]

#test_loss = [0.43769827485084534, 0.7226769924163818,0.7028810977935791, 0.
↪703898012638092, 0.7059929370880127]
labels = ['normal_bnn1','normal_bnn1', 'dropout_1', 'dropout_2', 'dropout_3' ]

df = pd.DataFrame({'drops':dropout_rates,'accuracy':test_accuracy},index =␣
↪labels)
df['accuracy'] = df['accuracy'] * 100
df['drops'] = df['drops'] * 100
ax = df.plot.bar(rot=0,figsize=(15,7),title="Comparison of Dropout Models with␣
↪BNN Models")
ax.set_xlabel("Various BNN Models")
ax.set_ylabel("Dropouts and Accuracies of Models")

for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1, p.get_height() * 1))
#for container in ax.containers:
#    ax.bar_label(container)
```




```
Early Stopping: learning rate = 0.005, epochs=50

Early Stop 1: patience = 3, Accuracy: 0.93466

Early Stop 2: patience = 4, Accuracy: 0.94833
```

```
Early Stop 3:  patience = 2, Accuracy:  0.96399

Normal BNN -1:  3 - layers, epochs = 50, learning rate = 1e-06, Accuracy:  0.94099

Normal BNN -2:  4 - layers, epochs = 50, learning rate = 1e-06, Accuracy:  0.96299
```

[32]:

```python
test_accuracy = [0.94099,0.96299 ,0.93466 ,0.94833 ,0.96399]
patience = [0,0,3,4,2]

#test_loss = [0.43769827485084534, 0.7226769924163818,0.7028810977935791, 0.
↪703898012638092, 0.7059929370880127]
labels = ['normal_bnn1','normal_bnn1', 'early_1', 'early_2', 'early_3' ]

df = pd.DataFrame({'patience':patience,'accuracy':test_accuracy},index = labels)
df['accuracy'] = df['accuracy'] * 100
df['drops'] = df['drops'] * 100
ax = df.plot.bar(rot=0,figsize=(15,7),title="Comparison of Early Stop Models␣
↪with BNN Models")
ax.set_xlabel("Various BNN Models")
ax.set_ylabel("Patience and Accuracies of Models")

for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1, p.get_height() * 1))
```

```
       ␣
↪---------------------------------------------------------------------------

       KeyError                                  Traceback (most recent call␣
↪last)

       /usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in␣
↪get_loc(self, key, method, tolerance)
      3360             try:
   -> 3361                 return self._engine.get_loc(casted_key)
      3362             except KeyError as err:


       /usr/local/lib/python3.7/dist-packages/pandas/_libs/index.pyx in pandas.
↪_libs.index.IndexEngine.get_loc()


       /usr/local/lib/python3.7/dist-packages/pandas/_libs/index.pyx in pandas.
↪_libs.index.IndexEngine.get_loc()


       pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()
```

```
    pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


    KeyError: 'drops'


  The above exception was the direct cause of the following exception:


    KeyError                                  Traceback (most recent call␣
↪last)

    <ipython-input-32-ab47efc3cfa9> in <module>()
      7 df = pd.DataFrame({'patience':patience,'accuracy':
↪test_accuracy},index = labels)
      8 df['accuracy'] = df['accuracy'] * 100
  ----> 9 df['drops'] = df['drops'] * 100
     10 ax = df.plot.bar(rot=0,figsize=(15,7),title="Comparison of Early␣
↪Stop Models with BNN Models")
     11 ax.set_xlabel("Various BNN Models")


    /usr/local/lib/python3.7/dist-packages/pandas/core/frame.py in␣
↪__getitem__(self, key)
   3456             if self.columns.nlevels > 1:
   3457                 return self._getitem_multilevel(key)
-> 3458             indexer = self.columns.get_loc(key)
   3459             if is_integer(indexer):
   3460                 indexer = [indexer]


    /usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in␣
↪get_loc(self, key, method, tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:


    KeyError: 'drops'
```

```
combined
```

```
[ ]:
```