Nama : Andrew

NIM : 2540119601

Kelas : LA09

Mata Kuliah : Deep Learning

Jurusan : Data Science

Libary

```
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import numpy as np
import operator
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
```

Load Data

```
from google.colab import drive
drive.mount('/content/drive')

df =
pd.read_csv('/content/drive/MyDrive/UTS-DeepLearning-Data/insurance.cs
v')
```

```
Mounted at /content/drive
```

```
df.head()
```

```
   age  sex     bmi  steps  children  smoker  region      charges  \
0   19    0  27.900   3009         0       1       3  16884.92400
1   18    1  33.770   3008         1       0       2   1725.55230
2   28    1  33.000   3009         3       0       2   4449.46200
3   33    1  22.705  10009         0       0       1  21984.47061
4   32    1  28.880   8010         0       0       1   3866.85520

   insuranceclaim
0               1
1               1
2               0
3               0
4               1
```

1A. [LO 3, LO 4, 5 poin] Dataset yang diberikan memiliki beberapa problem, lakukan praproses data untuk menyelesaikan problem dari data tersebut. Sebutkan problem apa saja yang kalian temukan dari data yang diberikan, berikan penjelasan mengenai pendekatan apa yang kalian gunakan dan kenapa memilih pendekatan yang dipilih?

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             1338 non-null   int64
 1   sex             1338 non-null   int64
 2   bmi             1338 non-null   float64
 3   steps           1338 non-null   int64
 4   children        1338 non-null   int64
 5   smoker          1338 non-null   int64
 6   region          1338 non-null   int64
 7   charges         1338 non-null   float64
 8   insuranceclaim  1338 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 94.2 KB
```

Dari sini dapat diketahui semua data sudah dalam format numerik. Sehingga tidak perlu dipindahkan kedapam category untuk memudahkan train data. Selain itu terdapat 2 data yang berbentuk desimal. Pada data ini juga terdapat 9 kolom dengan total data 1338 row.

```
print(df[df.duplicated()].shape)
```

```
(0, 9)
```

Dari sini dapat diketahui tidak ada data yang redundant satu dengan yang lainnya

```
 df.isna().sum()
```

```
age               0
sex               0
bmi               0
steps             0
children          0
smoker            0
region            0
charges           0
insuranceclaim    0
dtype: int64
```

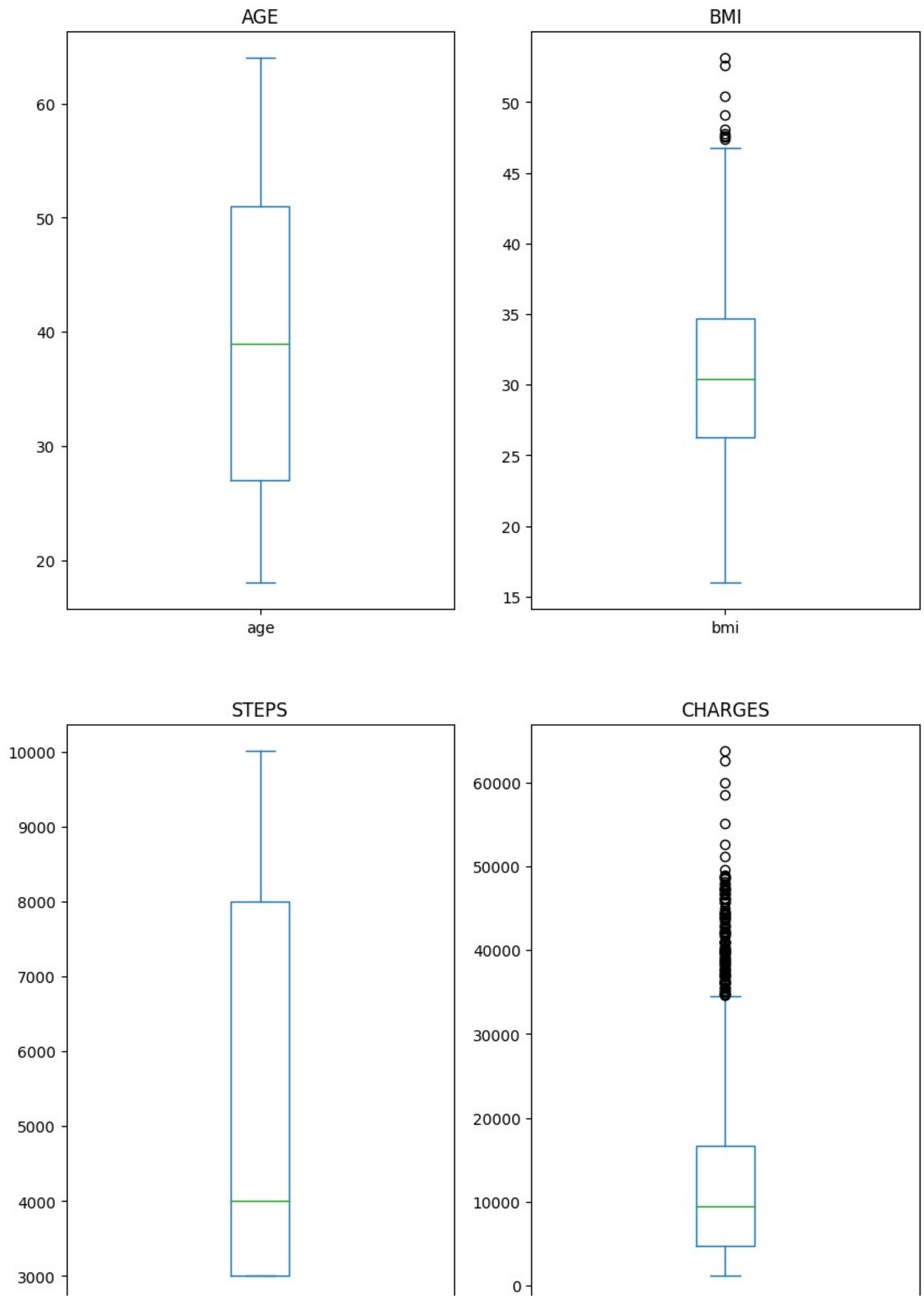
Dari missing value tidak di dapatkan sama sekali missing value pada dataset ini

Mendekteksi Outlier

```
data = ["age","bmi","steps","charges"]
plt.figure(figsize=(10, 15))
```

```python
plt.subplots_adjust(hspace=0.2)
plt.suptitle("box plot for numeric", fontsize=18)

for n,column in enumerate(data):
  ax = plt.subplot(2, 2, n + 1)
  df[column].plot(kind='box')
  ax.set_title(column.upper())
  ax.set_xlabel("")
```

box plot for numeric



Dari sini dapat dilihat visualisasi terhadap 2 kolom yang memiliki outlier, yaitu BMI dan charges.

Dari sini dapat diketahui kolom age tidak ada outlier dan steps

Menghapus beberapa outlier di dalam kolom bmi akan menjadi cara terbaik untuk menghapus noise dibanding dengan menghapus outlier di charges, karena charge sendiri memiliki variasi harga terngantung dari asuransi yang diberikan.

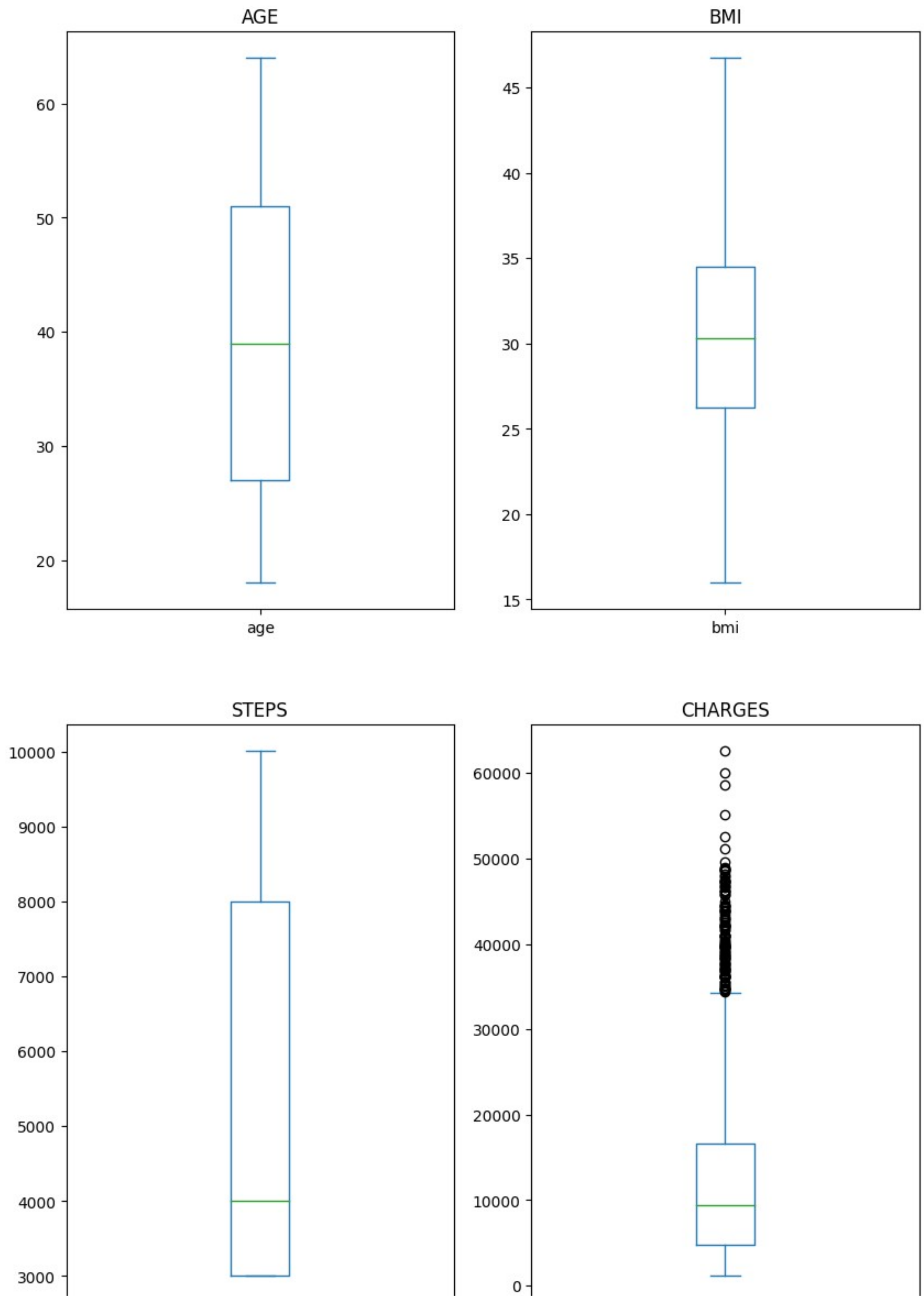
```
# checking how may outlier
Q1 = df['bmi'].quantile(0.25)
Q3 = df['bmi'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

x = (df['bmi'] >= Q1 - 1.5 * IQR) & (df['bmi'] <= Q3 + 1.5 *IQR)
df1 = df.copy()
df1 = df1.loc[x]
df1
```

```
      age  sex     bmi  steps  children  smoker  region
charges  \
0      19    0  27.900   3009         0       1       3  16884.92400

1      18    1  33.770   3008         1       0       2   1725.55230

2      28    1  33.000   3009         3       0       2   4449.46200

3      33    1  22.705  10009         0       0       1  21984.47061

4      32    1  28.880   8010         0       0       1   3866.85520

...   ...  ...     ...    ...       ...     ...     ...          ...

1333   50    1  30.970   4008         3       0       1  10600.54830

1334   18    0  31.920   3003         0       0       0   2205.98080

1335   18    0  36.850   3008         0       0       2   1629.83350

1336   21    0  25.800   8009         0       0       3   2007.94500

1337   61    0  29.070   8008         0       1       1  29141.36030


      insuranceclaim
0                  1
1                  1
2                  0
3                  0
```

```
4                  1
...               ...
1333               0
1334               1
1335               1
1336               0
1337               1

[1329 rows x 9 columns]
```

```
data = ["age","bmi","steps","charges"]
plt.figure(figsize=(10, 15))
plt.subplots_adjust(hspace=0.2)
plt.suptitle("box plot for numeric", fontsize=18)

for n,column in enumerate(data):
  ax = plt.subplot(2, 2, n + 1)
  df1[column].plot(kind='box')
  ax.set_title(column.upper())
  ax.set_xlabel("")
```

box plot for numeric



Untuk feature engeninering, yaitu scaler akan dilanjutkann pada saat split data

1B. [LO 3, LO 4, 5 poin] Lakukan eksplorasi data terlebih dahulu untuk memahami permasalahan yang dihadapi terlebih dahulu. Selanjutnya pisahkan dataset menjadi train, test dan validation set dengan ketentuan (80 train, 10 val, 10 test)

Pada soal ini saya menggunakan dataframe df untuk melakukan eksplorasi dan untuk memisahkan dataset akan menggunakan df1 yang tidak ada outlier.

```
df
```

```
      age  sex     bmi  steps  children  smoker  region
charges  \
0      19    0  27.900   3009         0       1       3  16884.92400

1      18    1  33.770   3008         1       0       2   1725.55230

2      28    1  33.000   3009         3       0       2   4449.46200

3      33    1  22.705  10009         0       0       1  21984.47061

4      32    1  28.880   8010         0       0       1   3866.85520

...   ...  ...     ...    ...       ...     ...     ...          ...

1333   50    1  30.970   4008         3       0       1  10600.54830

1334   18    0  31.920   3003         0       0       0   2205.98080

1335   18    0  36.850   3008         0       0       2   1629.83350

1336   21    0  25.800   8009         0       0       3   2007.94500

1337   61    0  29.070   8008         0       1       1  29141.36030


      insuranceclaim
0                  1
1                  1
2                  0
3                  0
4                  1
...              ...
1333               0
1334               1
1335               1
1336               0
1337               1
```

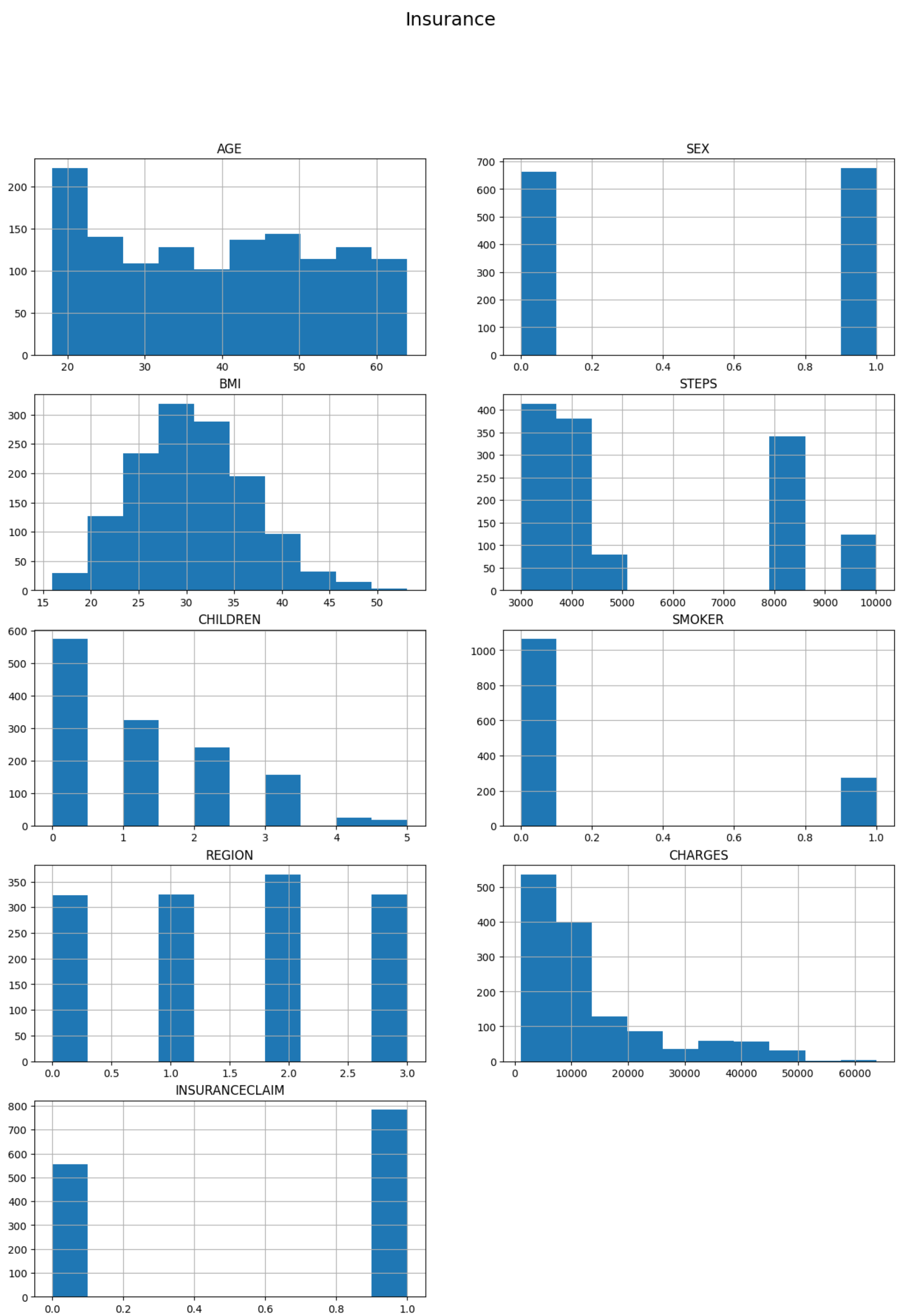
```
[1338 rows x 9 columns]
```

```
# check the dimmension
df.shape
```

```
(1338, 9)
```

Terdapat total data 1329 dengan 9 kolom

```
# histogram plot
plt.figure(figsize=(15, 20))
plt.subplots_adjust(hspace=0.2)
plt.suptitle("Insurance", fontsize=18)

for n,column in enumerate(df.columns):
  ax = plt.subplot(5, 2, n + 1)
  df[column].hist(ax=ax)
  ax.set_title(column.upper())
  ax.set_xlabel("")
```


Insurance
AGE
SEX
BMI
STEPS
CHILDREN
SMOKER
REGION
CHARGES
INSURANCECLAIM

Dari sini dapat diketahui yang mana merupakan categorical dan yang bukan dari penyebaran data yang diberikan. Dari sini juga sekilas dapat dilhat penyebaran data tidaklah berbentuk normalisasi.

```
df.corr()
```

```
                     age       sex       bmi     steps  children  
smoker  \
age             1.000000 -0.020856  0.109272 -0.167957  0.042469 -
0.025019
sex            -0.020856  1.000000  0.046371 -0.039470  0.017163  
0.076185
bmi             0.109272  0.046371  1.000000 -0.681149  0.012759  
0.003750
steps          -0.167957 -0.039470 -0.681149  1.000000  0.055346 -
0.267845
children        0.042469  0.017163  0.012759  0.055346  1.000000  
0.007673
smoker         -0.025019  0.076185  0.003750 -0.267845  0.007673  
1.000000
region          0.002127  0.004588  0.157566 -0.076483  0.016569 -
0.002181
charges         0.299008  0.057292  0.198341 -0.305570  0.067998  
0.787251
insuranceclaim  0.113723  0.031565  0.384198 -0.419514 -0.409526  
0.333261

                  region   charges  insuranceclaim  
age             0.002127  0.299008        0.113723  
sex             0.004588  0.057292        0.031565  
bmi             0.157566  0.198341        0.384198  
steps          -0.076483 -0.305570       -0.419514  
children        0.016569  0.067998       -0.409526  
smoker         -0.002181  0.787251        0.333261  
region          1.000000 -0.006208        0.020891  
charges        -0.006208  1.000000        0.309418  
insuranceclaim  0.020891  0.309418        1.000000  
```

Untuk lebih jelasnya mana yang besar dapat dilihat dengan menggunakan function di bawah berikut

```
individual_features_df = []
for i in range(0, len(df.columns) - 1):
    tmpDf = df[[df.columns[i], 'insuranceclaim']]
    tmpDf = tmpDf[tmpDf[df.columns[i]] != 0]
    individual_features_df.append(tmpDf)

all_correlations = {feature.columns[0]: feature.corr()
['insuranceclaim'][0] for feature in individual_features_df}
all_correlations = sorted(all_correlations.items(),
```

```
key=operator.itemgetter(1))
for (key, value) in all_correlations:
    print("{:>15}: {:>15}".format(key, value))
```

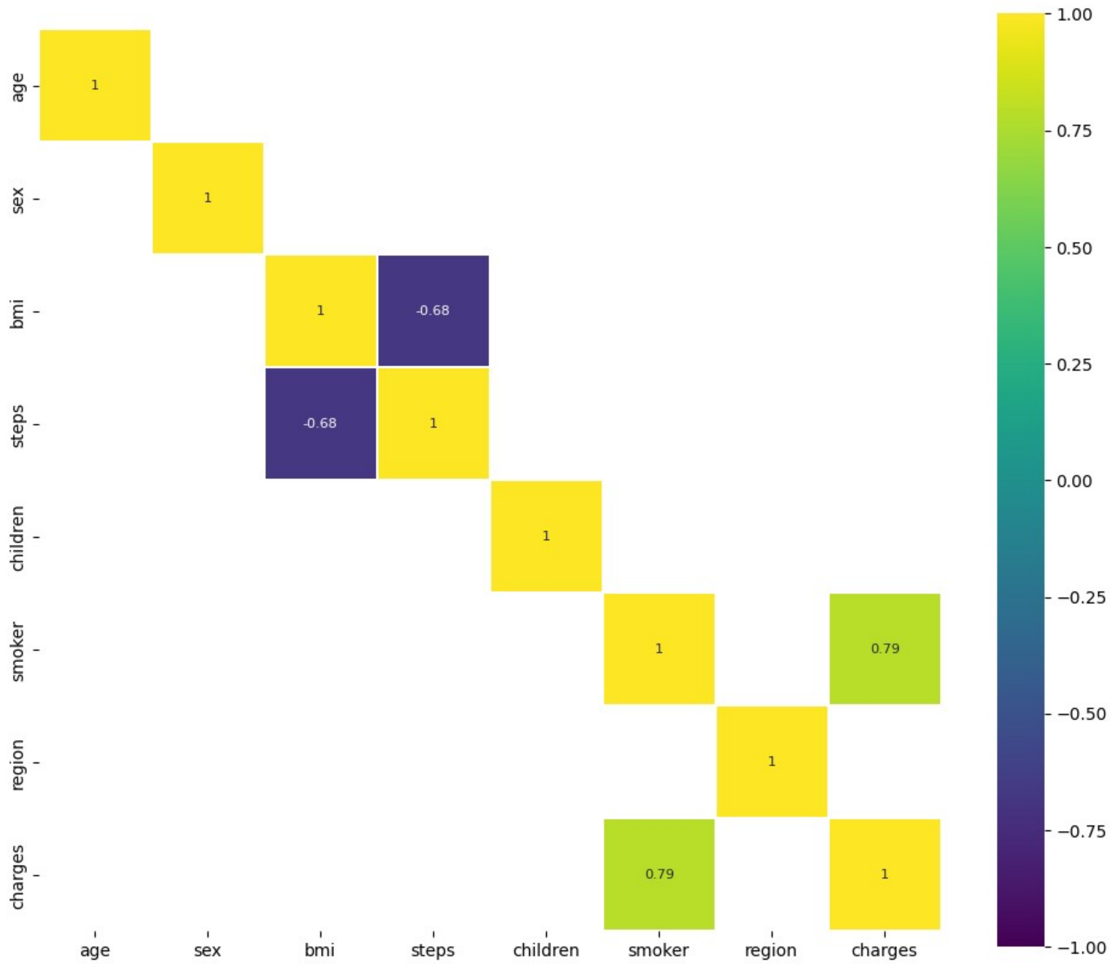
```
            age: 0.1137225624127475
            sex:             nan
          steps: -0.4195136142781132
       children: -0.22032635293122127
            bmi: 0.3841978190456
         smoker:             nan
         region: 0.049960683441651686
        charges: 0.30941820306029294
```

Dari sini dapat diketahui beberapa fitur yang memiliki korelasi terhadap insurance untuk yang tetinggi dapat dilihat dari fungsi dibawah.

```
golden_features_list = [key for key, value in all_correlations if
abs(value) >= 0.3]
print("Terdapat {} feature yang memiliki korelasi tinggi dengan
insurance claim:\n{}".format(len(golden_features_list),
golden_features_list))
```

```
Terdapat 3 feature yang memiliki korelasi tinggi dengan insurance
claim:
['steps', 'bmi', 'charges']
```

```
corr = df.drop('insuranceclaim', axis=1).corr()
plt.figure(figsize=(12, 10))

sns.heatmap(corr[(corr >= 0.4) | (corr <= -0.4)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size": 8}, square=True);
```



Dari korelasi fitur-fitur dapat diketahui hanya terdapat 2 relasi yang tinggi, yang mana pertama ada smoker dengan charges dan yang kedua ada steps dengan BMI. Dengan score, yaitu 0.79 dan -0.68.

Setelah melihat dan mengeksplor lebih dalam dataset ini. Setelah ini dataset akan dibagi menjadi train,validation, test.

```
#  Define X and Y.
# X is the feature to determine target (Y)
X = df1.drop(['insuranceclaim'],axis=1)
y = df1.insuranceclaim
```

Sebelum di bagi data akan di preprocessing lagi untuk mernomalisasi dari data X

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
X.shape
```

```
(1329, 8)
```

Setelah itu data akan dibagi menjadi ratio 80% train, 10% validation, dan 10% test

```
x_train, x_val_test, y_train, y_val_test =
train_test_split(X,y,test_size=0.2)
x_train.shape, x_val_test.shape, y_train.shape, y_val_test.shape
```

```
((1063, 8), (266, 8), (1063,), (266,))
```

Setelah di bagi 90% train dan 10% test. Train akan displit lagi untuk mendapatkan validasi

```
x_test, x_val, y_test, y_val = train_test_split(x_val_test,
y_val_test,test_size=0.5)
x_test.shape, x_val.shape, y_test.shape, y_val.shape
```

```
((133, 8), (133, 8), (133,), (133,))
```

Dengan begini data sudah terbagi mejadi bentuk yang diingankan

1C. [LO 3, LO 4, 5 poin] Buatlah arsitektur baseline dengan n nodes input layer, 2 buah hidden layer dengan banyak 2 × n nodes awal dan layer akhir banyak kelas nya (n, 2 × n, 2 × n, num_class). Keterangan: n adalah banyak input dan num_class adalah banyak kelas. Activation function untuk tiap hidden layer menggunakan ReLU

```
pip install git+https://github.com/tensorflow/docs
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-
9iz1_onc
  Running command git clone --filter=blob:none --quiet
https://github.com/tensorflow/docs /tmp/pip-req-build-9iz1_onc
  Resolved https://github.com/tensorflow/docs to commit
abfbe6e54864baa38dbb985b984acd304be610d4
  Preparing metadata (setup.py) ... ent already satisfied: absl-py
in /usr/local/lib/python3.10/dist-packages (from tensorflow-
docs==0.0.0.dev0) (1.4.0)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow-
docs==0.0.0.dev0) (3.1.2)
Requirement already satisfied: nbformat in
/usr/local/lib/python3.10/dist-packages (from tensorflow-
docs==0.0.0.dev0) (5.8.0)
Requirement already satisfied: protobuf>=3.12 in
/usr/local/lib/python3.10/dist-packages (from tensorflow-
docs==0.0.0.dev0) (3.20.3)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.10/dist-packages (from tensorflow-
docs==0.0.0.dev0) (6.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->tensorflow-
docs==0.0.0.dev0) (2.1.2)
Requirement already satisfied: traitlets>=5.1 in
/usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-
```

```
docs==0.0.0.dev0) (5.7.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-
docs==0.0.0.dev0) (4.3.3)
Requirement already satisfied: jupyter-core in
/usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-
docs==0.0.0.dev0) (5.3.0)
Requirement already satisfied: fastjsonschema in
/usr/local/lib/python3.10/dist-packages (from nbformat->tensorflow-
docs==0.0.0.dev0) (2.16.3)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!
=0.17.2,>=0.14.0 in /usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat->tensorflow-docs==0.0.0.dev0) (0.19.3)
Requirement already satisfied: attrs>=17.4.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->tensorflow-docs==0.0.0.dev0) (23.1.0)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core->nbformat-
>tensorflow-docs==0.0.0.dev0) (3.3.0)
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... e=tensorflow_docs-
0.0.0.dev0-py3-none-any.whl size=183273
sha256=101583b1689bf230d0dc601e507ad6cdd1c291124a9f90ac709c06bfc1111d2
5
  Stored in directory:
/tmp/pip-ephem-wheel-cache-n7rjgzxb/wheels/86/0f/1e/3b62293c8ffd0fd5a4
9508e6871cdb7554abe9c62afd35ec53
Successfully built tensorflow-docs
Installing collected packages: astor, tensorflow-docs
Successfully installed astor-0.8.1 tensorflow-docs-0.0.0.dev0
```

installing tensorflow docs

```
# Libary that used ini modeling

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_docs as tfdocs
import tensorflow_docs.modeling
from keras.utils.np_utils import to_categorical
```

Pada metode ini akan menggunakan library dari tensor, yaitu keras untuk membentuk ANN

```
y_train_onehot = to_categorical(y_train, num_classes=2)
y_val_OHE = to_categorical(y_val, num_classes=2)
y_test_OHE = to_categorical(y_test, num_classes=2)
```

Merubah kolom target menjadi One hot enconder agar dapat dibaca oleh loss function categorical crossentrophy.

```
x_train = np.asarray(x_train)
y_train_onehot = np.asarray(y_train_onehot)
```

Merubah ke array guna menhindari error.

```
# Initial the model
FirstModel = keras.Sequential()
```

Pada tahap ini model sudah dibentuk dengan nama model

```
# Add input layer, hidden layer, and output layer
# Hidden Layer 1
FirstModel.add(layers.Dense(16,input_shape=(8,), activation="relu"))
# Hidden Layer 2
FirstModel.add(layers.Dense(16,activation='relu'))
# Output layer
FirstModel.add(layers.Dense(2))
```

Dengan begitu terlalu terbuat model dengan susunan model (8,16,16,2)

```
FirstModel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                144

 dense_1 (Dense)             (None, 16)                272

 dense_2 (Dense)             (None, 2)                 34

=================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
_________________________________________________________________
```

Dengan melakukan summary dapat diketahui berapa layer telah dibuat dengan hidden layer

```
# Compile de model so have the output of accuracy
FirstModel.compile(loss = 'categorical_crossentropy',metrics =
['accuracy'])
```

Setelah itu dapat melakukan fitting model untuk mendapatkan hasil akurasi yang akan diberikan. Pada kasus ini akan menggunakan 40 epoch

```
epochs = 40

history = FirstModel.fit(
  x_train, y_train_onehot, validation_data= (x_val, y_val_OHE),
```

```
  epochs = epochs, verbose = 1,
  callbacks = [tfdocs.modeling.EpochDots()])
```

```
Epoch 1/40
25/34 [====================>........] - ETA: 0s - loss: 4.1904 -
accuracy: 0.4150
Epoch: 0, accuracy:0.4026,  loss:4.1092,  val_accuracy:0.3534,
val_loss:3.0870,
34/34 [==============================] - 2s 9ms/step - loss: 4.1092 -
accuracy: 0.4026 - val_loss: 3.0870 - val_accuracy: 0.3534
Epoch 2/40
34/34 [==============================] - 0s 3ms/step - loss: 3.2397 -
accuracy: 0.3866 - val_loss: 2.7225 - val_accuracy: 0.3308
Epoch 3/40
34/34 [==============================] - 0s 4ms/step - loss: 2.8029 -
accuracy: 0.3810 - val_loss: 2.2224 - val_accuracy: 0.3835
Epoch 4/40
34/34 [==============================] - 0s 5ms/step - loss: 2.3363 -
accuracy: 0.3678 - val_loss: 2.3179 - val_accuracy: 0.3910
Epoch 5/40
34/34 [==============================] - 0s 5ms/step - loss: 2.2565 -
accuracy: 0.3650 - val_loss: 1.7335 - val_accuracy: 0.3835
Epoch 6/40
34/34 [==============================] - 0s 4ms/step - loss: 1.9424 -
accuracy: 0.3622 - val_loss: 1.8213 - val_accuracy: 0.4060
Epoch 7/40
34/34 [==============================] - 0s 3ms/step - loss: 1.8193 -
accuracy: 0.3772 - val_loss: 2.0450 - val_accuracy: 0.3910
Epoch 8/40
34/34 [==============================] - 0s 3ms/step - loss: 1.6873 -
accuracy: 0.3829 - val_loss: 1.7155 - val_accuracy: 0.3684
Epoch 9/40
34/34 [==============================] - 0s 4ms/step - loss: 1.7674 -
accuracy: 0.3678 - val_loss: 1.7700 - val_accuracy: 0.3534
Epoch 10/40
34/34 [==============================] - 0s 4ms/step - loss: 1.7131 -
accuracy: 0.3650 - val_loss: 1.3242 - val_accuracy: 0.3534
Epoch 11/40
34/34 [==============================] - 0s 4ms/step - loss: 1.6625 -
accuracy: 0.3688 - val_loss: 1.5445 - val_accuracy: 0.3534
Epoch 12/40
34/34 [==============================] - 0s 4ms/step - loss: 1.5450 -
accuracy: 0.3697 - val_loss: 2.2308 - val_accuracy: 0.3609
Epoch 13/40
34/34 [==============================] - 0s 6ms/step - loss: 1.6126 -
accuracy: 0.3678 - val_loss: 1.8682 - val_accuracy: 0.3684
Epoch 14/40
34/34 [==============================] - 0s 5ms/step - loss: 1.8212 -
accuracy: 0.3782 - val_loss: 1.8883 - val_accuracy: 0.3308
Epoch 15/40
```

```
34/34 [==============================] - 0s 6ms/step - loss: 1.6295 -
accuracy: 0.3584 - val_loss: 2.6767 - val_accuracy: 0.3684
Epoch 16/40
34/34 [==============================] - 0s 13ms/step - loss: 1.6057 -
accuracy: 0.3735 - val_loss: 1.7881 - val_accuracy: 0.3759
Epoch 17/40
34/34 [==============================] - 0s 9ms/step - loss: 1.4265 -
accuracy: 0.3688 - val_loss: 1.3451 - val_accuracy: 0.3684
Epoch 18/40
34/34 [==============================] - 0s 9ms/step - loss: 1.3294 -
accuracy: 0.3509 - val_loss: 1.3442 - val_accuracy: 0.3383
Epoch 19/40
34/34 [==============================] - 0s 7ms/step - loss: 1.1415 -
accuracy: 0.3340 - val_loss: 1.1945 - val_accuracy: 0.3609
Epoch 20/40
34/34 [==============================] - 0s 8ms/step - loss: 1.2463 -
accuracy: 0.3283 - val_loss: 1.3038 - val_accuracy: 0.3459
Epoch 21/40
34/34 [==============================] - 0s 9ms/step - loss: 1.3505 -
accuracy: 0.3452 - val_loss: 1.2239 - val_accuracy: 0.3534
Epoch 22/40
34/34 [==============================] - 0s 10ms/step - loss: 1.2502 -
accuracy: 0.3293 - val_loss: 1.1922 - val_accuracy: 0.3383
Epoch 23/40
34/34 [==============================] - 0s 14ms/step - loss: 1.1327 -
accuracy: 0.3349 - val_loss: 1.1836 - val_accuracy: 0.3158
Epoch 24/40
34/34 [==============================] - 0s 14ms/step - loss: 1.1746 -
accuracy: 0.3246 - val_loss: 1.0671 - val_accuracy: 0.3609
Epoch 25/40
34/34 [==============================] - 0s 12ms/step - loss: 1.0573 -
accuracy: 0.3424 - val_loss: 1.0924 - val_accuracy: 0.3534
Epoch 26/40
34/34 [==============================] - 0s 9ms/step - loss: 1.3565 -
accuracy: 0.3481 - val_loss: 0.6699 - val_accuracy: 0.3459
Epoch 27/40
34/34 [==============================] - 0s 11ms/step - loss: 1.1407 -
accuracy: 0.3462 - val_loss: 1.0387 - val_accuracy: 0.3308
Epoch 28/40
34/34 [==============================] - 0s 9ms/step - loss: 1.0536 -
accuracy: 0.3452 - val_loss: 1.0390 - val_accuracy: 0.3383
Epoch 29/40
34/34 [==============================] - 0s 8ms/step - loss: 1.1629 -
accuracy: 0.3311 - val_loss: 0.9252 - val_accuracy: 0.3759
Epoch 30/40
34/34 [==============================] - 0s 7ms/step - loss: 1.0968 -
accuracy: 0.3405 - val_loss: 0.6662 - val_accuracy: 0.3158
Epoch 31/40
34/34 [==============================] - 0s 6ms/step - loss: 1.1224 -
accuracy: 0.3283 - val_loss: 1.0390 - val_accuracy: 0.3383
```

```
Epoch 32/40
34/34 [==============================] - 0s 7ms/step - loss: 1.2232 -
accuracy: 0.3311 - val_loss: 1.1682 - val_accuracy: 0.3459
Epoch 33/40
34/34 [==============================] - 0s 5ms/step - loss: 1.2710 -
accuracy: 0.3330 - val_loss: 1.1894 - val_accuracy: 0.3008
Epoch 34/40
34/34 [==============================] - 0s 7ms/step - loss: 1.2650 -
accuracy: 0.3067 - val_loss: 1.4017 - val_accuracy: 0.3308
Epoch 35/40
34/34 [==============================] - 0s 8ms/step - loss: 1.3909 -
accuracy: 0.3104 - val_loss: 1.4847 - val_accuracy: 0.3158
Epoch 36/40
34/34 [==============================] - 0s 6ms/step - loss: 1.2842 -
accuracy: 0.3057 - val_loss: 1.2744 - val_accuracy: 0.3308
Epoch 37/40
34/34 [==============================] - 0s 6ms/step - loss: 1.3462 -
accuracy: 0.3246 - val_loss: 1.2773 - val_accuracy: 0.3534
Epoch 38/40
34/34 [==============================] - 0s 7ms/step - loss: 1.1804 -
accuracy: 0.3518 - val_loss: 1.4760 - val_accuracy: 0.3684
Epoch 39/40
34/34 [==============================] - 0s 6ms/step - loss: 1.2507 -
accuracy: 0.3490 - val_loss: 1.7299 - val_accuracy: 0.3459
Epoch 40/40
34/34 [==============================] - 0s 5ms/step - loss: 1.3081 -
accuracy: 0.3518 - val_loss: 1.2033 - val_accuracy: 0.3910
```

Dari sini dengan menggunakan epochs 100 dan dengan batch size di dapat hasil akurasi yang didapatkan 45% dan val akurasi 49%.

```
FirstModel.evaluate(x_test, y_test_OHE)[1]
```

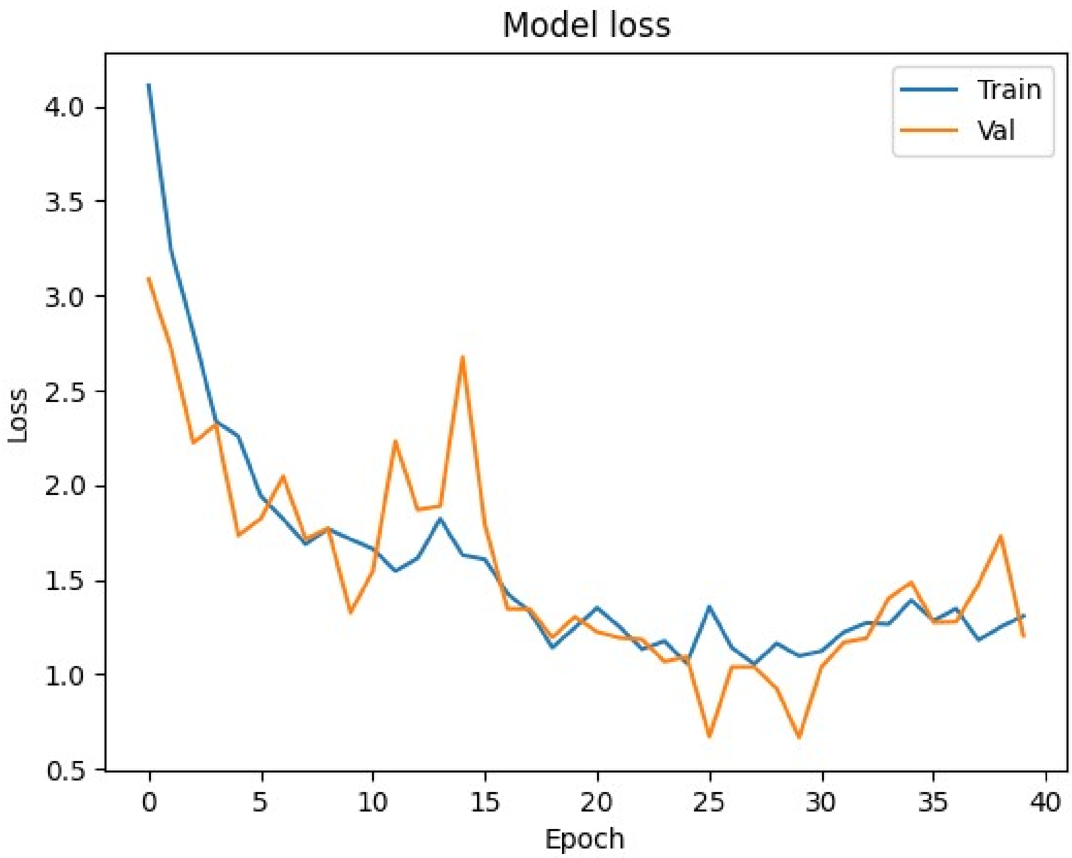
```
5/5 [==============================] - 0s 5ms/step - loss: 1.7342 -
accuracy: 0.4135
```

```
0.4135338366031647
```

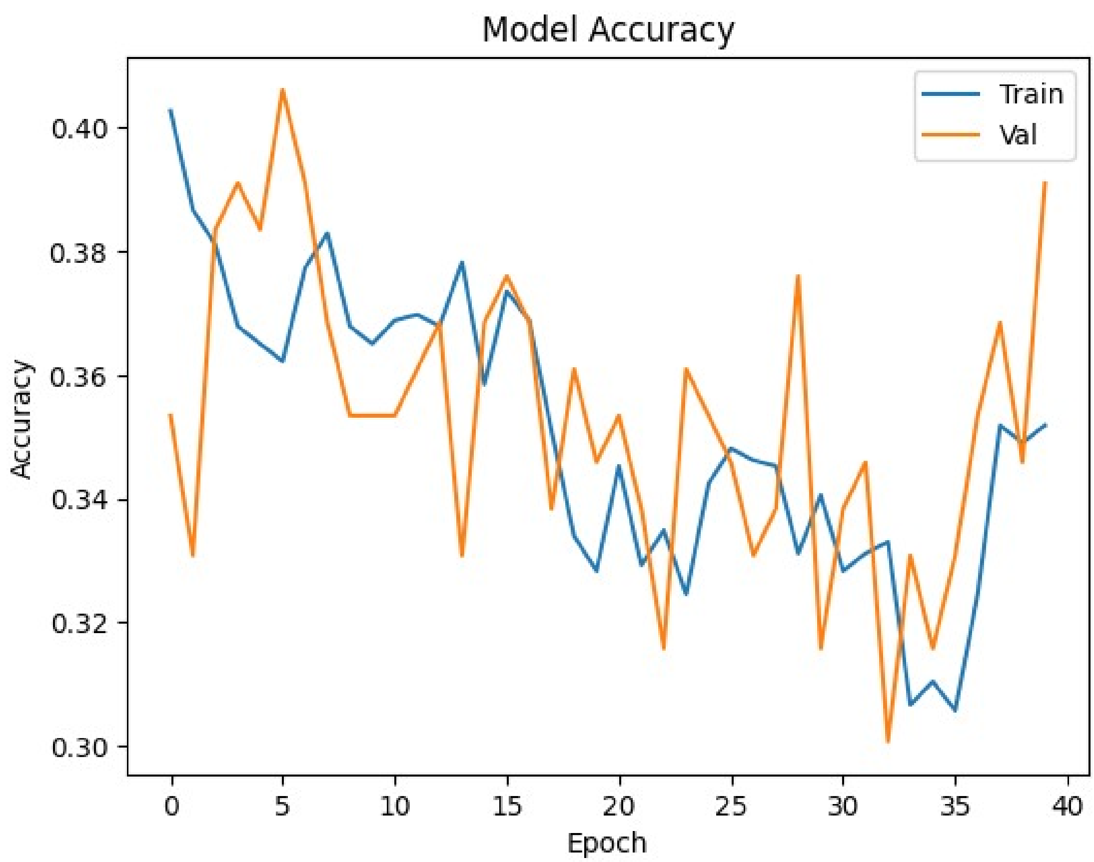
Hasil yang didapatkan dengan melakukan evaluasi dengan test model didapatkan hasil 0.47%

Untuk melihat lebih jelas kembali dapat melihat dari plot antar train dengan validation

```
# plotting model loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
# plotting accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Dari model yang dipatkan dapat diketahui pembentukan plot yang dihasilkan sangatlah overfitting dapat terlihat pada plot model loss dan model accuracy.

Selain itu akurasi yang dibentuk 35% dan untuk validasi akurasinya terdapat di 45%. Sedangkan pada saat dibandingkan dengan test mendapatkan hasil 41%.

1D. [LO 3, LO 4, 5 poin] Lakukan evaluasi unjuk kerja kedua arsitektur di atas pada test set dengan mencari nilai accuracy, precision, recall dan F1-Score. Dan berikan penjelasan mengenai hasilnya dengan rinci.

Pada nomor ini akan terdapat beberapa model yang dibuat guna untuk menbandingkan yang mana yang paling efektif

```
Model_1A = keras.Sequential()
```

Inisialisaikan model 1A dan akan ditambahkan arsitektur

```
# Add input layer, hidden layer, and output layer
# Hidden Layer 1
Model_1A.add(layers.Dense(16,input_shape=(8,), activation="relu"))
# Hidden Layer 2
Model_1A.add(layers.Dense(16,activation='relu'))
# Output layer
Model_1A.add(layers.Dense(2, activation='sigmoid'))
```

Pada pemodelan ini model yang digunakan terdapat penambahan activation fuction pada output, yaitu sigmoid yang berfungsi dengan baik dalam classification model. Selain itu tidak ada penambahan atau pengurangan layer pada model 1.

```
Model_1A.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_3 (Dense)             (None, 16)                144       
                                                                 
 dense_4 (Dense)             (None, 16)                272       
                                                                 
 dense_5 (Dense)             (None, 2)                 34        
                                                                 
=================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
_________________________________________________________________
```

Untuk model petama ini akan menggunakan optimzer adam dengan learning rate 0.01

```
# compiling the model
Model_1A.compile(loss = 
'categorical_crossentropy',optimizer=tf.optimizers.Adam(learning_rate=
0.01),metrics = ['accuracy'])

epochs = 100

history1 = Model_1A.fit(
  x_train, y_train_onehot, validation_data= (x_val, y_val_OHE),
  epochs = epochs, verbose = 1, batch_size=30,
  callbacks = [tfdocs.modeling.EpochDots()])
```

```
Epoch 1/100
22/36 [=================>............] - ETA: 0s - loss: 0.4775 - 
accuracy: 0.7848
Epoch: 0, accuracy:0.8128,  loss:0.4350,  val_accuracy:0.8872,  
val_loss:0.2684,  
36/36 [==============================] - 2s 15ms/step - loss: 0.4350 - 
accuracy: 0.8128 - val_loss: 0.2684 - val_accuracy: 0.8872
Epoch 2/100
36/36 [==============================] - 0s 6ms/step - loss: 0.2940 - 
accuracy: 0.8702 - val_loss: 0.2551 - val_accuracy: 0.9023
Epoch 3/100
36/36 [==============================] - 0s 8ms/step - loss: 0.2516 - 
accuracy: 0.8965 - val_loss: 0.2365 - val_accuracy: 0.8872
Epoch 4/100
36/36 [==============================] - 0s 7ms/step - loss: 0.2329 - 
```

```
accuracy: 0.9050 - val_loss: 0.2306 - val_accuracy: 0.9098
Epoch 5/100
36/36 [==============================] - 0s 12ms/step - loss: 0.2111 -
accuracy: 0.9172 - val_loss: 0.1857 - val_accuracy: 0.9173
Epoch 6/100
36/36 [==============================] - 0s 6ms/step - loss: 0.1939 -
accuracy: 0.9163 - val_loss: 0.1660 - val_accuracy: 0.9248
Epoch 7/100
36/36 [==============================] - 0s 7ms/step - loss: 0.1912 -
accuracy: 0.9229 - val_loss: 0.1915 - val_accuracy: 0.9398
Epoch 8/100
36/36 [==============================] - 0s 8ms/step - loss: 0.1877 -
accuracy: 0.9238 - val_loss: 0.1832 - val_accuracy: 0.9248
Epoch 9/100
36/36 [==============================] - 0s 6ms/step - loss: 0.1776 -
accuracy: 0.9210 - val_loss: 0.1611 - val_accuracy: 0.9098
Epoch 10/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1636 -
accuracy: 0.9304 - val_loss: 0.1800 - val_accuracy: 0.9173
Epoch 11/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1469 -
accuracy: 0.9276 - val_loss: 0.2039 - val_accuracy: 0.8947
Epoch 12/100
36/36 [==============================] - 0s 10ms/step - loss: 0.1545 -
accuracy: 0.9266 - val_loss: 0.1799 - val_accuracy: 0.9023
Epoch 13/100
36/36 [==============================] - 0s 8ms/step - loss: 0.1373 -
accuracy: 0.9370 - val_loss: 0.1526 - val_accuracy: 0.9398
Epoch 14/100
36/36 [==============================] - 0s 8ms/step - loss: 0.1325 -
accuracy: 0.9379 - val_loss: 0.1135 - val_accuracy: 0.9248
Epoch 15/100
36/36 [==============================] - 0s 9ms/step - loss: 0.1340 -
accuracy: 0.9370 - val_loss: 0.1582 - val_accuracy: 0.9323
Epoch 16/100
36/36 [==============================] - 0s 11ms/step - loss: 0.1332 -
accuracy: 0.9351 - val_loss: 0.1322 - val_accuracy: 0.9248
Epoch 17/100
36/36 [==============================] - 0s 12ms/step - loss: 0.1365 -
accuracy: 0.9445 - val_loss: 0.1164 - val_accuracy: 0.9398
Epoch 18/100
36/36 [==============================] - 1s 15ms/step - loss: 0.1120 -
accuracy: 0.9464 - val_loss: 0.1430 - val_accuracy: 0.9549
Epoch 19/100
36/36 [==============================] - 0s 13ms/step - loss: 0.1043 -
accuracy: 0.9520 - val_loss: 0.1858 - val_accuracy: 0.9023
Epoch 20/100
36/36 [==============================] - 0s 11ms/step - loss: 0.1058 -
accuracy: 0.9567 - val_loss: 0.1112 - val_accuracy: 0.9398
Epoch 21/100
```

```
36/36 [==============================] - 0s 12ms/step - loss: 0.0998 -
accuracy: 0.9530 - val_loss: 0.0946 - val_accuracy: 0.9549
Epoch 22/100
36/36 [==============================] - 0s 13ms/step - loss: 0.0975 -
accuracy: 0.9605 - val_loss: 0.1345 - val_accuracy: 0.9323
Epoch 23/100
36/36 [==============================] - 0s 11ms/step - loss: 0.0913 -
accuracy: 0.9605 - val_loss: 0.1162 - val_accuracy: 0.9248
Epoch 24/100
36/36 [==============================] - 0s 13ms/step - loss: 0.0907 -
accuracy: 0.9643 - val_loss: 0.1021 - val_accuracy: 0.9624
Epoch 25/100
36/36 [==============================] - 1s 14ms/step - loss: 0.0692 -
accuracy: 0.9765 - val_loss: 0.0920 - val_accuracy: 0.9549
Epoch 26/100
36/36 [==============================] - 0s 12ms/step - loss: 0.0861 -
accuracy: 0.9586 - val_loss: 0.0634 - val_accuracy: 0.9699
Epoch 27/100
36/36 [==============================] - 0s 8ms/step - loss: 0.0878 -
accuracy: 0.9605 - val_loss: 0.1130 - val_accuracy: 0.9323
Epoch 28/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0744 -
accuracy: 0.9680 - val_loss: 0.0731 - val_accuracy: 0.9699
Epoch 29/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0768 -
accuracy: 0.9652 - val_loss: 0.0902 - val_accuracy: 0.9474
Epoch 30/100
36/36 [==============================] - 0s 7ms/step - loss: 0.0838 -
accuracy: 0.9577 - val_loss: 0.1368 - val_accuracy: 0.9549
Epoch 31/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0992 -
accuracy: 0.9614 - val_loss: 0.1099 - val_accuracy: 0.9624
Epoch 32/100
36/36 [==============================] - 0s 8ms/step - loss: 0.0683 -
accuracy: 0.9699 - val_loss: 0.0668 - val_accuracy: 0.9624
Epoch 33/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0560 -
accuracy: 0.9802 - val_loss: 0.0971 - val_accuracy: 0.9549
Epoch 34/100
36/36 [==============================] - 0s 8ms/step - loss: 0.0691 -
accuracy: 0.9699 - val_loss: 0.1416 - val_accuracy: 0.9549
Epoch 35/100
36/36 [==============================] - 0s 11ms/step - loss: 0.0665 -
accuracy: 0.9727 - val_loss: 0.0892 - val_accuracy: 0.9624
Epoch 36/100
36/36 [==============================] - 0s 9ms/step - loss: 0.0611 -
accuracy: 0.9746 - val_loss: 0.1992 - val_accuracy: 0.9248
Epoch 37/100
36/36 [==============================] - 0s 8ms/step - loss: 0.0863 -
accuracy: 0.9643 - val_loss: 0.0988 - val_accuracy: 0.9624
```

```
Epoch 38/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0804 -
accuracy: 0.9652 - val_loss: 0.0869 - val_accuracy: 0.9549
Epoch 39/100
36/36 [==============================] - 0s 9ms/step - loss: 0.0575 -
accuracy: 0.9718 - val_loss: 0.0767 - val_accuracy: 0.9624
Epoch 40/100
36/36 [==============================] - 0s 9ms/step - loss: 0.0515 -
accuracy: 0.9793 - val_loss: 0.0576 - val_accuracy: 0.9850
Epoch 41/100
36/36 [==============================] - 0s 7ms/step - loss: 0.0528 -
accuracy: 0.9774 - val_loss: 0.1175 - val_accuracy: 0.9474
Epoch 42/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0824 -
accuracy: 0.9652 - val_loss: 0.1488 - val_accuracy: 0.9474
Epoch 43/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0733 -
accuracy: 0.9737 - val_loss: 0.0790 - val_accuracy: 0.9624
Epoch 44/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0488 -
accuracy: 0.9821 - val_loss: 0.1148 - val_accuracy: 0.9624
Epoch 45/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0431 -
accuracy: 0.9831 - val_loss: 0.1192 - val_accuracy: 0.9549
Epoch 46/100
36/36 [==============================] - 0s 9ms/step - loss: 0.0536 -
accuracy: 0.9765 - val_loss: 0.0973 - val_accuracy: 0.9549
Epoch 47/100
36/36 [==============================] - 0s 10ms/step - loss: 0.0584 -
accuracy: 0.9784 - val_loss: 0.1035 - val_accuracy: 0.9549
Epoch 48/100
36/36 [==============================] - 0s 13ms/step - loss: 0.0601 -
accuracy: 0.9793 - val_loss: 0.0888 - val_accuracy: 0.9549
Epoch 49/100
36/36 [==============================] - 0s 10ms/step - loss: 0.0484 -
accuracy: 0.9746 - val_loss: 0.1024 - val_accuracy: 0.9549
Epoch 50/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0375 -
accuracy: 0.9831 - val_loss: 0.1108 - val_accuracy: 0.9624
Epoch 51/100
36/36 [==============================] - 0s 9ms/step - loss: 0.0327 -
accuracy: 0.9897 - val_loss: 0.0822 - val_accuracy: 0.9624
Epoch 52/100
36/36 [==============================] - 0s 9ms/step - loss: 0.0424 -
accuracy: 0.9802 - val_loss: 0.0954 - val_accuracy: 0.9549
Epoch 53/100
36/36 [==============================] - 0s 7ms/step - loss: 0.0471 -
accuracy: 0.9765 - val_loss: 0.1237 - val_accuracy: 0.9549
Epoch 54/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0432 -
```

```
accuracy: 0.9831 - val_loss: 0.1124 - val_accuracy: 0.9624
Epoch 55/100
36/36 [==============================] - 0s 10ms/step - loss: 0.0549 -
accuracy: 0.9774 - val_loss: 0.0679 - val_accuracy: 0.9699
Epoch 56/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0518 -
accuracy: 0.9793 - val_loss: 0.1234 - val_accuracy: 0.9474
Epoch 57/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0780 -
accuracy: 0.9727 - val_loss: 0.1898 - val_accuracy: 0.9398
Epoch 58/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0783 -
accuracy: 0.9727 - val_loss: 0.0615 - val_accuracy: 0.9699
Epoch 59/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0497 -
accuracy: 0.9802 - val_loss: 0.1243 - val_accuracy: 0.9474
Epoch 60/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0358 -
accuracy: 0.9859 - val_loss: 0.0735 - val_accuracy: 0.9549
Epoch 61/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0346 -
accuracy: 0.9878 - val_loss: 0.0718 - val_accuracy: 0.9624
Epoch 62/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0336 -
accuracy: 0.9849 - val_loss: 0.1143 - val_accuracy: 0.9549
Epoch 63/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0293 -
accuracy: 0.9915 - val_loss: 0.1318 - val_accuracy: 0.9549
Epoch 64/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0328 -
accuracy: 0.9878 - val_loss: 0.1464 - val_accuracy: 0.9474
Epoch 65/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0367 -
accuracy: 0.9868 - val_loss: 0.1274 - val_accuracy: 0.9549
Epoch 66/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0458 -
accuracy: 0.9812 - val_loss: 0.1103 - val_accuracy: 0.9624
Epoch 67/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0380 -
accuracy: 0.9840 - val_loss: 0.0691 - val_accuracy: 0.9624
Epoch 68/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0360 -
accuracy: 0.9849 - val_loss: 0.0716 - val_accuracy: 0.9624
Epoch 69/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0359 -
accuracy: 0.9859 - val_loss: 0.1420 - val_accuracy: 0.9549
Epoch 70/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0403 -
accuracy: 0.9868 - val_loss: 0.0573 - val_accuracy: 0.9699
Epoch 71/100
```

```
36/36 [==============================] - 0s 4ms/step - loss: 0.0414 -
accuracy: 0.9831 - val_loss: 0.1129 - val_accuracy: 0.9549
Epoch 72/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0322 -
accuracy: 0.9897 - val_loss: 0.1220 - val_accuracy: 0.9474
Epoch 73/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0315 -
accuracy: 0.9878 - val_loss: 0.1296 - val_accuracy: 0.9699
Epoch 74/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0279 -
accuracy: 0.9897 - val_loss: 0.1362 - val_accuracy: 0.9474
Epoch 75/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0425 -
accuracy: 0.9859 - val_loss: 0.0559 - val_accuracy: 0.9774
Epoch 76/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0460 -
accuracy: 0.9802 - val_loss: 0.0702 - val_accuracy: 0.9774
Epoch 77/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0395 -
accuracy: 0.9849 - val_loss: 0.1027 - val_accuracy: 0.9774
Epoch 78/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0511 -
accuracy: 0.9774 - val_loss: 0.1010 - val_accuracy: 0.9474
Epoch 79/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0676 -
accuracy: 0.9774 - val_loss: 0.1210 - val_accuracy: 0.9549
Epoch 80/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0491 -
accuracy: 0.9812 - val_loss: 0.0838 - val_accuracy: 0.9549
Epoch 81/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0378 -
accuracy: 0.9849 - val_loss: 0.1176 - val_accuracy: 0.9699
Epoch 82/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0726 -
accuracy: 0.9774 - val_loss: 0.1059 - val_accuracy: 0.9699
Epoch 83/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1647 -
accuracy: 0.9558 - val_loss: 0.1678 - val_accuracy: 0.9248
Epoch 84/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0697 -
accuracy: 0.9690 - val_loss: 0.0996 - val_accuracy: 0.9624
Epoch 85/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0471 -
accuracy: 0.9802 - val_loss: 0.0732 - val_accuracy: 0.9699
Epoch 86/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0338 -
accuracy: 0.9897 - val_loss: 0.0905 - val_accuracy: 0.9699
Epoch 87/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0299 -
accuracy: 0.9868 - val_loss: 0.0941 - val_accuracy: 0.9549
```

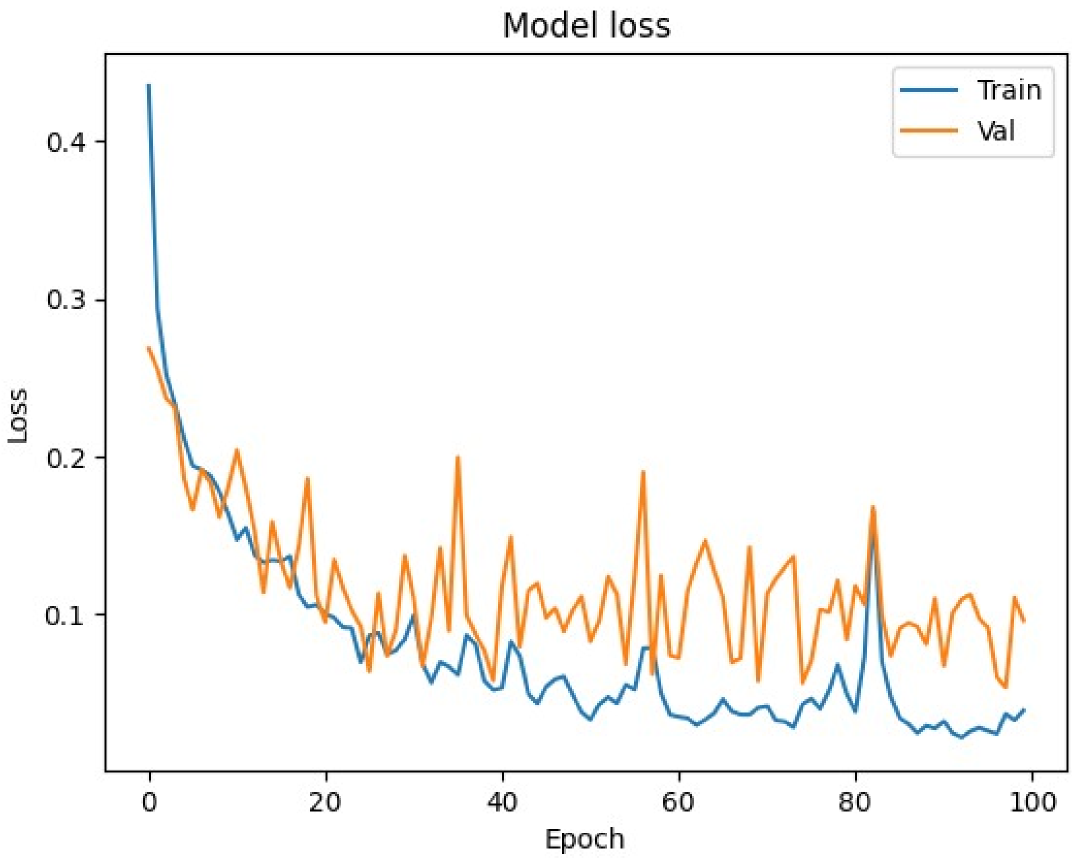
```
Epoch 88/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0242 -
accuracy: 0.9925 - val_loss: 0.0917 - val_accuracy: 0.9624
Epoch 89/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0289 -
accuracy: 0.9906 - val_loss: 0.0807 - val_accuracy: 0.9699
Epoch 90/100
36/36 [==============================] - 0s 6ms/step - loss: 0.0272 -
accuracy: 0.9906 - val_loss: 0.1096 - val_accuracy: 0.9549
Epoch 91/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0317 -
accuracy: 0.9868 - val_loss: 0.0669 - val_accuracy: 0.9699
Epoch 92/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0243 -
accuracy: 0.9897 - val_loss: 0.1003 - val_accuracy: 0.9699
Epoch 93/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0213 -
accuracy: 0.9934 - val_loss: 0.1091 - val_accuracy: 0.9699
Epoch 94/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0254 -
accuracy: 0.9906 - val_loss: 0.1122 - val_accuracy: 0.9699
Epoch 95/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0277 -
accuracy: 0.9906 - val_loss: 0.0970 - val_accuracy: 0.9774
Epoch 96/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0257 -
accuracy: 0.9915 - val_loss: 0.0910 - val_accuracy: 0.9699
Epoch 97/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0238 -
accuracy: 0.9925 - val_loss: 0.0598 - val_accuracy: 0.9774
Epoch 98/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0364 -
accuracy: 0.9887 - val_loss: 0.0530 - val_accuracy: 0.9699
Epoch 99/100
36/36 [==============================] - 0s 5ms/step - loss: 0.0323 -
accuracy: 0.9887 - val_loss: 0.1102 - val_accuracy: 0.9624
Epoch 100/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0385 -
accuracy: 0.9868 - val_loss: 0.0958 - val_accuracy: 0.9624
```

```
Model_1A.evaluate(x_test, y_test_OHE)[1]
```

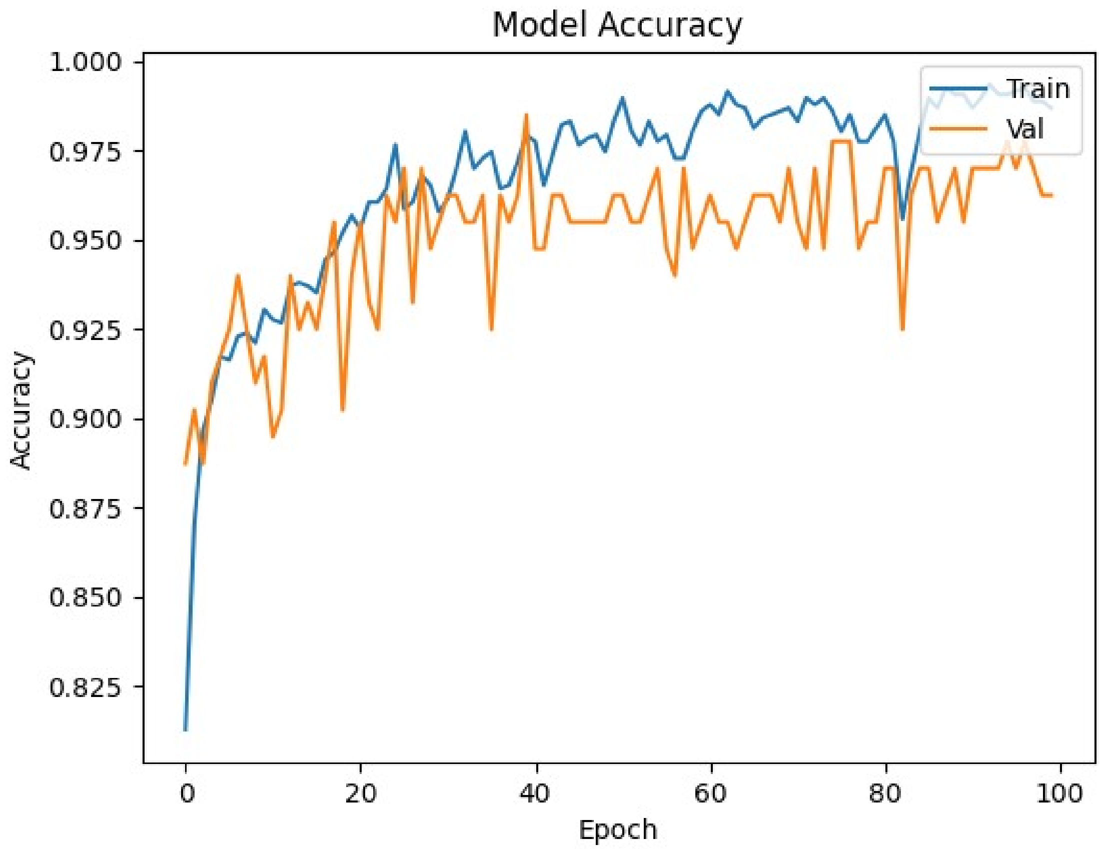
```
5/5 [==============================] - 0s 3ms/step - loss: 0.1771 -
accuracy: 0.9549

0.9548872113227844
```

```
# plotting model loss
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Model loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
# plotting accuracy
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Dari sini model 1A dengan menggunakan Adam optimizer mendapatkan hasil yang sangat baik terlihat dari akurasi yang diberikan. Hal ini dapat dilihat dari akurasi model dapat nilai diatas 99% dan evaluasi terdapat di nilai diatas 96%. Pada saat dibandingkan dengan test akurasi yang didapatkan 94%.

Plot yang dibentuk baik itu dari model loss dan akurasi membentukkan spike di beberapa tenpat. Hal ini yang menentukan suatu model overfitting. Sehingga diperlukan revisi dalam tuning.

Membentuk model 1B yang mana akan menganti model yang dibuat dengan menambahkan learning rate, karena dengan menurunkan learning rate dapat membantu menurunkan overfitting.

```
Model_1B = keras.Sequential()

# Add input layer, hidden layer, and output layer
# Hidden Layer 1
Model_1B.add(layers.Dense(16,input_shape=(8,), activation="relu"))
# Hidden Layer 2
Model_1B.add(layers.Dense(16,activation='relu'))
# Output layer
Model_1B.add(layers.Dense(2, activation='sigmoid'))

Model_1B.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 16)                144

 dense_7 (Dense)             (None, 16)                272

 dense_8 (Dense)             (None, 2)                 34

=================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
_________________________________________________________________
```

```
# compiling the model
Model_1B.compile(loss =
'categorical_crossentropy',optimizer=tf.optimizers.Adam(learning_rate=
0.001),metrics = ['accuracy'])

epochs = 100

history2 = Model_1B.fit(
  x_train, y_train_onehot, validation_data= (x_val, y_val_OHE),
  epochs = epochs, verbose = 1, batch_size=30,
  callbacks = [tfdocs.modeling.EpochDots()])
```

```
Epoch 1/100
32/36 [=========================>....] - ETA: 0s - loss: 0.6129 -
accuracy: 0.6812
Epoch: 0, accuracy:0.6858,  loss:0.6081,  val_accuracy:0.7970,
val_loss:0.5465,
36/36 [==============================] - 1s 11ms/step - loss: 0.6081 -
accuracy: 0.6858 - val_loss: 0.5465 - val_accuracy: 0.7970
Epoch 2/100
36/36 [==============================] - 0s 4ms/step - loss: 0.5280 -
accuracy: 0.7648 - val_loss: 0.4781 - val_accuracy: 0.8045
Epoch 3/100
36/36 [==============================] - 0s 5ms/step - loss: 0.4718 -
accuracy: 0.7968 - val_loss: 0.4224 - val_accuracy: 0.8421
Epoch 4/100
36/36 [==============================] - 0s 4ms/step - loss: 0.4326 -
accuracy: 0.8184 - val_loss: 0.3770 - val_accuracy: 0.8421
Epoch 5/100
36/36 [==============================] - 0s 3ms/step - loss: 0.4033 -
accuracy: 0.8297 - val_loss: 0.3435 - val_accuracy: 0.8571
Epoch 6/100
36/36 [==============================] - 0s 4ms/step - loss: 0.3818 -
accuracy: 0.8542 - val_loss: 0.3165 - val_accuracy: 0.8722
Epoch 7/100
```

```
36/36 [==============================] - 0s 4ms/step - loss: 0.3617 -
accuracy: 0.8561 - val_loss: 0.2937 - val_accuracy: 0.8722
Epoch 8/100
36/36 [==============================] - 0s 4ms/step - loss: 0.3443 -
accuracy: 0.8683 - val_loss: 0.2771 - val_accuracy: 0.8872
Epoch 9/100
36/36 [==============================] - 0s 5ms/step - loss: 0.3287 -
accuracy: 0.8702 - val_loss: 0.2615 - val_accuracy: 0.9173
Epoch 10/100
36/36 [==============================] - 0s 4ms/step - loss: 0.3135 -
accuracy: 0.8777 - val_loss: 0.2483 - val_accuracy: 0.9248
Epoch 11/100
36/36 [==============================] - 0s 4ms/step - loss: 0.3008 -
accuracy: 0.8786 - val_loss: 0.2400 - val_accuracy: 0.9248
Epoch 12/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2887 -
accuracy: 0.8918 - val_loss: 0.2331 - val_accuracy: 0.9323
Epoch 13/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2778 -
accuracy: 0.8871 - val_loss: 0.2234 - val_accuracy: 0.9398
Epoch 14/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2666 -
accuracy: 0.8946 - val_loss: 0.2130 - val_accuracy: 0.9398
Epoch 15/100
36/36 [==============================] - 0s 5ms/step - loss: 0.2591 -
accuracy: 0.8928 - val_loss: 0.2060 - val_accuracy: 0.9398
Epoch 16/100
36/36 [==============================] - 0s 5ms/step - loss: 0.2480 -
accuracy: 0.9012 - val_loss: 0.1995 - val_accuracy: 0.9398
Epoch 17/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2398 -
accuracy: 0.9087 - val_loss: 0.1949 - val_accuracy: 0.9398
Epoch 18/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2330 -
accuracy: 0.9059 - val_loss: 0.1905 - val_accuracy: 0.9549
Epoch 19/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2264 -
accuracy: 0.9116 - val_loss: 0.1927 - val_accuracy: 0.9248
Epoch 20/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2212 -
accuracy: 0.9135 - val_loss: 0.1769 - val_accuracy: 0.9549
Epoch 21/100
36/36 [==============================] - 0s 5ms/step - loss: 0.2150 -
accuracy: 0.9163 - val_loss: 0.1794 - val_accuracy: 0.9398
Epoch 22/100
36/36 [==============================] - 0s 3ms/step - loss: 0.2121 -
accuracy: 0.9200 - val_loss: 0.1677 - val_accuracy: 0.9474
Epoch 23/100
36/36 [==============================] - 0s 3ms/step - loss: 0.2077 -
accuracy: 0.9285 - val_loss: 0.1731 - val_accuracy: 0.9474
```

```
Epoch 24/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2038 -
accuracy: 0.9200 - val_loss: 0.1641 - val_accuracy: 0.9474
Epoch 25/100
36/36 [==============================] - 0s 4ms/step - loss: 0.2015 -
accuracy: 0.9191 - val_loss: 0.1702 - val_accuracy: 0.9474
Epoch 26/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1950 -
accuracy: 0.9276 - val_loss: 0.1607 - val_accuracy: 0.9474
Epoch 27/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1924 -
accuracy: 0.9247 - val_loss: 0.1646 - val_accuracy: 0.9474
Epoch 28/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1892 -
accuracy: 0.9247 - val_loss: 0.1608 - val_accuracy: 0.9474
Epoch 29/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1875 -
accuracy: 0.9276 - val_loss: 0.1652 - val_accuracy: 0.9398
Epoch 30/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1830 -
accuracy: 0.9266 - val_loss: 0.1535 - val_accuracy: 0.9398
Epoch 31/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1813 -
accuracy: 0.9276 - val_loss: 0.1535 - val_accuracy: 0.9474
Epoch 32/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1782 -
accuracy: 0.9257 - val_loss: 0.1566 - val_accuracy: 0.9398
Epoch 33/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1762 -
accuracy: 0.9285 - val_loss: 0.1523 - val_accuracy: 0.9549
Epoch 34/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1751 -
accuracy: 0.9341 - val_loss: 0.1568 - val_accuracy: 0.9398
Epoch 35/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1707 -
accuracy: 0.9332 - val_loss: 0.1533 - val_accuracy: 0.9474
Epoch 36/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1691 -
accuracy: 0.9285 - val_loss: 0.1593 - val_accuracy: 0.9474
Epoch 37/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1667 -
accuracy: 0.9294 - val_loss: 0.1568 - val_accuracy: 0.9398
Epoch 38/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1654 -
accuracy: 0.9294 - val_loss: 0.1493 - val_accuracy: 0.9474
Epoch 39/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1636 -
accuracy: 0.9360 - val_loss: 0.1503 - val_accuracy: 0.9398
Epoch 40/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1627 -
```

```
accuracy: 0.9323 - val_loss: 0.1450 - val_accuracy: 0.9474
Epoch 41/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1599 -
accuracy: 0.9370 - val_loss: 0.1629 - val_accuracy: 0.9398
Epoch 42/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1559 -
accuracy: 0.9341 - val_loss: 0.1451 - val_accuracy: 0.9398
Epoch 43/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1550 -
accuracy: 0.9379 - val_loss: 0.1524 - val_accuracy: 0.9323
Epoch 44/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1532 -
accuracy: 0.9351 - val_loss: 0.1400 - val_accuracy: 0.9549
Epoch 45/100
36/36 [==============================] - 0s 7ms/step - loss: 0.1511 -
accuracy: 0.9417 - val_loss: 0.1499 - val_accuracy: 0.9323
Epoch 46/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1494 -
accuracy: 0.9417 - val_loss: 0.1496 - val_accuracy: 0.9323
Epoch 47/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1476 -
accuracy: 0.9436 - val_loss: 0.1420 - val_accuracy: 0.9474
Epoch 48/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1449 -
accuracy: 0.9445 - val_loss: 0.1496 - val_accuracy: 0.9323
Epoch 49/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1452 -
accuracy: 0.9417 - val_loss: 0.1390 - val_accuracy: 0.9323
Epoch 50/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1432 -
accuracy: 0.9407 - val_loss: 0.1429 - val_accuracy: 0.9474
Epoch 51/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1442 -
accuracy: 0.9417 - val_loss: 0.1527 - val_accuracy: 0.9323
Epoch 52/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1394 -
accuracy: 0.9483 - val_loss: 0.1447 - val_accuracy: 0.9398
Epoch 53/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1383 -
accuracy: 0.9492 - val_loss: 0.1394 - val_accuracy: 0.9323
Epoch 54/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1381 -
accuracy: 0.9426 - val_loss: 0.1455 - val_accuracy: 0.9323
Epoch 55/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1368 -
accuracy: 0.9492 - val_loss: 0.1425 - val_accuracy: 0.9323
Epoch 56/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1333 -
accuracy: 0.9426 - val_loss: 0.1426 - val_accuracy: 0.9398
Epoch 57/100
```

```
36/36 [==============================] - 0s 5ms/step - loss: 0.1333 -
accuracy: 0.9492 - val_loss: 0.1429 - val_accuracy: 0.9398
Epoch 58/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1328 -
accuracy: 0.9520 - val_loss: 0.1397 - val_accuracy: 0.9323
Epoch 59/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1321 -
accuracy: 0.9492 - val_loss: 0.1437 - val_accuracy: 0.9398
Epoch 60/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1292 -
accuracy: 0.9473 - val_loss: 0.1438 - val_accuracy: 0.9398
Epoch 61/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1277 -
accuracy: 0.9436 - val_loss: 0.1372 - val_accuracy: 0.9474
Epoch 62/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1262 -
accuracy: 0.9530 - val_loss: 0.1477 - val_accuracy: 0.9398
Epoch 63/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1264 -
accuracy: 0.9483 - val_loss: 0.1420 - val_accuracy: 0.9398
Epoch 64/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1246 -
accuracy: 0.9492 - val_loss: 0.1351 - val_accuracy: 0.9398
Epoch 65/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1247 -
accuracy: 0.9511 - val_loss: 0.1442 - val_accuracy: 0.9398
Epoch 66/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1230 -
accuracy: 0.9464 - val_loss: 0.1490 - val_accuracy: 0.9398
Epoch 67/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1212 -
accuracy: 0.9492 - val_loss: 0.1396 - val_accuracy: 0.9398
Epoch 68/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1211 -
accuracy: 0.9492 - val_loss: 0.1379 - val_accuracy: 0.9398
Epoch 69/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1198 -
accuracy: 0.9539 - val_loss: 0.1441 - val_accuracy: 0.9398
Epoch 70/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1187 -
accuracy: 0.9520 - val_loss: 0.1490 - val_accuracy: 0.9323
Epoch 71/100
36/36 [==============================] - 0s 5ms/step - loss: 0.1177 -
accuracy: 0.9539 - val_loss: 0.1360 - val_accuracy: 0.9474
Epoch 72/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1181 -
accuracy: 0.9511 - val_loss: 0.1410 - val_accuracy: 0.9323
Epoch 73/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1143 -
accuracy: 0.9567 - val_loss: 0.1405 - val_accuracy: 0.9474
```

```
Epoch 74/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1138 -
accuracy: 0.9586 - val_loss: 0.1480 - val_accuracy: 0.9248
Epoch 75/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1140 -
accuracy: 0.9539 - val_loss: 0.1455 - val_accuracy: 0.9323
Epoch 76/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1128 -
accuracy: 0.9577 - val_loss: 0.1389 - val_accuracy: 0.9474
Epoch 77/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1119 -
accuracy: 0.9548 - val_loss: 0.1459 - val_accuracy: 0.9248
Epoch 78/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1118 -
accuracy: 0.9548 - val_loss: 0.1366 - val_accuracy: 0.9474
Epoch 79/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1097 -
accuracy: 0.9577 - val_loss: 0.1382 - val_accuracy: 0.9474
Epoch 80/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1096 -
accuracy: 0.9614 - val_loss: 0.1375 - val_accuracy: 0.9323
Epoch 81/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1092 -
accuracy: 0.9558 - val_loss: 0.1360 - val_accuracy: 0.9474
Epoch 82/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1063 -
accuracy: 0.9633 - val_loss: 0.1392 - val_accuracy: 0.9398
Epoch 83/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1046 -
accuracy: 0.9595 - val_loss: 0.1408 - val_accuracy: 0.9398
Epoch 84/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1037 -
accuracy: 0.9577 - val_loss: 0.1304 - val_accuracy: 0.9474
Epoch 85/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1035 -
accuracy: 0.9643 - val_loss: 0.1408 - val_accuracy: 0.9323
Epoch 86/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1034 -
accuracy: 0.9643 - val_loss: 0.1375 - val_accuracy: 0.9398
Epoch 87/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1039 -
accuracy: 0.9624 - val_loss: 0.1432 - val_accuracy: 0.9398
Epoch 88/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1023 -
accuracy: 0.9652 - val_loss: 0.1346 - val_accuracy: 0.9474
Epoch 89/100
36/36 [==============================] - 0s 3ms/step - loss: 0.1016 -
accuracy: 0.9690 - val_loss: 0.1318 - val_accuracy: 0.9474
Epoch 90/100
36/36 [==============================] - 0s 4ms/step - loss: 0.1005 -
```

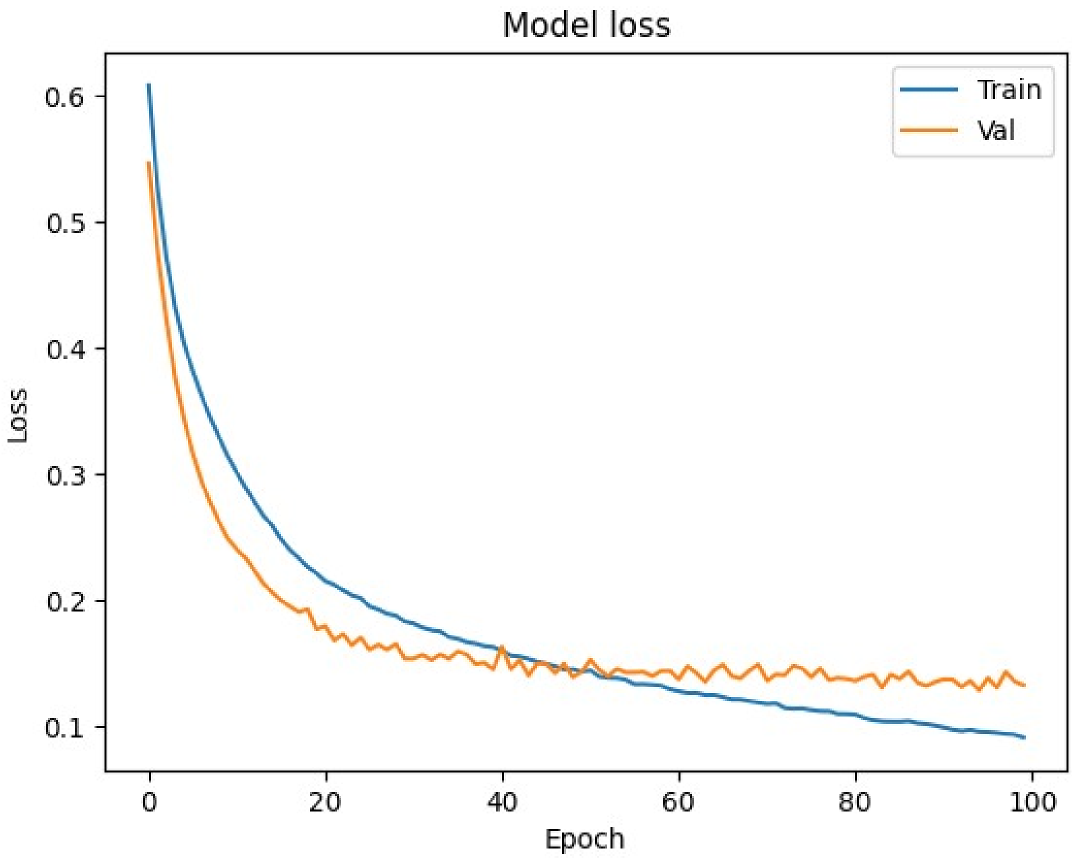
```
accuracy: 0.9614 - val_loss: 0.1350 - val_accuracy: 0.9474
Epoch 91/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0989 -
accuracy: 0.9633 - val_loss: 0.1371 - val_accuracy: 0.9474
Epoch 92/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0973 -
accuracy: 0.9680 - val_loss: 0.1370 - val_accuracy: 0.9398
Epoch 93/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0962 -
accuracy: 0.9671 - val_loss: 0.1311 - val_accuracy: 0.9474
Epoch 94/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0971 -
accuracy: 0.9661 - val_loss: 0.1359 - val_accuracy: 0.9474
Epoch 95/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0957 -
accuracy: 0.9661 - val_loss: 0.1282 - val_accuracy: 0.9474
Epoch 96/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0954 -
accuracy: 0.9708 - val_loss: 0.1382 - val_accuracy: 0.9474
Epoch 97/100
36/36 [==============================] - 0s 4ms/step - loss: 0.0947 -
accuracy: 0.9680 - val_loss: 0.1304 - val_accuracy: 0.9474
Epoch 98/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0938 -
accuracy: 0.9652 - val_loss: 0.1429 - val_accuracy: 0.9398
Epoch 99/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0933 -
accuracy: 0.9661 - val_loss: 0.1356 - val_accuracy: 0.9474
Epoch 100/100
36/36 [==============================] - 0s 3ms/step - loss: 0.0910 -
accuracy: 0.9680 - val_loss: 0.1325 - val_accuracy: 0.9398
```

```python
Model_1B.evaluate(x_test, y_test_OHE)[1]
```

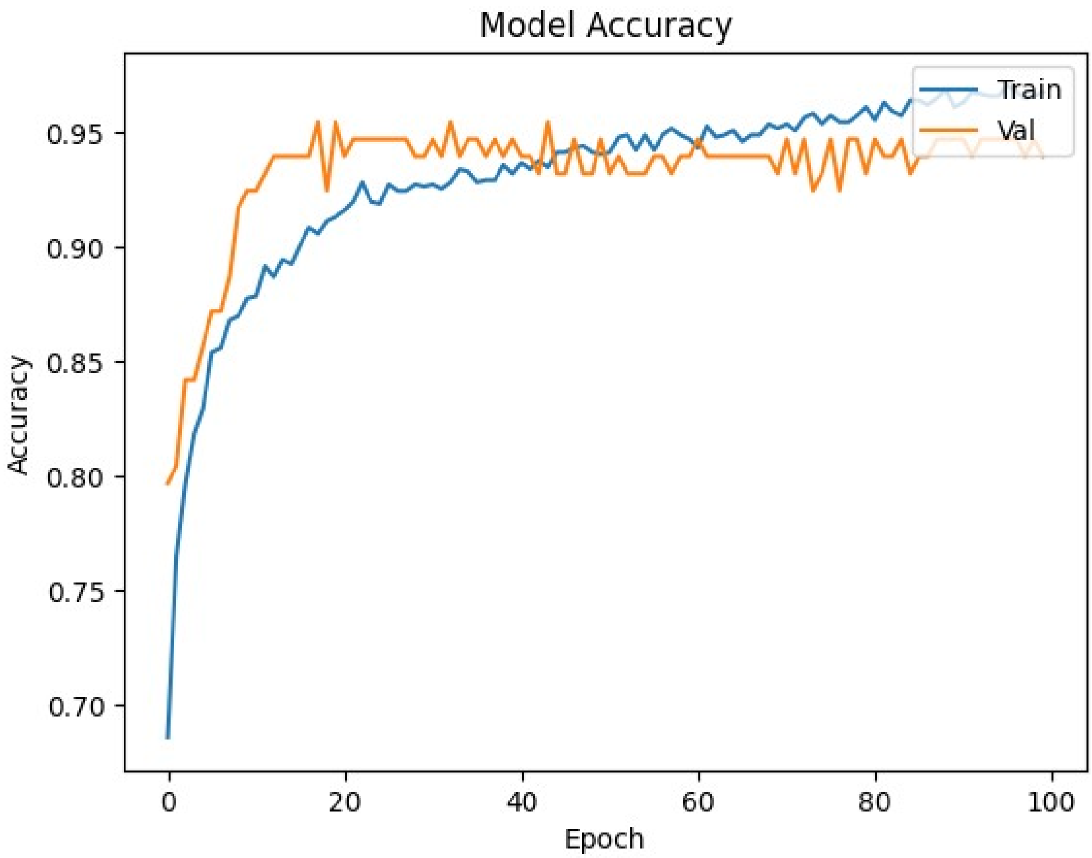
```
5/5 [==============================] - 0s 3ms/step - loss: 0.1384 -
accuracy: 0.9549
```

```
0.9548872113227844
```

```python
# plotting model loss
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
# plotting accuracy
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Dari model kedua ini dikarenakan learning rate dari Adam tetap membawakan hasil overfit ke dalam model walaupun tidak banyak. Maka dari itu diperlukan pergantian optimizer menjadi SGD dan melakukan beberapa adjust terhadap struktur ANN.

Pada model 2 ini didapatkan akurasi 95% dan untuk validation akurasi 91%. Sedangkan pada saat dibandingkan dengan test menghasilkan akurasi 91%.

Membentuk model 1C yang mana akan menganti model yang dibuat dengan hidden layer dan melakukan adijust terhadap jumlah neuron

```
Model_1C = keras.Sequential()

# Add input layer, hidden layer, and output layer
# Hidden Layer 1
Model_1C.add(layers.Dense(14,input_shape=(8,), activation="relu"))
# Hidden Layer 2
Model_1C.add(layers.Dense(18,activation='relu'))
# Output layer
Model_1C.add(layers.Dense(2, activation='sigmoid'))
```

Model yang digunkanan masih tetap sama

```
Model_1C.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 14)                126

 dense_22 (Dense)            (None, 18)                270

 dense_23 (Dense)            (None, 2)                 38

=================================================================
Total params: 434
Trainable params: 434
Non-trainable params: 0
_________________________________________________________________
```

```
# compiling the model
Model_1C.compile(loss =
'categorical_crossentropy',optimizer=tf.optimizers.SGD(learning_rate=0
.01),metrics = ['accuracy'])

epochs = 100

history3 = Model_1C.fit(
  x_train, y_train_onehot, validation_data= (x_val, y_val_OHE),
  epochs = epochs, verbose = 1, batch_size=20,
  callbacks = [tfdocs.modeling.EpochDots()])
```

```
Epoch 1/100
30/54 [===============>..............] - ETA: 0s - loss: 0.7308 -
accuracy: 0.5650
Epoch: 0, accuracy:0.5851,  loss:0.7008,  val_accuracy:0.6090,
val_loss:0.6651,
54/54 [==============================] - 1s 6ms/step - loss: 0.7008 -
accuracy: 0.5851 - val_loss: 0.6651 - val_accuracy: 0.6090
Epoch 2/100
54/54 [==============================] - 0s 3ms/step - loss: 0.6432 -
accuracy: 0.6529 - val_loss: 0.6186 - val_accuracy: 0.6767
Epoch 3/100
54/54 [==============================] - 0s 3ms/step - loss: 0.6012 -
accuracy: 0.7244 - val_loss: 0.5782 - val_accuracy: 0.7218
Epoch 4/100
54/54 [==============================] - 0s 3ms/step - loss: 0.5638 -
accuracy: 0.7516 - val_loss: 0.5425 - val_accuracy: 0.7519
Epoch 5/100
54/54 [==============================] - 0s 3ms/step - loss: 0.5319 -
accuracy: 0.7648 - val_loss: 0.5137 - val_accuracy: 0.7594
Epoch 6/100
54/54 [==============================] - 0s 3ms/step - loss: 0.5055 -
accuracy: 0.7752 - val_loss: 0.4868 - val_accuracy: 0.7669
Epoch 7/100
```

```
54/54 [==============================] - 0s 3ms/step - loss: 0.4821 -
accuracy: 0.7827 - val_loss: 0.4616 - val_accuracy: 0.7669
Epoch 8/100
54/54 [==============================] - 0s 3ms/step - loss: 0.4617 -
accuracy: 0.7930 - val_loss: 0.4385 - val_accuracy: 0.7820
Epoch 9/100
54/54 [==============================] - 0s 2ms/step - loss: 0.4446 -
accuracy: 0.8043 - val_loss: 0.4188 - val_accuracy: 0.8045
Epoch 10/100
54/54 [==============================] - 0s 2ms/step - loss: 0.4294 -
accuracy: 0.8081 - val_loss: 0.4000 - val_accuracy: 0.8195
Epoch 11/100
54/54 [==============================] - 0s 2ms/step - loss: 0.4151 -
accuracy: 0.8156 - val_loss: 0.3820 - val_accuracy: 0.8496
Epoch 12/100
54/54 [==============================] - 0s 3ms/step - loss: 0.4024 -
accuracy: 0.8241 - val_loss: 0.3650 - val_accuracy: 0.8571
Epoch 13/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3903 -
accuracy: 0.8382 - val_loss: 0.3509 - val_accuracy: 0.8496
Epoch 14/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3790 -
accuracy: 0.8542 - val_loss: 0.3365 - val_accuracy: 0.8797
Epoch 15/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3684 -
accuracy: 0.8645 - val_loss: 0.3208 - val_accuracy: 0.9098
Epoch 16/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3590 -
accuracy: 0.8674 - val_loss: 0.3088 - val_accuracy: 0.9173
Epoch 17/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3504 -
accuracy: 0.8749 - val_loss: 0.3006 - val_accuracy: 0.9248
Epoch 18/100
54/54 [==============================] - 0s 2ms/step - loss: 0.3432 -
accuracy: 0.8758 - val_loss: 0.2919 - val_accuracy: 0.9248
Epoch 19/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3365 -
accuracy: 0.8833 - val_loss: 0.2845 - val_accuracy: 0.9323
Epoch 20/100
54/54 [==============================] - 0s 2ms/step - loss: 0.3302 -
accuracy: 0.8852 - val_loss: 0.2763 - val_accuracy: 0.9398
Epoch 21/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3247 -
accuracy: 0.8871 - val_loss: 0.2702 - val_accuracy: 0.9398
Epoch 22/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3193 -
accuracy: 0.8899 - val_loss: 0.2623 - val_accuracy: 0.9398
Epoch 23/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3147 -
accuracy: 0.8928 - val_loss: 0.2608 - val_accuracy: 0.9323
```

```
Epoch 24/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3101 -
accuracy: 0.8899 - val_loss: 0.2557 - val_accuracy: 0.9398
Epoch 25/100
54/54 [==============================] - 0s 3ms/step - loss: 0.3060 -
accuracy: 0.8918 - val_loss: 0.2534 - val_accuracy: 0.9398
Epoch 26/100
54/54 [==============================] - 0s 2ms/step - loss: 0.3015 -
accuracy: 0.8899 - val_loss: 0.2455 - val_accuracy: 0.9474
Epoch 27/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2981 -
accuracy: 0.8909 - val_loss: 0.2477 - val_accuracy: 0.9398
Epoch 28/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2939 -
accuracy: 0.8956 - val_loss: 0.2411 - val_accuracy: 0.9474
Epoch 29/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2903 -
accuracy: 0.8956 - val_loss: 0.2412 - val_accuracy: 0.9474
Epoch 30/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2866 -
accuracy: 0.8918 - val_loss: 0.2366 - val_accuracy: 0.9549
Epoch 31/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2833 -
accuracy: 0.8928 - val_loss: 0.2312 - val_accuracy: 0.9549
Epoch 32/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2802 -
accuracy: 0.8928 - val_loss: 0.2337 - val_accuracy: 0.9398
Epoch 33/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2774 -
accuracy: 0.8937 - val_loss: 0.2279 - val_accuracy: 0.9549
Epoch 34/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2739 -
accuracy: 0.8993 - val_loss: 0.2286 - val_accuracy: 0.9549
Epoch 35/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2712 -
accuracy: 0.9003 - val_loss: 0.2278 - val_accuracy: 0.9474
Epoch 36/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2681 -
accuracy: 0.9003 - val_loss: 0.2212 - val_accuracy: 0.9549
Epoch 37/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2657 -
accuracy: 0.9031 - val_loss: 0.2203 - val_accuracy: 0.9549
Epoch 38/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2627 -
accuracy: 0.9031 - val_loss: 0.2211 - val_accuracy: 0.9474
Epoch 39/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2602 -
accuracy: 0.9031 - val_loss: 0.2208 - val_accuracy: 0.9474
Epoch 40/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2576 -
```

```
accuracy: 0.9059 - val_loss: 0.2158 - val_accuracy: 0.9624
Epoch 41/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2553 -
accuracy: 0.9069 - val_loss: 0.2145 - val_accuracy: 0.9624
Epoch 42/100
54/54 [==============================] - 0s 5ms/step - loss: 0.2528 -
accuracy: 0.9116 - val_loss: 0.2173 - val_accuracy: 0.9474
Epoch 43/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2506 -
accuracy: 0.9116 - val_loss: 0.2143 - val_accuracy: 0.9699
Epoch 44/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2484 -
accuracy: 0.9116 - val_loss: 0.2133 - val_accuracy: 0.9624
Epoch 45/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2465 -
accuracy: 0.9135 - val_loss: 0.2129 - val_accuracy: 0.9624
Epoch 46/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2442 -
accuracy: 0.9125 - val_loss: 0.2200 - val_accuracy: 0.9398
Epoch 47/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2424 -
accuracy: 0.9135 - val_loss: 0.2161 - val_accuracy: 0.9549
Epoch 48/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2403 -
accuracy: 0.9163 - val_loss: 0.2140 - val_accuracy: 0.9549
Epoch 49/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2386 -
accuracy: 0.9135 - val_loss: 0.2125 - val_accuracy: 0.9624
Epoch 50/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2372 -
accuracy: 0.9144 - val_loss: 0.2115 - val_accuracy: 0.9549
Epoch 51/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2351 -
accuracy: 0.9153 - val_loss: 0.2117 - val_accuracy: 0.9549
Epoch 52/100
54/54 [==============================] - 0s 4ms/step - loss: 0.2333 -
accuracy: 0.9153 - val_loss: 0.2129 - val_accuracy: 0.9474
Epoch 53/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2316 -
accuracy: 0.9153 - val_loss: 0.2168 - val_accuracy: 0.9398
Epoch 54/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2299 -
accuracy: 0.9182 - val_loss: 0.2117 - val_accuracy: 0.9549
Epoch 55/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2288 -
accuracy: 0.9135 - val_loss: 0.2173 - val_accuracy: 0.9323
Epoch 56/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2269 -
accuracy: 0.9144 - val_loss: 0.2217 - val_accuracy: 0.9323
Epoch 57/100
```

```
54/54 [==============================] - 0s 2ms/step - loss: 0.2249 -
accuracy: 0.9135 - val_loss: 0.2151 - val_accuracy: 0.9474
Epoch 58/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2235 -
accuracy: 0.9144 - val_loss: 0.2211 - val_accuracy: 0.9248
Epoch 59/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2227 -
accuracy: 0.9125 - val_loss: 0.2106 - val_accuracy: 0.9474
Epoch 60/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2210 -
accuracy: 0.9163 - val_loss: 0.2113 - val_accuracy: 0.9474
Epoch 61/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2198 -
accuracy: 0.9191 - val_loss: 0.2133 - val_accuracy: 0.9398
Epoch 62/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2189 -
accuracy: 0.9153 - val_loss: 0.2088 - val_accuracy: 0.9474
Epoch 63/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2179 -
accuracy: 0.9200 - val_loss: 0.2295 - val_accuracy: 0.9248
Epoch 64/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2177 -
accuracy: 0.9200 - val_loss: 0.2124 - val_accuracy: 0.9474
Epoch 65/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2153 -
accuracy: 0.9182 - val_loss: 0.2065 - val_accuracy: 0.9474
Epoch 66/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2144 -
accuracy: 0.9200 - val_loss: 0.2059 - val_accuracy: 0.9474
Epoch 67/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2137 -
accuracy: 0.9210 - val_loss: 0.2080 - val_accuracy: 0.9474
Epoch 68/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2115 -
accuracy: 0.9153 - val_loss: 0.2062 - val_accuracy: 0.9474
Epoch 69/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2109 -
accuracy: 0.9210 - val_loss: 0.2126 - val_accuracy: 0.9474
Epoch 70/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2099 -
accuracy: 0.9238 - val_loss: 0.2113 - val_accuracy: 0.9398
Epoch 71/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2091 -
accuracy: 0.9247 - val_loss: 0.2132 - val_accuracy: 0.9323
Epoch 72/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2090 -
accuracy: 0.9247 - val_loss: 0.2066 - val_accuracy: 0.9474
Epoch 73/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2072 -
accuracy: 0.9238 - val_loss: 0.2064 - val_accuracy: 0.9474
```

```
Epoch 74/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2062 -
accuracy: 0.9238 - val_loss: 0.2071 - val_accuracy: 0.9398
Epoch 75/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2054 -
accuracy: 0.9247 - val_loss: 0.2047 - val_accuracy: 0.9474
Epoch 76/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2042 -
accuracy: 0.9257 - val_loss: 0.2128 - val_accuracy: 0.9398
Epoch 77/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2041 -
accuracy: 0.9257 - val_loss: 0.2060 - val_accuracy: 0.9474
Epoch 78/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2024 -
accuracy: 0.9276 - val_loss: 0.2122 - val_accuracy: 0.9398
Epoch 79/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2022 -
accuracy: 0.9257 - val_loss: 0.2074 - val_accuracy: 0.9474
Epoch 80/100
54/54 [==============================] - 0s 2ms/step - loss: 0.2012 -
accuracy: 0.9276 - val_loss: 0.2050 - val_accuracy: 0.9474
Epoch 81/100
54/54 [==============================] - 0s 3ms/step - loss: 0.2000 -
accuracy: 0.9266 - val_loss: 0.2079 - val_accuracy: 0.9474
Epoch 82/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1989 -
accuracy: 0.9285 - val_loss: 0.2000 - val_accuracy: 0.9549
Epoch 83/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1993 -
accuracy: 0.9276 - val_loss: 0.2038 - val_accuracy: 0.9474
Epoch 84/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1984 -
accuracy: 0.9257 - val_loss: 0.1979 - val_accuracy: 0.9549
Epoch 85/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1978 -
accuracy: 0.9285 - val_loss: 0.2019 - val_accuracy: 0.9474
Epoch 86/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1959 -
accuracy: 0.9276 - val_loss: 0.1969 - val_accuracy: 0.9549
Epoch 87/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1962 -
accuracy: 0.9294 - val_loss: 0.2008 - val_accuracy: 0.9398
Epoch 88/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1949 -
accuracy: 0.9294 - val_loss: 0.2055 - val_accuracy: 0.9398
Epoch 89/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1941 -
accuracy: 0.9276 - val_loss: 0.1944 - val_accuracy: 0.9474
Epoch 90/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1942 -
```

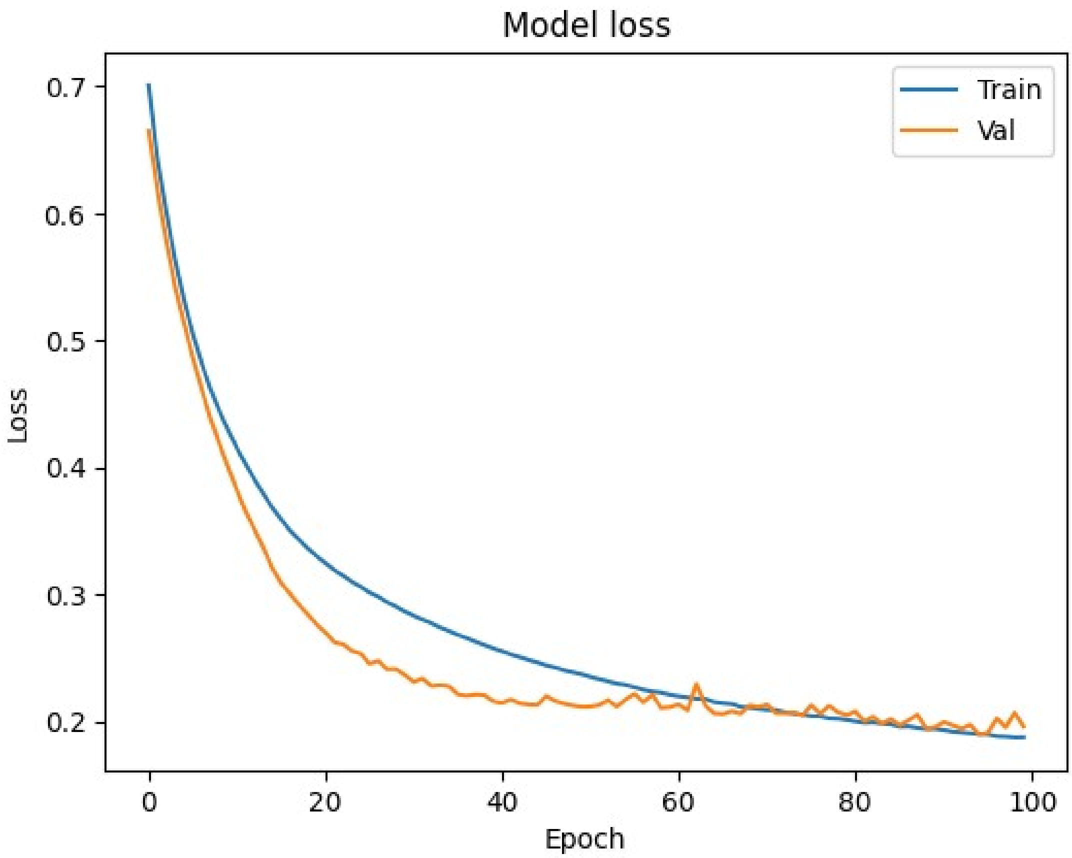
```
accuracy: 0.9276 - val_loss: 0.1955 - val_accuracy: 0.9474
Epoch 91/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1933 -
accuracy: 0.9257 - val_loss: 0.1995 - val_accuracy: 0.9398
Epoch 92/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1920 -
accuracy: 0.9247 - val_loss: 0.1970 - val_accuracy: 0.9398
Epoch 93/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1913 -
accuracy: 0.9294 - val_loss: 0.1943 - val_accuracy: 0.9398
Epoch 94/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1906 -
accuracy: 0.9266 - val_loss: 0.1973 - val_accuracy: 0.9398
Epoch 95/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1900 -
accuracy: 0.9257 - val_loss: 0.1895 - val_accuracy: 0.9474
Epoch 96/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1897 -
accuracy: 0.9266 - val_loss: 0.1913 - val_accuracy: 0.9549
Epoch 97/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1883 -
accuracy: 0.9247 - val_loss: 0.2024 - val_accuracy: 0.9323
Epoch 98/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1880 -
accuracy: 0.9266 - val_loss: 0.1954 - val_accuracy: 0.9398
Epoch 99/100
54/54 [==============================] - 0s 3ms/step - loss: 0.1874 -
accuracy: 0.9257 - val_loss: 0.2070 - val_accuracy: 0.9323
Epoch 100/100
54/54 [==============================] - 0s 2ms/step - loss: 0.1875 -
accuracy: 0.9247 - val_loss: 0.1961 - val_accuracy: 0.9398
```

```
Model_1C.evaluate(x_test, y_test_OHE)[1]
```

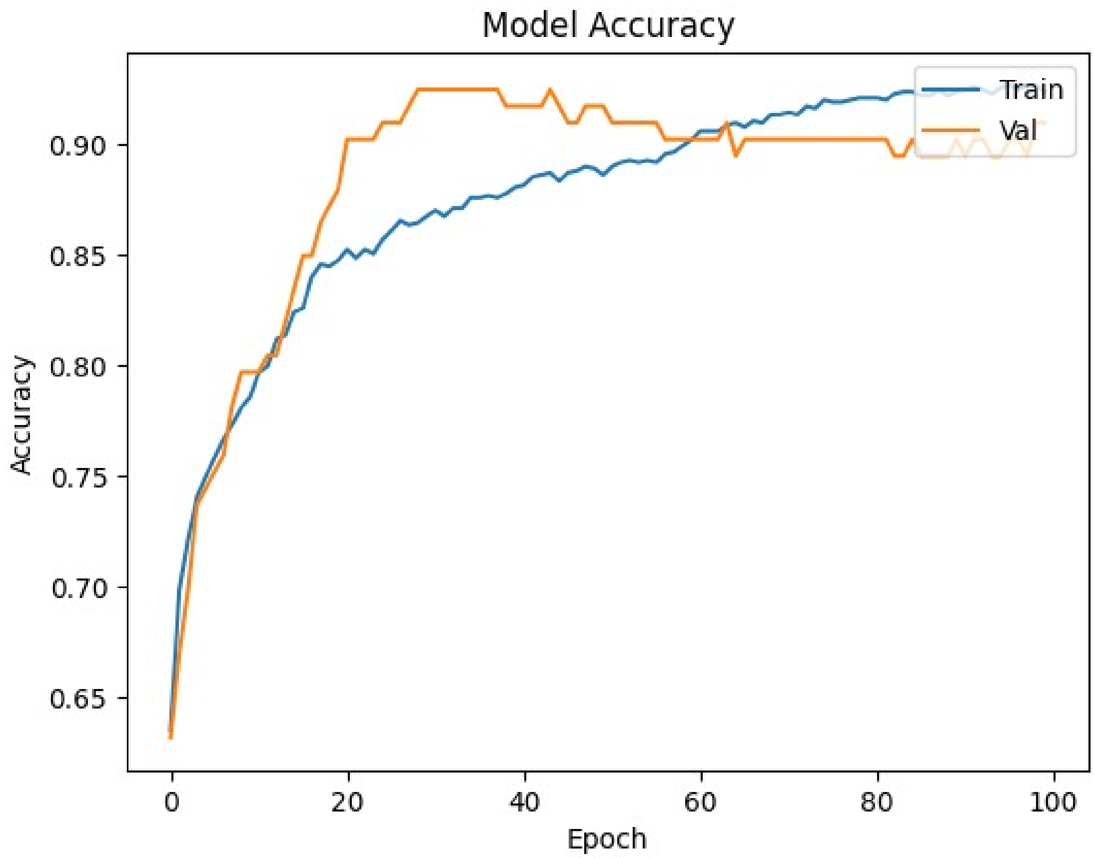
```
5/5 [==============================] - 0s 4ms/step - loss: 0.2815 -
accuracy: 0.9323
```

```
0.932330846786499
```

```
# plotting model loss
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```python
# plotting accuracy
plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Dengan mengganti optimizer menjadi SGD dan learning rate dicepatkan menjadi 0.01 dapat mengontrol overfit dari model. Walaupun hasil dari akurasi tidak akan sebagus dengan menggunakan optimizer dari Adam. Akurasi yang didapatkan adalah 92% dan untuk validasi 93%. Sedangkan pada saat di evaluasi dengan test 93%.

Pada model ketiga ini juga terdapat pergantian jumlah neuron, tetapi jumlah hidden layer masih tetaplah sama.

Kesimpulan yang bisa didapatkan adalah perbedaan dari model nomor 1C adalah :

- Perbedaan jumlah neuron

  Pada perbedaan ini yang saya ganti dapat terlihat pada neuron yang awalanya hidden layer 1, 16 neuron menjadi 14 neuoron. Sedangkan untuk hidden layer 2, dari 16 neuron menjadi 18 neuron.

- Perbedaan activasion function

  Pada perbedaana activasion function dapat terlihat pada hidden layer terakhir atau output yang mana pada awalnya tidak ada activation function ditambahkan dengan sigmoid activation function di layer terakhir. Sehingga dapat menambahkan akurasi performa

- Penggunaan optimizer

Pada penggunaan optimzer yang dipilih adalah SGD dengan learning rate 0.01. Saya tidak memilih optimzer Adam karena adalam pada model yang tergolong simple cenderung overfit dapat dilihat dari spike loss yang dihasilkan dan lebih susah dikontrol.

- Penyetelan batch size

  Pada model ini batch size yang saya gunakan 20 untuk mendapatkan hasil output optimal yaitu meningkat akurasi yang diberikan

- Pergantian epochs

  Pada model epoch yang saya gunakan 100 untuk melihat model dengan iterasi yang lebih lama dari awalnya 40. Saya tidak menambahkan lebih dari 100 karena dapat memungkinkan overfitting.

- arsitektur

  Pada model ini tetap menggunakan aristektur yang sama dari basenya yaitu dua hidden layer dengan input 8 dan output 2.

- Batch mornalization

  Pada model ini saat digunkanan batch normalization akan terjadi spike di model loss sehingga saya menghapus pengunaan batch normalization dapaat dikarenakan arsitektur model yang sederhana.

1F. [LO 3, LO 4, 5 poin] Lakukan evaluasi unjuk kerja kedua arsitektur di atas pada test set dengan mencari nilai accuracy, precision, recall dan F1-Score. Dan berikan penjelasan mengenai hasilnya dengan rinci.

Dari sini akan membuat model predict untuk melihat hasil prediksi dan akan disimpkan ke variable

```
# model base
y_pred_base = FirstModel.predict(x_test)
# modified model
y_pred_mod = Model_1C.predict(x_test)
```

```
5/5 [==============================] - 0s 2ms/step
5/5 [==============================] - 0s 2ms/step
```

```
# import libray for classification danconfusion_matrix
from sklearn.metrics import classification_report, confusion_matrix
```

Melihat confussion table

```
# confussion table
print("Base Model :")
print(classification_report(y_test, y_pred_base.argmax(axis=1)))
print()
```

```
print("Modified Model :")
print(classification_report(y_test, y_pred_mod.argmax(axis=1)))
```

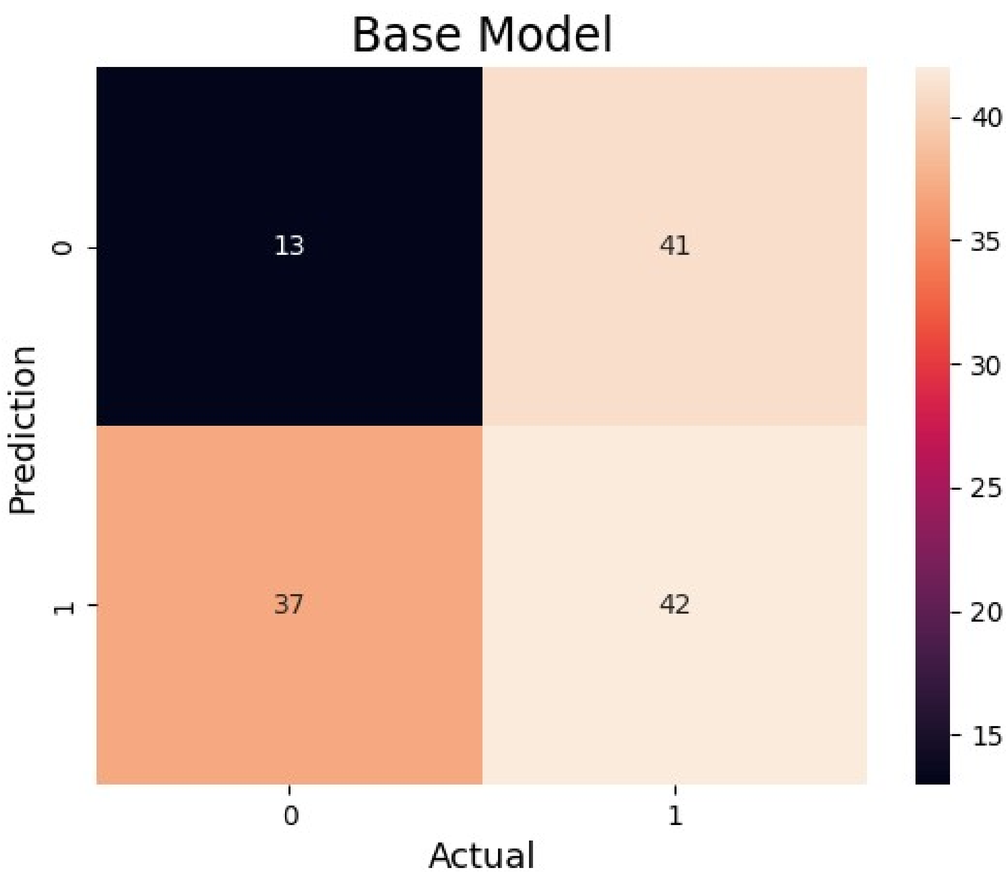
```
Base Model :
              precision    recall  f1-score   support

           0       0.26      0.24      0.25        54
           1       0.51      0.53      0.52        79

    accuracy                           0.41       133
   macro avg       0.38      0.39      0.38       133
weighted avg       0.41      0.41      0.41       133


Modified Model :
              precision    recall  f1-score   support

           0       0.87      0.98      0.92        54
           1       0.99      0.90      0.94        79

    accuracy                           0.93       133
   macro avg       0.93      0.94      0.93       133
weighted avg       0.94      0.93      0.93       133
```

Define fuction buat confussion matrix

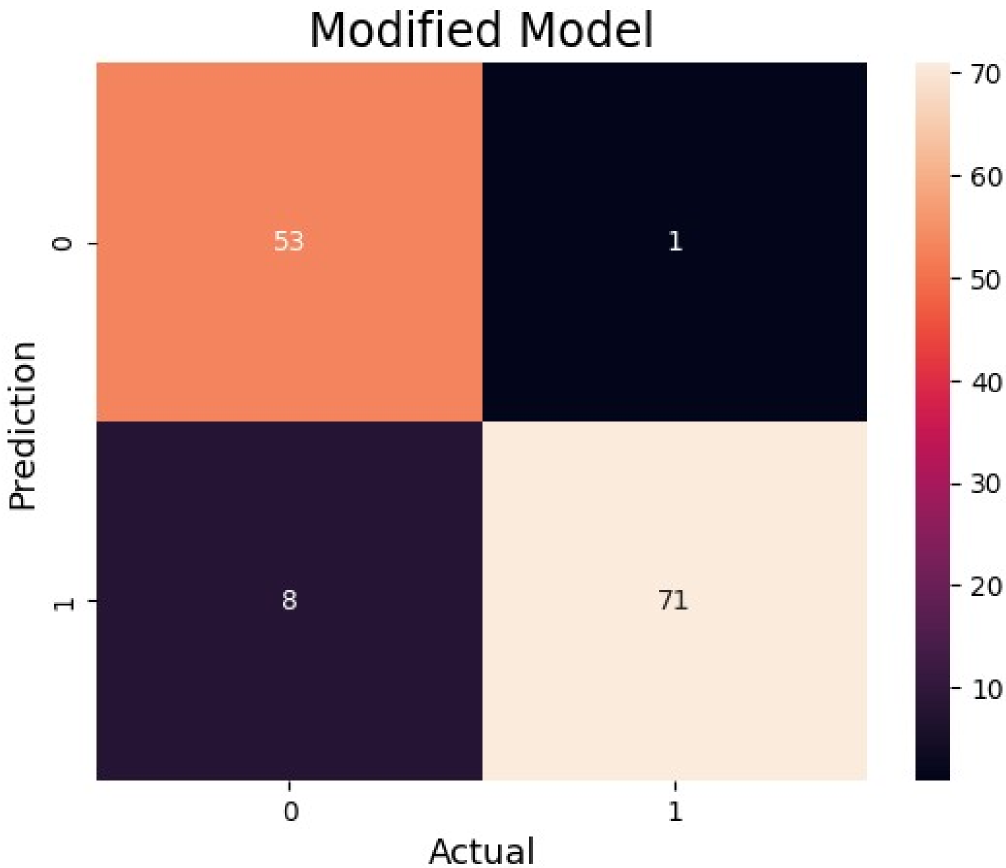
```
def con_plot(y_test,y_pred,label):
  conf = confusion_matrix(y_test,y_pred)
  sns.heatmap(conf,annot=True)
  plt.ylabel('Prediction',fontsize=13)
  plt.xlabel('Actual',fontsize=13)
  plt.title(label,fontsize=17)
  plt.show()
```

Model base counfison matrix :

```
# plotting confusion matrix
con_plot(y_test, y_pred_base.argmax(axis=1),"Base Model")
con_plot(y_test, y_pred_mod.argmax(axis=1),"Modified Model")
```



Base Model
Prediction
Actual
0
1
0
1
13
41
37
42
40
35
30
25
20
15



Dari sini dapat dilihat dengan modifed model dapat membuat prediksi lebih baik lagi dapat dilihat dari perbedaan confussion model antara modifed dengan base. Base memiliki value yang bukan true value (0,0 atau 1,1) banyak, yiatu 37 dan 41. Sedangkan dengan modified model di dapatkan value yang bukan true value lebih dikit, yaitu 8 dan 1.

JIki di lihat dari classification report di dapatkan base model mendapatkan rata" 50% sendangkan untuk modified mendapatkan rata" di 90%.

1E. [LO 1, LO 2, LO 3, LO 4 5 poin] Buatlah video presentasi yang menjelaskan arsitektur yang dibangun untuk mengklasifikasikan sebuah klaim ini.

Link video penjelasan : https://drive.google.com/file/d/1G-ZBwxn9a5N80Gyjbx0GPnfen0zVq7pc/view?usp=sharing