lab-mse-part-a

December 4, 2023

breast cancer prediction using ann

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
```

[2]:
```
cancer=load_breast_cancer()
x=cancer.data
y=cancer.target
```

[3]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

[4]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
```

[5]:
```
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)
```

[6]:
```
import tensorflow as tf
from tensorflow import keras
```

```
WARNING:tensorflow:From C:\Users\HP\anaconda3\envs\AML-manyahegde\lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

[7]:
```
model=keras.Sequential([
    keras.layers.Flatten(input_shape=(30,)),
    keras.layers.Dense(6,activation='relu',kernel_initializer='he_uniform'),
    keras.layers.Dense(6,activation='relu',kernel_initializer='he_uniform'),
    keras.layers.
 ↪Dense(1,activation='sigmoid',kernel_initializer='glorot_uniform')
])
```

```
WARNING:tensorflow:From C:\Users\HP\anaconda3\envs\AML-manyahegde\lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
```

```
Please use tf.compat.v1.get_default_graph instead.
```

[8]:
```python
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\HP\anaconda3\envs\AML-manyahegde\lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

[9]:
```python
history=model.fit(x_train,y_train,validation_split=0.1,epochs=50)
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\HP\anaconda3\envs\AML-manyahegde\lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\HP\anaconda3\envs\AML-manyahegde\lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

13/13 [==============================] - 2s 37ms/step - loss: 0.4724 - accuracy:
0.8240 - val_loss: 0.5150 - val_accuracy: 0.8043
Epoch 2/50
13/13 [==============================] - 0s 9ms/step - loss: 0.4165 - accuracy:
0.8435 - val_loss: 0.4774 - val_accuracy: 0.8696
Epoch 3/50
13/13 [==============================] - 0s 8ms/step - loss: 0.3738 - accuracy:
0.8851 - val_loss: 0.4419 - val_accuracy: 0.8696
Epoch 4/50
13/13 [==============================] - 0s 9ms/step - loss: 0.3392 - accuracy:
0.9046 - val_loss: 0.4064 - val_accuracy: 0.8696
Epoch 5/50
13/13 [==============================] - 0s 9ms/step - loss: 0.3102 - accuracy:
0.9144 - val_loss: 0.3728 - val_accuracy: 0.9130
Epoch 6/50
13/13 [==============================] - 0s 9ms/step - loss: 0.2845 - accuracy:
0.9193 - val_loss: 0.3447 - val_accuracy: 0.9348
Epoch 7/50
13/13 [==============================] - 0s 9ms/step - loss: 0.2626 - accuracy:
0.9291 - val_loss: 0.3187 - val_accuracy: 0.9348
Epoch 8/50
13/13 [==============================] - 0s 7ms/step - loss: 0.2432 - accuracy:
0.9315 - val_loss: 0.2931 - val_accuracy: 0.9348
Epoch 9/50
13/13 [==============================] - 0s 8ms/step - loss: 0.2264 - accuracy:
0.9389 - val_loss: 0.2674 - val_accuracy: 0.9565
```

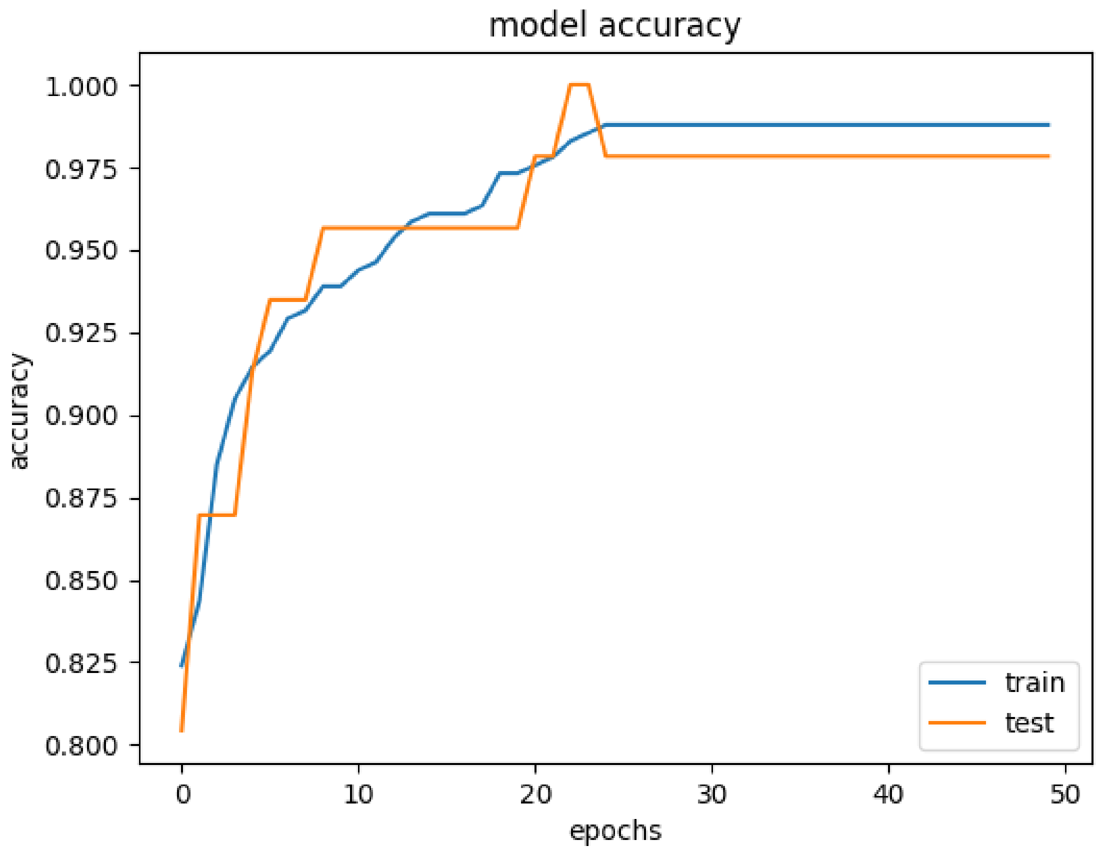
```
Epoch 10/50
13/13 [==============================] - 0s 8ms/step - loss: 0.2104 - accuracy:
0.9389 - val_loss: 0.2446 - val_accuracy: 0.9565
Epoch 11/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1961 - accuracy:
0.9438 - val_loss: 0.2219 - val_accuracy: 0.9565
Epoch 12/50
13/13 [==============================] - 0s 9ms/step - loss: 0.1819 - accuracy:
0.9462 - val_loss: 0.2027 - val_accuracy: 0.9565
Epoch 13/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1679 - accuracy:
0.9535 - val_loss: 0.1870 - val_accuracy: 0.9565
Epoch 14/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1567 - accuracy:
0.9584 - val_loss: 0.1741 - val_accuracy: 0.9565
Epoch 15/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1467 - accuracy:
0.9609 - val_loss: 0.1631 - val_accuracy: 0.9565
Epoch 16/50
13/13 [==============================] - 0s 7ms/step - loss: 0.1381 - accuracy:
0.9609 - val_loss: 0.1539 - val_accuracy: 0.9565
Epoch 17/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1300 - accuracy:
0.9609 - val_loss: 0.1454 - val_accuracy: 0.9565
Epoch 18/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1228 - accuracy:
0.9633 - val_loss: 0.1375 - val_accuracy: 0.9565
Epoch 19/50
13/13 [==============================] - 0s 8ms/step - loss: 0.1155 - accuracy:
0.9731 - val_loss: 0.1297 - val_accuracy: 0.9565
Epoch 20/50
13/13 [==============================] - 0s 7ms/step - loss: 0.1087 - accuracy:
0.9731 - val_loss: 0.1205 - val_accuracy: 0.9565
Epoch 21/50
13/13 [==============================] - 0s 9ms/step - loss: 0.1028 - accuracy:
0.9756 - val_loss: 0.1117 - val_accuracy: 0.9783
Epoch 22/50
13/13 [==============================] - 0s 7ms/step - loss: 0.0976 - accuracy:
0.9780 - val_loss: 0.1036 - val_accuracy: 0.9783
Epoch 23/50
13/13 [==============================] - 0s 7ms/step - loss: 0.0931 - accuracy:
0.9829 - val_loss: 0.0956 - val_accuracy: 1.0000
Epoch 24/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0893 - accuracy:
0.9853 - val_loss: 0.0896 - val_accuracy: 1.0000
Epoch 25/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0853 - accuracy:
0.9878 - val_loss: 0.0855 - val_accuracy: 0.9783
```

```
Epoch 26/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0824 - accuracy:
0.9878 - val_loss: 0.0823 - val_accuracy: 0.9783
Epoch 27/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0798 - accuracy:
0.9878 - val_loss: 0.0785 - val_accuracy: 0.9783
Epoch 28/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0773 - accuracy:
0.9878 - val_loss: 0.0760 - val_accuracy: 0.9783
Epoch 29/50
13/13 [==============================] - 0s 9ms/step - loss: 0.0755 - accuracy:
0.9878 - val_loss: 0.0735 - val_accuracy: 0.9783
Epoch 30/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0736 - accuracy:
0.9878 - val_loss: 0.0703 - val_accuracy: 0.9783
Epoch 31/50
13/13 [==============================] - 0s 9ms/step - loss: 0.0717 - accuracy:
0.9878 - val_loss: 0.0686 - val_accuracy: 0.9783
Epoch 32/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0702 - accuracy:
0.9878 - val_loss: 0.0668 - val_accuracy: 0.9783
Epoch 33/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0687 - accuracy:
0.9878 - val_loss: 0.0656 - val_accuracy: 0.9783
Epoch 34/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0674 - accuracy:
0.9878 - val_loss: 0.0639 - val_accuracy: 0.9783
Epoch 35/50
13/13 [==============================] - 0s 7ms/step - loss: 0.0661 - accuracy:
0.9878 - val_loss: 0.0625 - val_accuracy: 0.9783
Epoch 36/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0649 - accuracy:
0.9878 - val_loss: 0.0607 - val_accuracy: 0.9783
Epoch 37/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0638 - accuracy:
0.9878 - val_loss: 0.0604 - val_accuracy: 0.9783
Epoch 38/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0627 - accuracy:
0.9878 - val_loss: 0.0590 - val_accuracy: 0.9783
Epoch 39/50
13/13 [==============================] - 0s 9ms/step - loss: 0.0617 - accuracy:
0.9878 - val_loss: 0.0578 - val_accuracy: 0.9783
Epoch 40/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0608 - accuracy:
0.9878 - val_loss: 0.0575 - val_accuracy: 0.9783
Epoch 41/50
13/13 [==============================] - 0s 9ms/step - loss: 0.0599 - accuracy:
0.9878 - val_loss: 0.0563 - val_accuracy: 0.9783
```

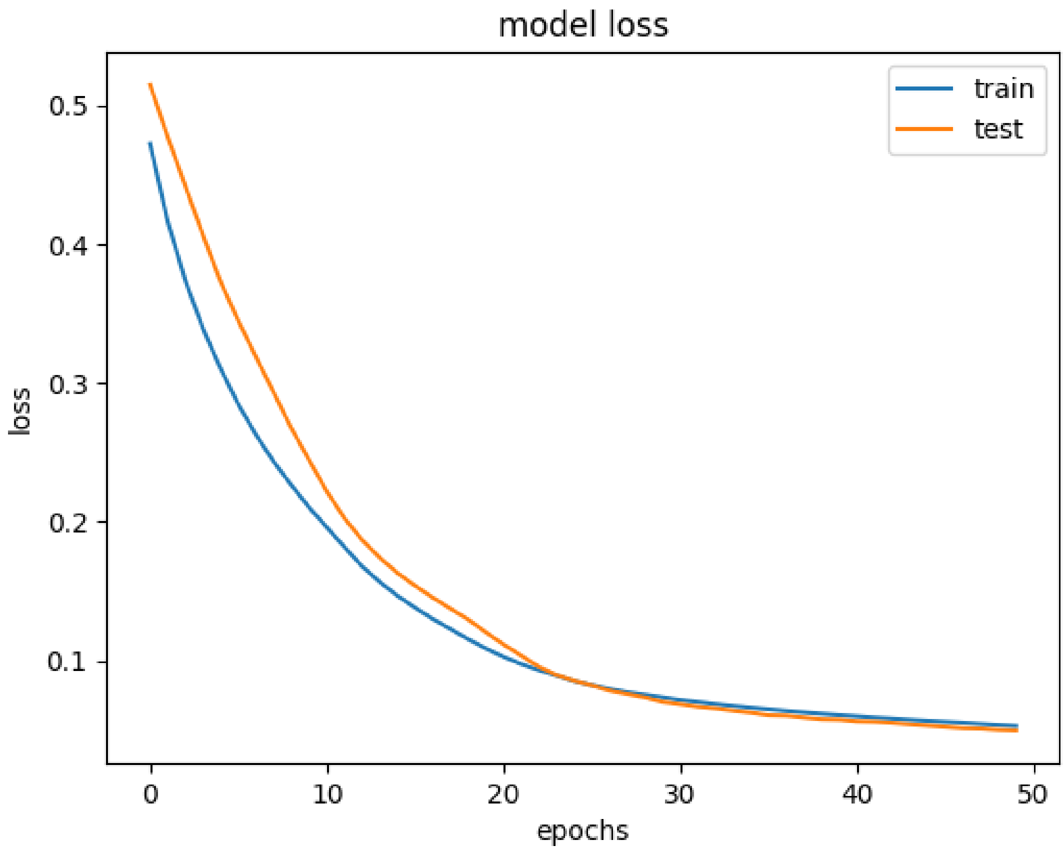
```
Epoch 42/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0590 - accuracy:
0.9878 - val_loss: 0.0560 - val_accuracy: 0.9783
Epoch 43/50
13/13 [==============================] - 0s 9ms/step - loss: 0.0582 - accuracy:
0.9878 - val_loss: 0.0552 - val_accuracy: 0.9783
Epoch 44/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0574 - accuracy:
0.9878 - val_loss: 0.0541 - val_accuracy: 0.9783
Epoch 45/50
13/13 [==============================] - 0s 9ms/step - loss: 0.0566 - accuracy:
0.9878 - val_loss: 0.0533 - val_accuracy: 0.9783
Epoch 46/50
13/13 [==============================] - 0s 10ms/step - loss: 0.0559 - accuracy:
0.9878 - val_loss: 0.0523 - val_accuracy: 0.9783
Epoch 47/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0552 - accuracy:
0.9878 - val_loss: 0.0514 - val_accuracy: 0.9783
Epoch 48/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0544 - accuracy:
0.9878 - val_loss: 0.0511 - val_accuracy: 0.9783
Epoch 49/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0537 - accuracy:
0.9878 - val_loss: 0.0502 - val_accuracy: 0.9783
Epoch 50/50
13/13 [==============================] - 0s 8ms/step - loss: 0.0530 - accuracy:
0.9878 - val_loss: 0.0498 - val_accuracy: 0.9783
```

[10]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.title('model accuracy')
plt.legend(['train','test'],loc='lower right')
plt.show()
```



[11]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
plt.title('model loss')
plt.legend(['train','test'],loc='upper right')
plt.show()
```



[12]:
```
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)
```

```
4/4 [==============================] - 0s 3ms/step
```

[13]:
```
from sklearn.metrics import accuracy_score
score=accuracy_score(y_pred,y_test)
score
```

[13]: 0.9385964912280702

churn prediction using ann

[166]:
```
import pandas as pd

df=pd.read_csv('Churn_Modelling.csv')
```

[167]:
```
df.head()
```

[167]:
```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

[168]:
```
df.shape
```

[168]:
```
(10000, 14)
```

[169]:
```
df.isnull().sum()
```

[169]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

[170]:
```
y=df['Exited']
```

[171]:
```
x=df.iloc[:,3:13]
x
```

[171]:
```
      CreditScore Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
0             619    France  Female   42       2       0.00              1
1             608     Spain  Female   41       1   83807.86              1
2             502    France  Female   42       8  159660.80              3
3             699    France  Female   39       1       0.00              2
4             850     Spain  Female   43       2  125510.82              1
…             …       …   …  …   …     …          …
9995          771    France    Male   39       5       0.00              2
9996          516    France    Male   35      10   57369.61              1
9997          709    France  Female   36       7       0.00              1
9998          772   Germany    Male   42       3   75075.31              2
9999          792    France  Female   28       4  130142.79              1

      HasCrCard  IsActiveMember  EstimatedSalary
0             1               1        101348.88
1             0               1        112542.58
2             1               0        113931.57
3             0               0         93826.63
4             1               1         79084.10
…           …              …              …
9995          1               0         96270.64
9996          1               1        101699.77
9997          0               1         42085.58
9998          1               0         92888.52
9999          1               0         38190.78

[10000 rows x 10 columns]
```

[172]:
```python
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

x['Geography']=le.fit_transform(df['Geography'])
```

[173]:
```python
x['Gender']=le.fit_transform(df['Gender'])
```

[174]:
```python
x
```

[174]:
```
      CreditScore  Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
0             619          0       0   42       2       0.00              1
1             608          2       0   41       1   83807.86              1
2             502          0       0   42       8  159660.80              3
3             699          0       0   39       1       0.00              2
4             850          2       0   43       2  125510.82              1
…             …        …     …  …     …        …          …
9995          771          0       1   39       5       0.00              2
9996          516          0       1   35      10   57369.61              1
9997          709          0       0   36       7       0.00              1
```

```
9998          772          1        1   42        3   75075.31              2
9999          792          0        0   28        4  130142.79              1

      HasCrCard  IsActiveMember  EstimatedSalary
0             1               1        101348.88
1             0               1        112542.58
2             1               0        113931.57
3             0               0         93826.63
4             1               1         79084.10
…           …              …               …
9995          1               0         96270.64
9996          1               1        101699.77
9997          0               1         42085.58
9998          1               0         92888.52
9999          1               0         38190.78

[10000 rows x 10 columns]
```

[175]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

[176]:
```python
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()

x_train=ss.fit_transform(x_train)
x_test=ss.fit_transform(x_test)
x
```

[176]:
```
      CreditScore  Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
0             619          0       0   42       2       0.00              1
1             608          2       0   41       1   83807.86              1
2             502          0       0   42       8  159660.80              3
3             699          0       0   39       1       0.00              2
4             850          2       0   43       2  125510.82              1
…            …        …     …  …     …        …            …
9995          771          0       1   39       5       0.00              2
9996          516          0       1   35      10   57369.61              1
9997          709          0       0   36       7       0.00              1
9998          772          1       1   42       3   75075.31              2
9999          792          0       0   28       4  130142.79              1

      HasCrCard  IsActiveMember  EstimatedSalary
0             1               1        101348.88
1             0               1        112542.58
2             1               0        113931.57
3             0               0         93826.63
4             1               1         79084.10
```

```
…          …              …              …
9995        1              0          96270.64
9996        1              1         101699.77
9997        0              1          42085.58
9998        1              0          92888.52
9999        1              0          38190.78

[10000 rows x 10 columns]
```

[177]:

```python
import tensorflow as tf
from tensorflow import keras
```

[178]:

```python
model=keras.Sequential([
    keras.layers.Flatten(input_shape=(10,)),
    keras.layers.Dense(6,activation='relu',kernel_initializer='he_uniform'),
    keras.layers.Dense(6,activation='relu',kernel_initializer='he_uniform'),
    keras.layers.
 ↪Dense(1,activation='sigmoid',kernel_initializer='glorot_uniform')
])
```

[179]:

```python
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

[180]:

```python
history=model.fit(x_train,y_train,validation_split=0.1,epochs=50)
```

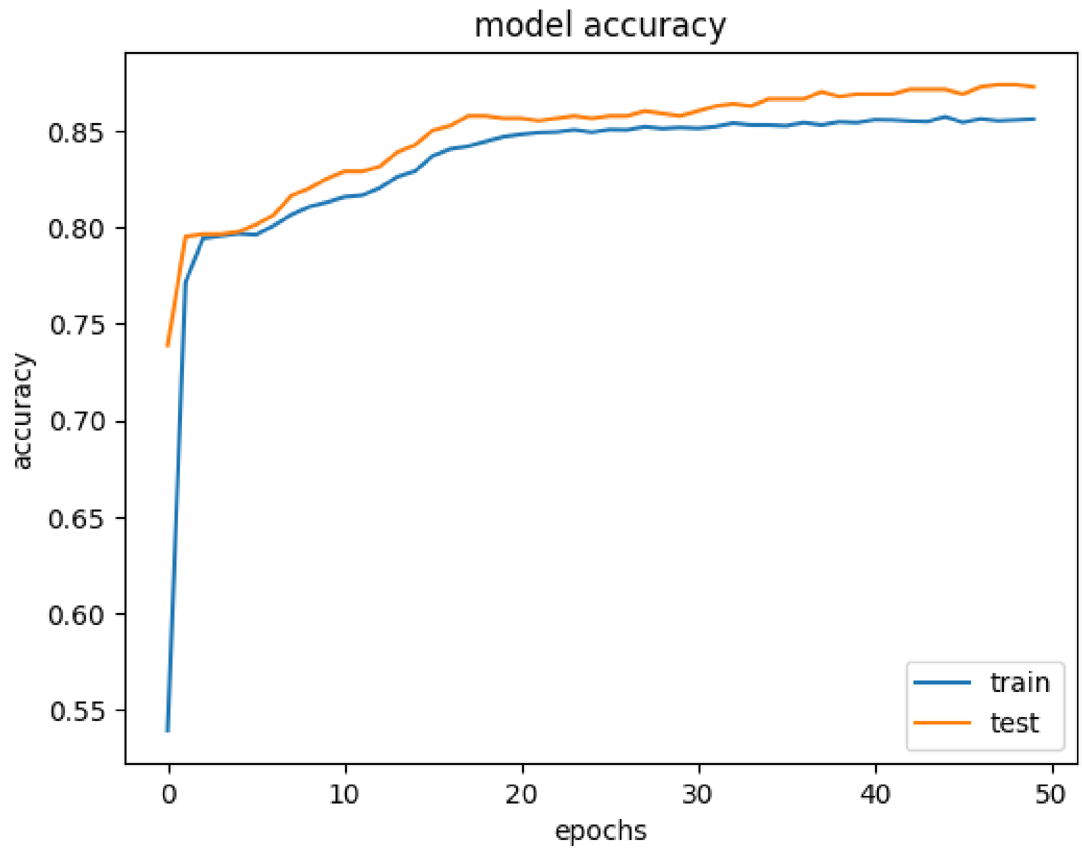
```
Epoch 1/50
225/225 [==============================] - 1s 3ms/step - loss: 0.7679 -
accuracy: 0.5394 - val_loss: 0.5948 - val_accuracy: 0.7387
Epoch 2/50
225/225 [==============================] - 1s 2ms/step - loss: 0.5472 -
accuracy: 0.7713 - val_loss: 0.5167 - val_accuracy: 0.7950
Epoch 3/50
225/225 [==============================] - 0s 2ms/step - loss: 0.5035 -
accuracy: 0.7942 - val_loss: 0.4880 - val_accuracy: 0.7962
Epoch 4/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4814 -
accuracy: 0.7954 - val_loss: 0.4706 - val_accuracy: 0.7962
Epoch 5/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4667 -
accuracy: 0.7965 - val_loss: 0.4564 - val_accuracy: 0.7975
Epoch 6/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4552 -
accuracy: 0.7960 - val_loss: 0.4439 - val_accuracy: 0.8012
Epoch 7/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4456 -
accuracy: 0.8007 - val_loss: 0.4324 - val_accuracy: 0.8062
Epoch 8/50
225/225 [==============================] - 1s 2ms/step - loss: 0.4380 -
```

```
accuracy: 0.8064 - val_loss: 0.4224 - val_accuracy: 0.8163
Epoch 9/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4321 -
accuracy: 0.8104 - val_loss: 0.4147 - val_accuracy: 0.8200
Epoch 10/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4269 -
accuracy: 0.8128 - val_loss: 0.4082 - val_accuracy: 0.8250
Epoch 11/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4218 -
accuracy: 0.8156 - val_loss: 0.4018 - val_accuracy: 0.8288
Epoch 12/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4168 -
accuracy: 0.8164 - val_loss: 0.3970 - val_accuracy: 0.8288
Epoch 13/50
225/225 [==============================] - 0s 2ms/step - loss: 0.4115 -
accuracy: 0.8203 - val_loss: 0.3915 - val_accuracy: 0.8313
Epoch 14/50
225/225 [==============================] - 1s 2ms/step - loss: 0.4058 -
accuracy: 0.8260 - val_loss: 0.3866 - val_accuracy: 0.8388
Epoch 15/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3991 -
accuracy: 0.8289 - val_loss: 0.3799 - val_accuracy: 0.8425
Epoch 16/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3913 -
accuracy: 0.8368 - val_loss: 0.3746 - val_accuracy: 0.8500
Epoch 17/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3846 -
accuracy: 0.8404 - val_loss: 0.3699 - val_accuracy: 0.8525
Epoch 18/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3794 -
accuracy: 0.8418 - val_loss: 0.3682 - val_accuracy: 0.8575
Epoch 19/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3755 -
accuracy: 0.8443 - val_loss: 0.3640 - val_accuracy: 0.8575
Epoch 20/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3722 -
accuracy: 0.8467 - val_loss: 0.3599 - val_accuracy: 0.8562
Epoch 21/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3699 -
accuracy: 0.8479 - val_loss: 0.3564 - val_accuracy: 0.8562
Epoch 22/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3674 -
accuracy: 0.8489 - val_loss: 0.3550 - val_accuracy: 0.8550
Epoch 23/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3653 -
accuracy: 0.8492 - val_loss: 0.3518 - val_accuracy: 0.8562
Epoch 24/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3640 -
```

```
accuracy: 0.8503 - val_loss: 0.3510 - val_accuracy: 0.8575
Epoch 25/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3624 -
accuracy: 0.8490 - val_loss: 0.3491 - val_accuracy: 0.8562
Epoch 26/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3609 -
accuracy: 0.8504 - val_loss: 0.3471 - val_accuracy: 0.8575
Epoch 27/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3604 -
accuracy: 0.8503 - val_loss: 0.3460 - val_accuracy: 0.8575
Epoch 28/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3597 -
accuracy: 0.8519 - val_loss: 0.3445 - val_accuracy: 0.8600
Epoch 29/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3587 -
accuracy: 0.8510 - val_loss: 0.3443 - val_accuracy: 0.8587
Epoch 30/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3579 -
accuracy: 0.8515 - val_loss: 0.3424 - val_accuracy: 0.8575
Epoch 31/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3569 -
accuracy: 0.8511 - val_loss: 0.3407 - val_accuracy: 0.8600
Epoch 32/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3558 -
accuracy: 0.8519 - val_loss: 0.3395 - val_accuracy: 0.8625
Epoch 33/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3550 -
accuracy: 0.8537 - val_loss: 0.3390 - val_accuracy: 0.8637
Epoch 34/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3544 -
accuracy: 0.8528 - val_loss: 0.3374 - val_accuracy: 0.8625
Epoch 35/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3535 -
accuracy: 0.8528 - val_loss: 0.3382 - val_accuracy: 0.8662
Epoch 36/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3532 -
accuracy: 0.8524 - val_loss: 0.3366 - val_accuracy: 0.8662
Epoch 37/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3527 -
accuracy: 0.8540 - val_loss: 0.3349 - val_accuracy: 0.8662
Epoch 38/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3520 -
accuracy: 0.8528 - val_loss: 0.3332 - val_accuracy: 0.8700
Epoch 39/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3516 -
accuracy: 0.8544 - val_loss: 0.3335 - val_accuracy: 0.8675
Epoch 40/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3508 -
```

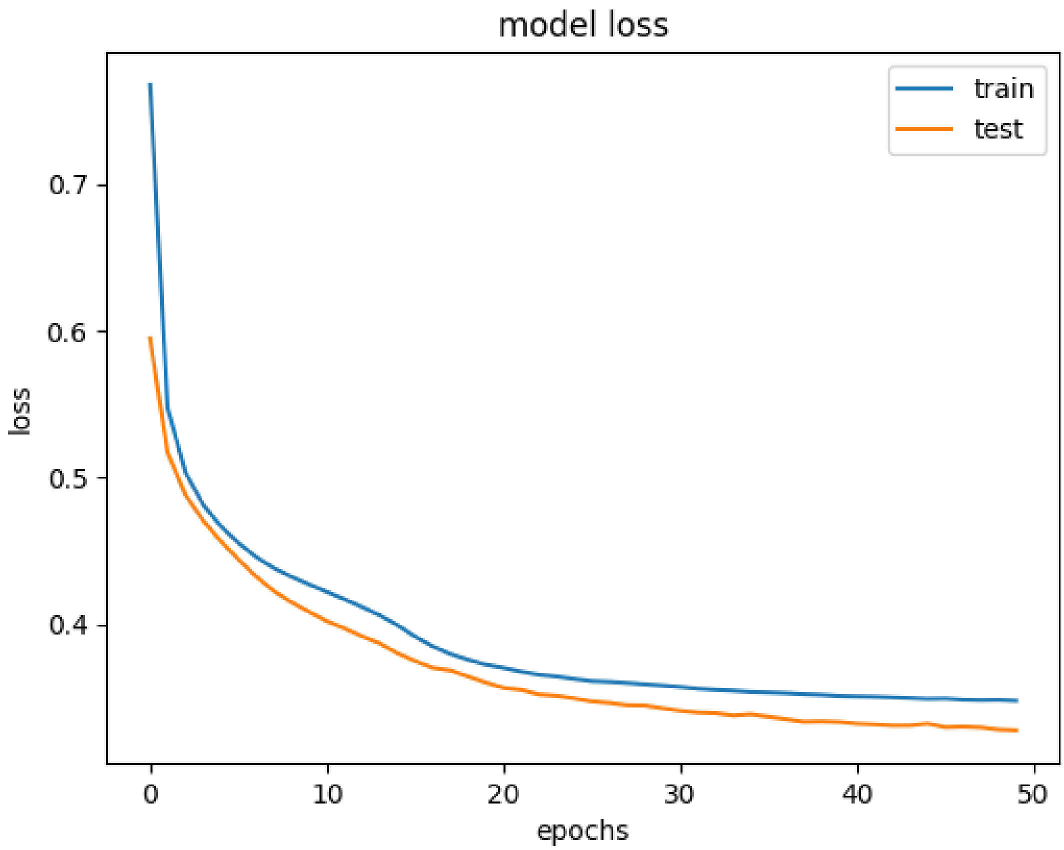
```
accuracy: 0.8540 - val_loss: 0.3330 - val_accuracy: 0.8687
Epoch 41/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3504 -
accuracy: 0.8556 - val_loss: 0.3319 - val_accuracy: 0.8687
Epoch 42/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3502 -
accuracy: 0.8554 - val_loss: 0.3314 - val_accuracy: 0.8687
Epoch 43/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3498 -
accuracy: 0.8549 - val_loss: 0.3307 - val_accuracy: 0.8712
Epoch 44/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3493 -
accuracy: 0.8546 - val_loss: 0.3307 - val_accuracy: 0.8712
Epoch 45/50
225/225 [==============================] - 0s 2ms/step - loss: 0.3488 -
accuracy: 0.8571 - val_loss: 0.3319 - val_accuracy: 0.8712
Epoch 46/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3490 -
accuracy: 0.8542 - val_loss: 0.3295 - val_accuracy: 0.8687
Epoch 47/50
225/225 [==============================] - 1s 3ms/step - loss: 0.3482 -
accuracy: 0.8560 - val_loss: 0.3298 - val_accuracy: 0.8725
Epoch 48/50
225/225 [==============================] - 1s 3ms/step - loss: 0.3480 -
accuracy: 0.8550 - val_loss: 0.3293 - val_accuracy: 0.8737
Epoch 49/50
225/225 [==============================] - 1s 2ms/step - loss: 0.3481 -
accuracy: 0.8554 - val_loss: 0.3278 - val_accuracy: 0.8737
Epoch 50/50
225/225 [==============================] - 1s 3ms/step - loss: 0.3476 -
accuracy: 0.8558 - val_loss: 0.3273 - val_accuracy: 0.8725
```

[181]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['train','test'],loc='lower right')
plt.show()
```



[182]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['train','test'],loc='upper right')
plt.show()
```



[183]:
```
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)
```

```
63/63 [==============================] - 0s 1ms/step
```

[184]:
```
from sklearn.metrics import accuracy_score
score=accuracy_score(y_pred,y_test)
score
```

[184]: 0.8665

hyper paramter tuning using keras hyper parameter tuning

[153]:
```
import pandas as pd
```

[154]:
```
df=pd.read_csv('Real_Combine.csv')
df.shape
```

[154]: (1093, 9)

[155]:
```
df=df.dropna()
```

[156]:
```
df.head()
```

[156]:

	T	TM	Tm	SLP	H	VV	V	VM	PM 2.5
0	7.4	9.8	4.8	1017.6	93.0	0.5	4.3	9.4	219.720833
1	7.8	12.7	4.4	1018.5	87.0	0.6	4.4	11.1	182.187500
2	6.7	13.4	2.4	1019.4	82.0	0.6	4.8	11.1	154.037500
3	8.6	15.5	3.3	1018.7	72.0	0.8	8.1	20.6	223.208333
4	12.4	20.9	4.4	1017.3	61.0	1.3	8.7	22.2	200.645833

[157]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

[158]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras_tuner.tuners import RandomSearch
```

[159]:
```
def hyper_tuner(param):
    model=keras.Sequential()
    for i in range(param.Int('num_layers',2,20)):
        model.add(layers.Dense(units=param.Int('units_'+str(i),
                                      min_value=32,
                                      max_value=512,
                                      step=32),
                  activation='relu'))
    model.add(layers.Dense(1,activation='linear'))
    model.compile(optimizer='adam',
                  loss='mean_absolute_error',
                  metrics=['mean_absolute_error'])
    return model
```

[160]:
```
tuner = RandomSearch(
    hyper_tuner,
    objective='val_mean_absolute_error',
    max_trials=5,
    executions_per_trial=3,
    directory='project',
    overwrite=True,
    project_name = 'Air Quality Index AQI'
)
```

[161]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 3
```

```
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 20, 'step': 1,
'sampling': 'linear'}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step':
32, 'sampling': 'linear'}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step':
32, 'sampling': 'linear'}
```

[162]:
```
from sklearn.model_selection import train_test_split as tts
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪3,random_state=0)
```

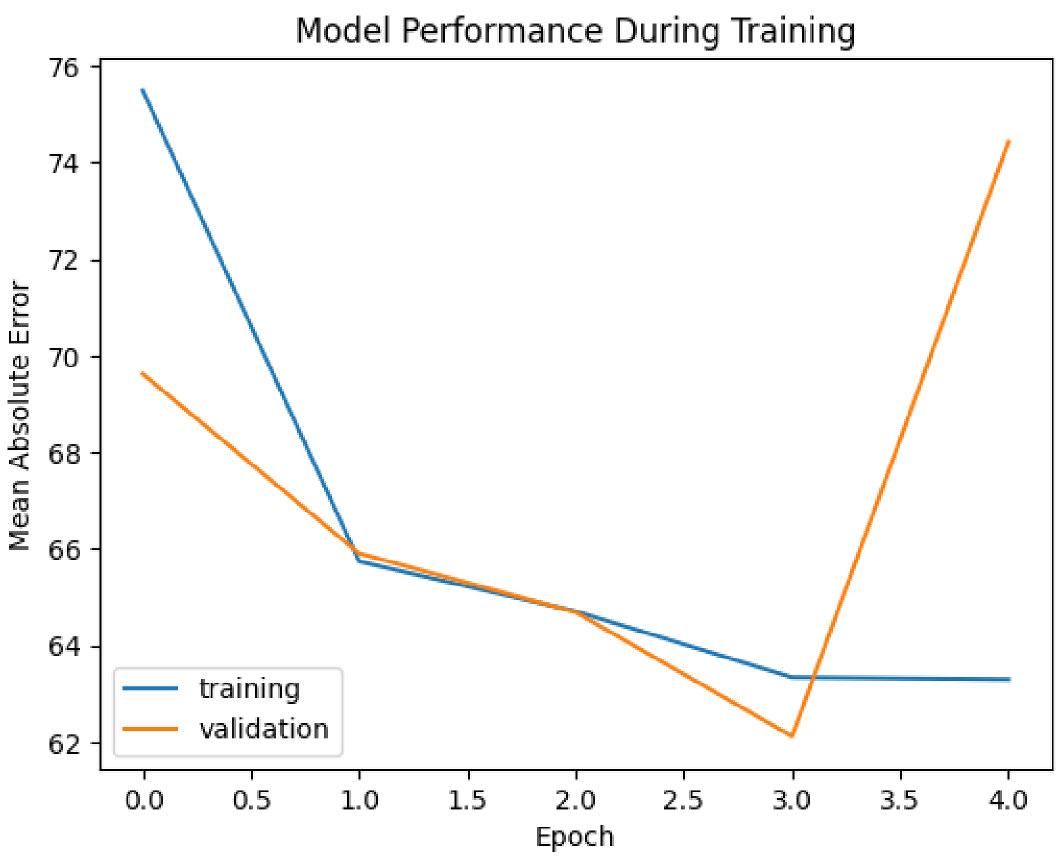
[163]:
```
tuner.search(x_train , y_train , epochs=5, validation_data=(x_test,y_test))
```

```
Trial 5 Complete [00h 00m 12s]
val_mean_absolute_error: 61.63383865356445

Best val_mean_absolute_error So Far: 59.83472188313802
Total elapsed time: 00h 01m 28s
```

[164]:
```
import matplotlib.pyplot as plt

%matplotlib inline

#Get the best Hyperparameters found during the search
best_hps = tuner.get_best_hyperparameters(1)[0]

#Build the Model witht he best hyperparameters
model=hyper_tuner(best_hps)

#Train the model with the best hyperparameters on the full training set
history = model.fit(x_train,y_train , epochs=5 ,validation_data =␣
↪(x_test,y_test))

#Plot the Training and Validation Metrics for each Epoch
plt.plot(history.history['mean_absolute_error'] , label='training')
plt.plot(history.history['val_mean_absolute_error'] , label='validation')
plt.title('Model Performance During Training')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.show()
```

```
Epoch 1/5
24/24 [==============================] - 3s 21ms/step - loss: 75.4888 -
mean_absolute_error: 75.4888 - val_loss: 69.6223 - val_mean_absolute_error:
```

```
69.6223
Epoch 2/5
24/24 [==============================] - 0s 7ms/step - loss: 65.7399 -
mean_absolute_error: 65.7399 - val_loss: 65.9056 - val_mean_absolute_error:
65.9056
Epoch 3/5
24/24 [==============================] - 0s 8ms/step - loss: 64.7094 -
mean_absolute_error: 64.7094 - val_loss: 64.6912 - val_mean_absolute_error:
64.6912
Epoch 4/5
24/24 [==============================] - 0s 7ms/step - loss: 63.3427 -
mean_absolute_error: 63.3427 - val_loss: 62.1212 - val_mean_absolute_error:
62.1212
Epoch 5/5
24/24 [==============================] - 0s 7ms/step - loss: 63.3008 -
mean_absolute_error: 63.3008 - val_loss: 74.4264 - val_mean_absolute_error:
74.4264
```



[165]:
```
best_hps.values
```

[165]:
```
{'num_layers': 8,
 'units_0': 416,
 'units_1': 224,
 'units_2': 192,
 'units_3': 352,
 'units_4': 64,
 'units_5': 288,
 'units_6': 416,
 'units_7': 256,
 'units_8': 128,
 'units_9': 128,
 'units_10': 352,
 'units_11': 352,
 'units_12': 480,
 'units_13': 128,
 'units_14': 288,
 'units_15': 192,
 'units_16': 256,
 'units_17': 192,
 'units_18': 320,
 'units_19': 192}
```

Classification of MNIST data using ANN

[185]:
```
import tensorflow as tf
from tensorflow import keras
```

[186]:
```
(x_train, y_train),(x_test,y_test)=keras.datasets.mnist.load_data()
```

[187]:
```
x_train=x_train/255.0
x_test=x_test/255.0
```

[188]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),  # Convert the 28x28 Image into␣
 ↪a 1D Array
    keras.layers.Dense(128,activation='relu'),  # Hidden Layer with 128 Units
    keras.layers.Dense(10,activation='softmax')  #Output Layer with 10 units
])
```

[189]:
```
model.compile(optimizer='adam' , loss='sparse_categorical_crossentropy' ,␣
 ↪metrics=['accuracy'])
```

[190]:
```
history = model.fit(x_train,y_train,epochs=5,validation_data=(x_test,y_test))
```

```
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2618 -
accuracy: 0.9245 - val_loss: 0.1456 - val_accuracy: 0.9569
Epoch 2/5
```

```
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1166 -
accuracy: 0.9655 - val_loss: 0.1107 - val_accuracy: 0.9666
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0791 -
accuracy: 0.9758 - val_loss: 0.0857 - val_accuracy: 0.9738
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0600 -
accuracy: 0.9815 - val_loss: 0.0842 - val_accuracy: 0.9737
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0465 -
accuracy: 0.9855 - val_loss: 0.0723 - val_accuracy: 0.9783
```

[191]:
```
# Predict the Labels of the test Set
import numpy as np
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
y_pred
```

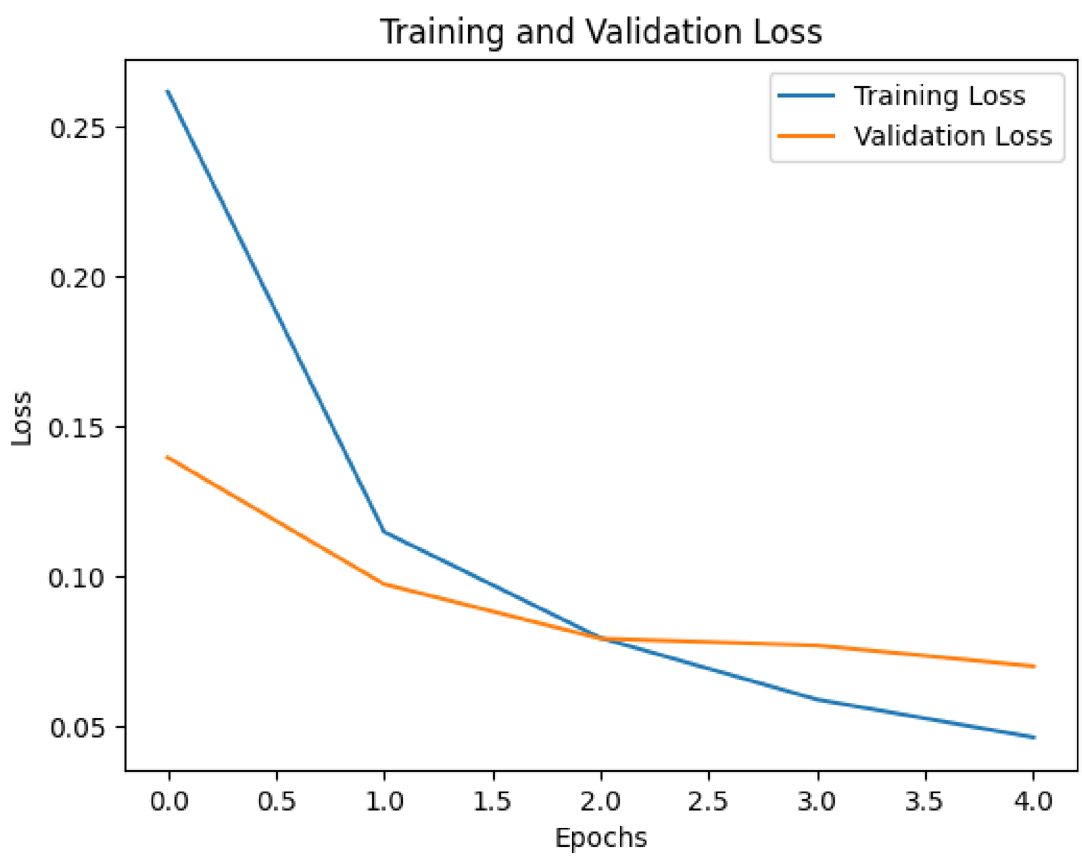
```
313/313 [==============================] - 1s 2ms/step
```

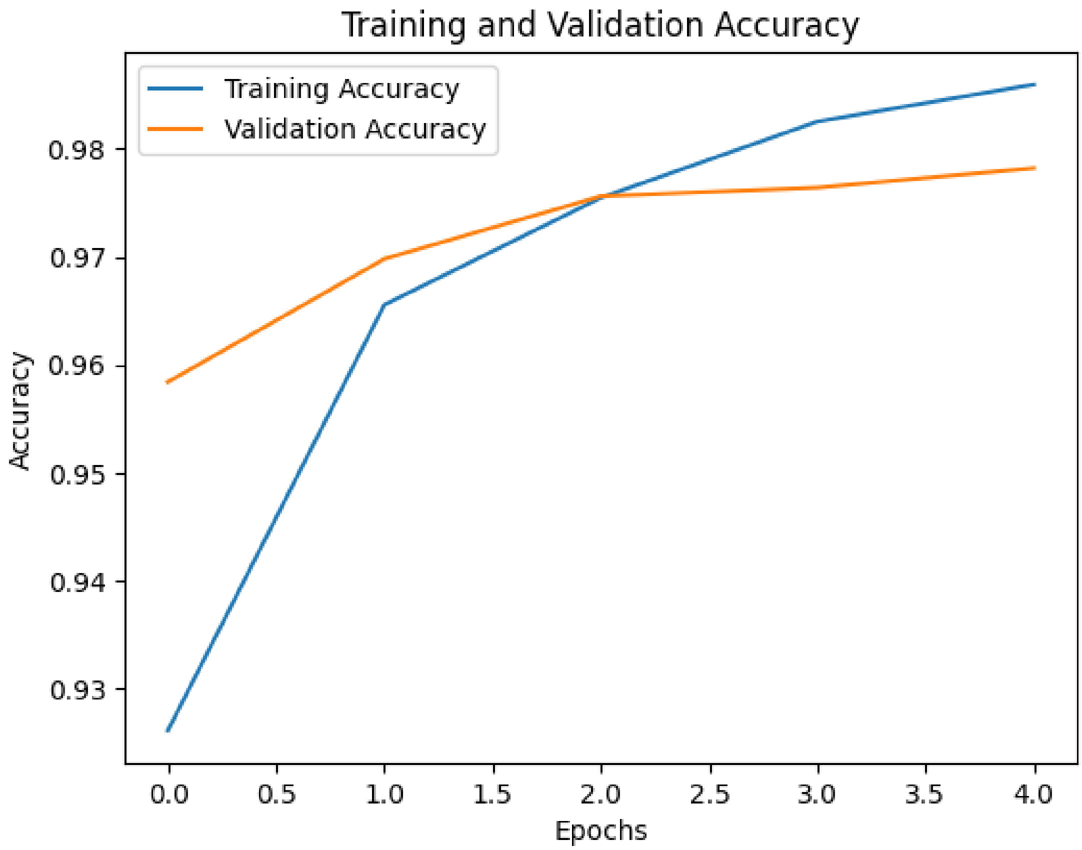
[191]:
```
array([7, 2, 1, …, 4, 5, 6], dtype=int64)
```

[192]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_pred,y_test)
# Print the Confusion Matrix
print('Confusion Matrix')
print(cm)

# Calculate the Accuracy
acc=accuracy_score(y_pred,y_test)

# Printing the Accuracy
print('Accuracy :',acc)
```

```
Confusion Matrix
[[ 963    0    3    0    0    2    2    0    2    1]
 [   0 1125    1    0    0    0    4    4    0    2]
 [   1    1 1005    3    1    0    2    9    2    0]
 [   1    1    2  987    0    6    1    1    3    3]
 [   2    0    2    0  949    1    4    0    4    6]
 [   3    1    0    4    0  869    2    0    4    1]
 [   4    3    2    0    4    2  938    0    1    0]
 [   1    1    8    5    3    2    1 1006    3    4]
 [   3    3    8    6    1    6    4    3  951    2]
 [   2    0    1    5   24    4    0    5    4  990]]
Accuracy : 0.9783
```

[54]:
```
import matplotlib.pyplot as plt

# Plotting the Training and Validation Loss
plt.plot(history.history['loss'] , label='Training Loss')
plt.plot(history.history['val_loss'] , label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


# Plotting the Training and Validation Loss
plt.plot(history.history['accuracy'] , label='Training Accuracy')
plt.plot(history.history['val_accuracy'] , label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



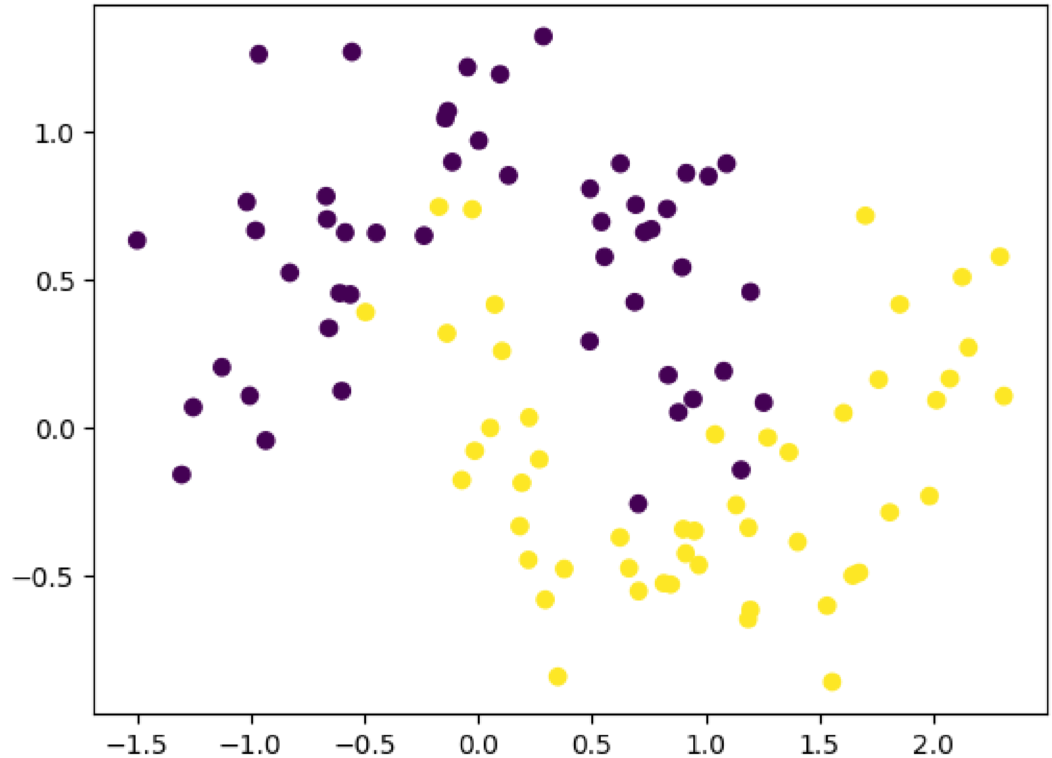


dropout layer

[193]:
```
import numpy as np
from sklearn.datasets import make_moons
```

[194]:
```
X, y = make_moons(100, noise=0.25, random_state=2)
```

[195]:
```
import matplotlib.pyplot as plt

plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```



[196]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras import layers

model = Sequential([
    layers.Dense(128, input_dim=2, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

[197]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 128)               384

 dropout (Dropout)           (None, 128)               0
```

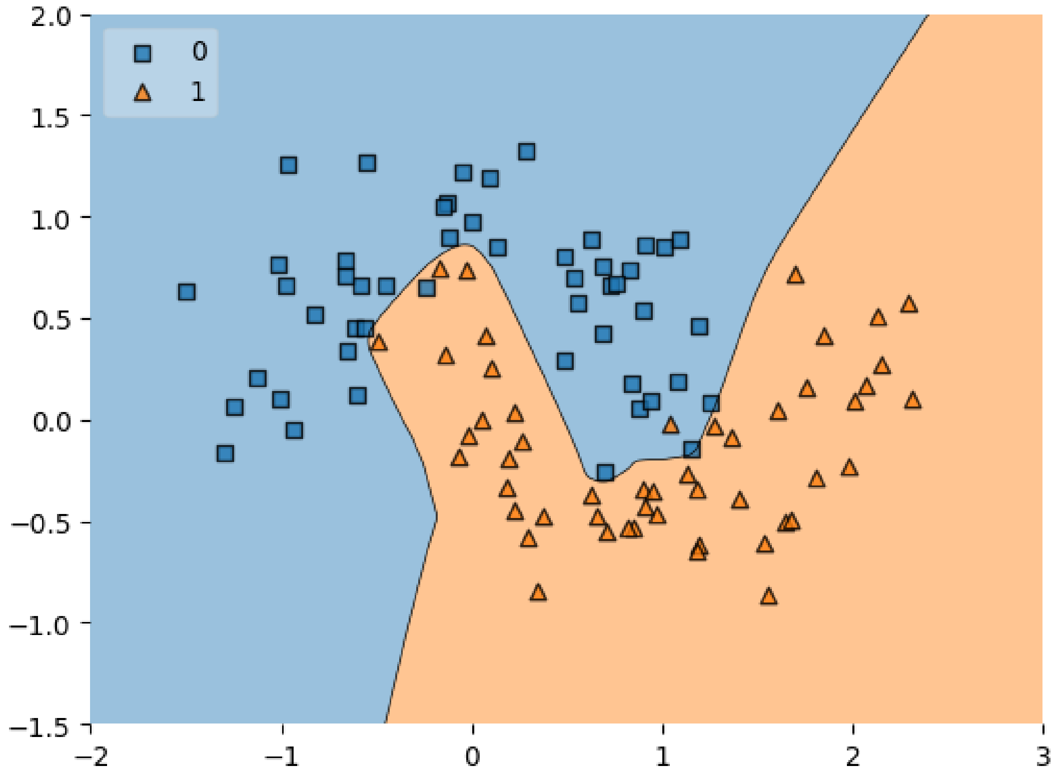
```
 dense_23 (Dense)            (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_24 (Dense)            (None, 1)                 129

=================================================================
Total params: 17025 (66.50 KB)
Trainable params: 17025 (66.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[198]:
```
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

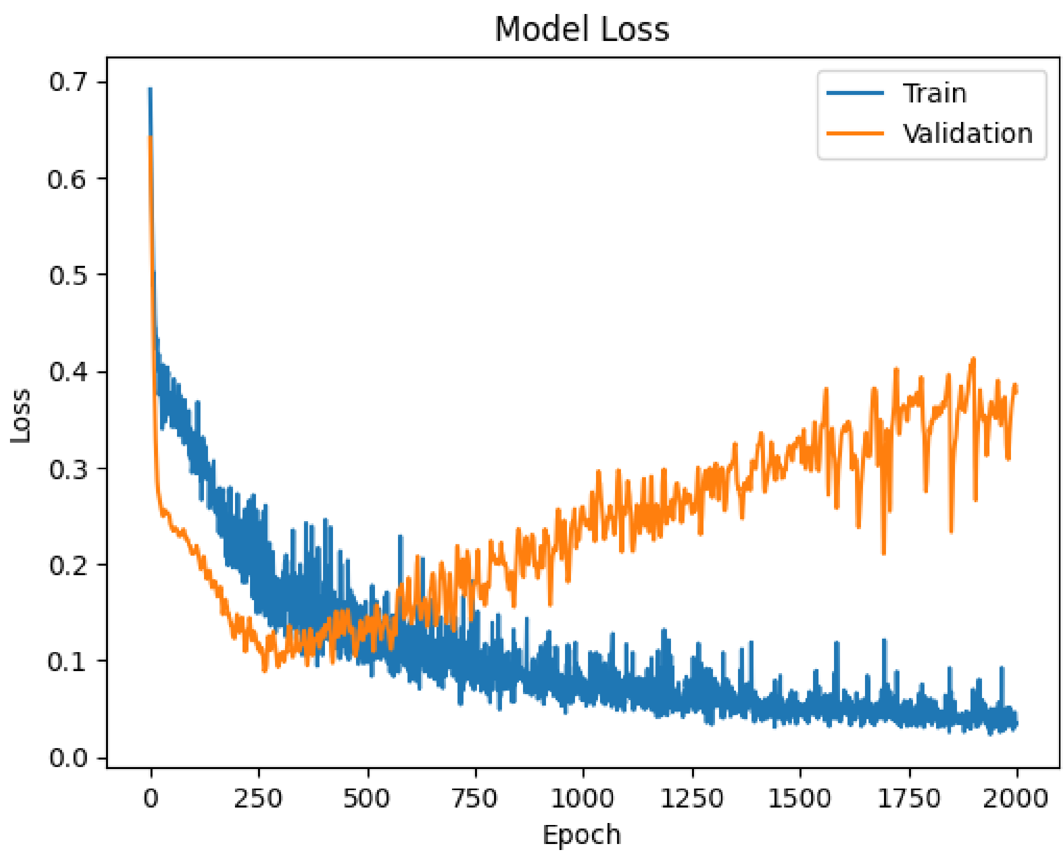
[199]:
```
history = model.fit(X, y, epochs=2000, validation_split=0.2, verbose=0)
```

[200]:
```
# Visualize the decision boundary
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y.astype('int'), clf=model, legend=2)
plt.xlim(-2,3)
plt.ylim(-1.5,2)
plt.show()
```

```
9600/9600 [==============================] - 21s 2ms/step
```



[201]:
```
# Plot the loss curve
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```



[202]:
```
# Calculation of accuarcy of each model
# Calculate the accuracy for model1
acc_model1 = history.history['accuracy'][-1] * 100
acc_model1
```

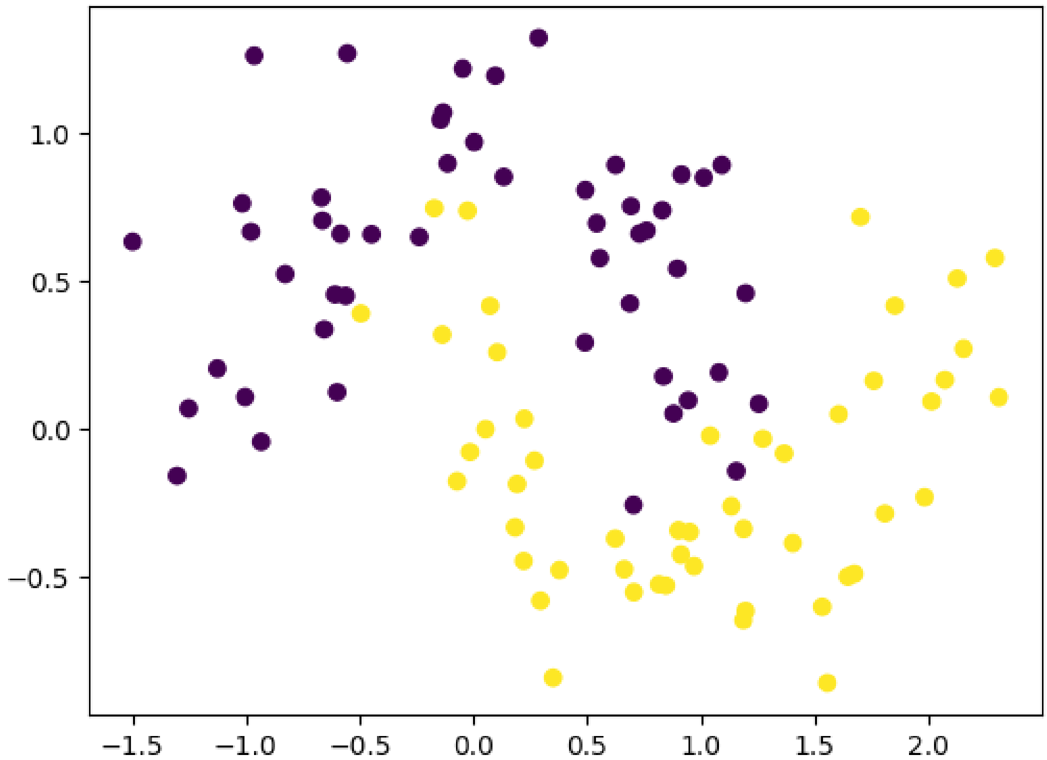
[202]: 97.50000238418579

regularization techniques

[65]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
import seaborn as sns
from mlxtend.plotting import plot_decision_regions
import tensorflow
from tensorflow import keras
from keras.models import Sequential
from keras import layers
import visualkeras
```

[66]:
```
X, y = make_moons(100, noise=0.25,random_state=2) # toy dataset with 2 features:
↪ 100 samples
```

[67]:
```
plt.scatter(X[:,0], X[:,1], c=y) # to generates different colors with binary␣
↪values in data
plt.show()
```



[68]:
```
X.shape
```

[68]: (100, 2)

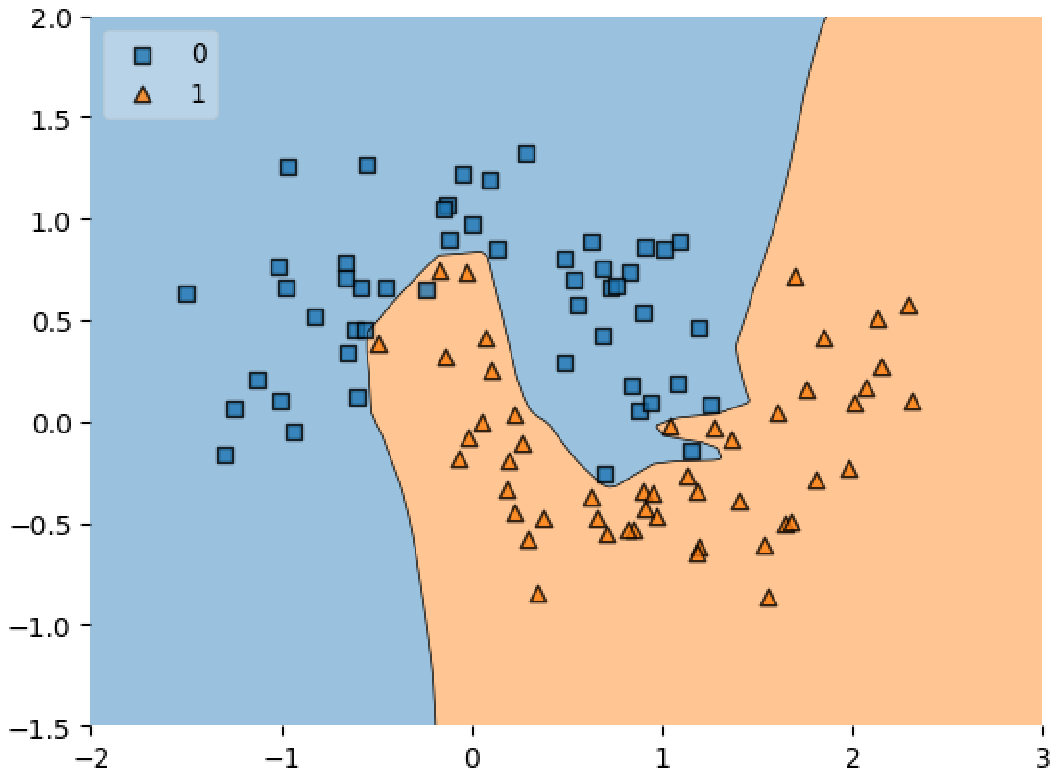
[69]:
```
model1 = Sequential([
    layers.Dense(128, input_dim=2, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(1, activation='sigmoid')
])
```

[70]:
```
model1.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
history1 = model1.fit(X, y, epochs=2000, validation_split =0.2,verbose=0)
```

[71]:

```
plot_decision_regions(X, y.astype('int'), clf=model1, legend=2) # X is for
↪input data, y=integer labels, clf=model1 trained classifier, legend=2
↪location of legend point
plt.xlim(-2,3)
plt.ylim(-1.5,2)
plt.show()
```

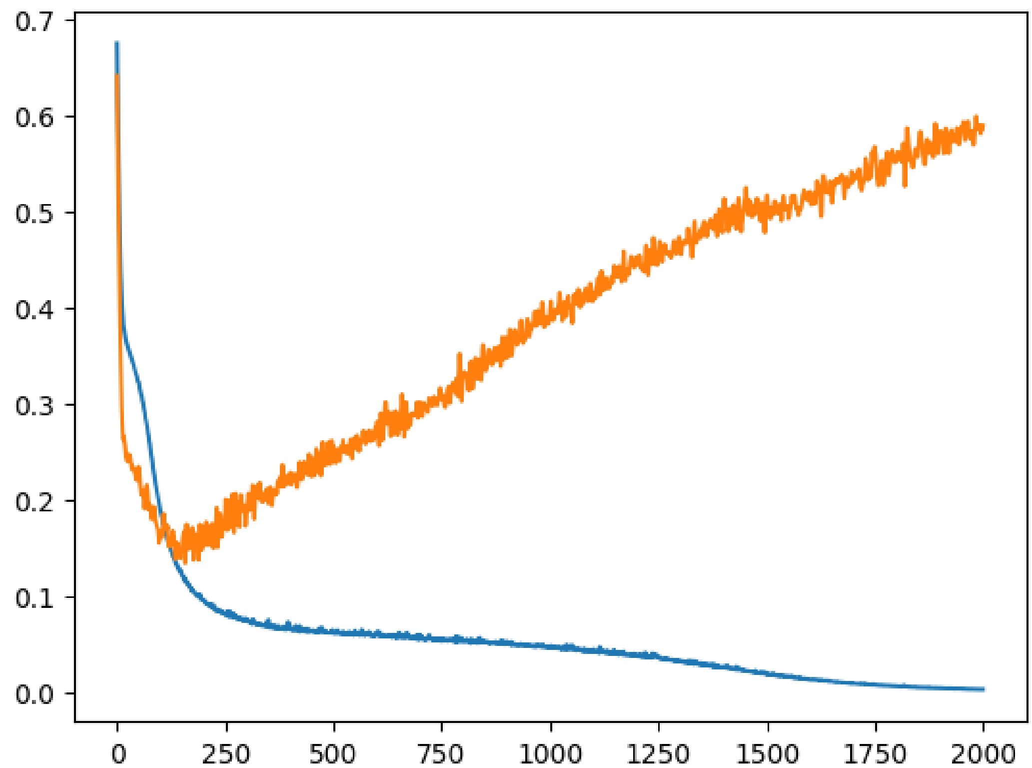
```
9600/9600 [==============================] - 24s 3ms/step
```



[72]:

```
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
```

[72]: [<matplotlib.lines.Line2D at 0x1a5709034f0>]



[75]:
```
model2 = Sequential([
    layers.Dense(128,input_dim=2,␣
 ↪activation="relu",kernel_regularizer=tensorflow.keras.regularizers.l2(0.
 ↪001)),
    layers.Dense(128, activation="relu",kernel_regularizer=tensorflow.keras.
 ↪regularizers.l1(0.001)),
    layers.Dense(1,activation='sigmoid')
])
```

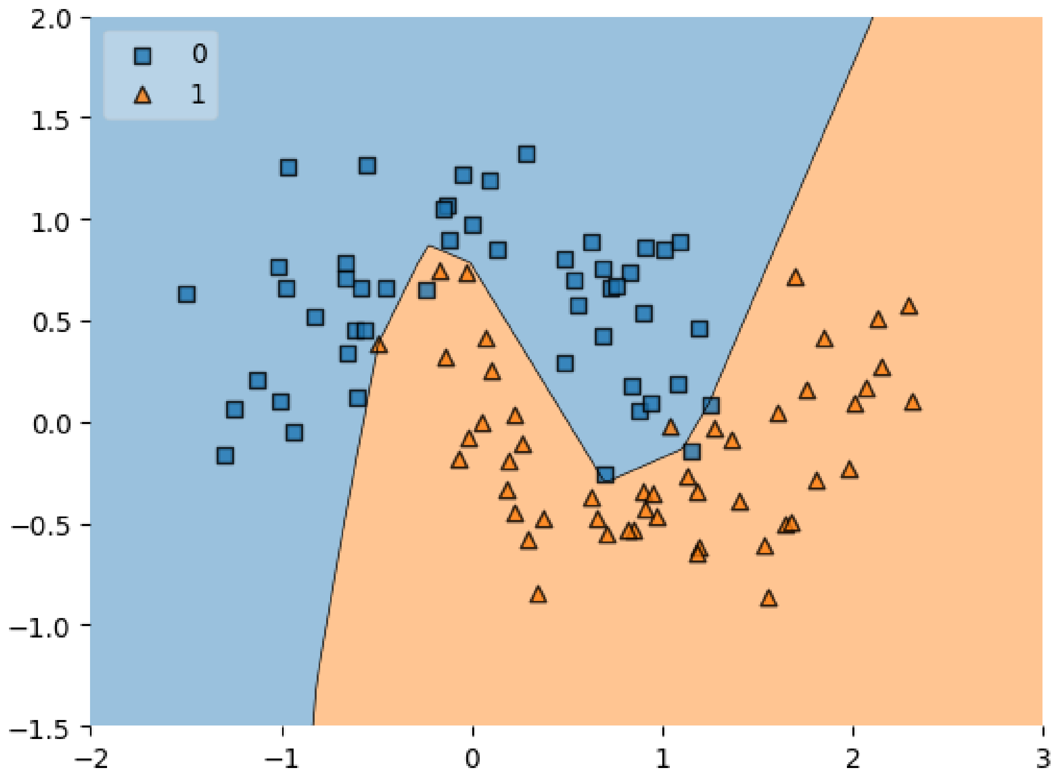
[76]:
```
model2.summary()
visualkeras.layered_view(model2).show()
```

```
Model: "sequential_5"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_19 (Dense)            (None, 128)               384

 dense_20 (Dense)            (None, 128)               16512

 dense_21 (Dense)            (None, 1)                 129
```

```
=================================================================
Total params: 17025 (66.50 KB)
Trainable params: 17025 (66.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

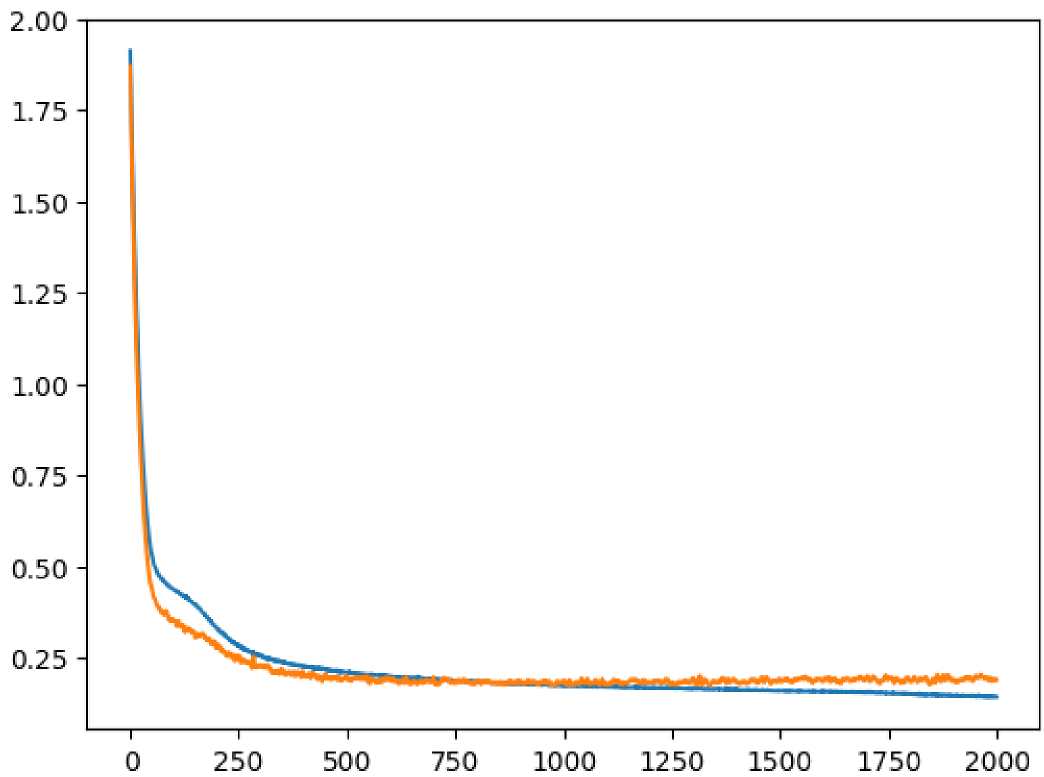
[77]:
```
model2.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
history2 = model2.fit(X, y, epochs=2000, validation_split =0.2,verbose=0)
plot_decision_regions(X, y.astype('int'), clf=model2, legend=2)
plt.xlim(-2,3)
plt.ylim(-1.5,2)
plt.show()
```

```
9600/9600 [==============================] - 22s 2ms/step
```



[78]:
```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
```

[78]: [<matplotlib.lines.Line2D at 0x1a5704c1e20>]



[79]:
```
# Calculation of accuarcy of each model
# Calculate the accuracy for model1
acc_model1 = history1.history['accuracy'][-1] * 100
# Calculate the accuracy for model2
acc_model2 = history2.history['accuracy'][-1] * 100
print(f"Accuracy for Model 1: {acc_model1:.2f}%")
print(f"Accuracy for Model 2: {acc_model2:.2f}%")
```

```
Accuracy for Model 1: 100.00%
Accuracy for Model 2: 95.00%
```

Prediction of sentiments using ANN

[212]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
```

[213]:
```
df=pd.read_csv("sentiment.csv")
df.head()
```

[213]:
```
   Index                                     message to examine  \
0    106  just had a real good moment. i missssssssss hi…
1    217         is reading manga  http://plurk.com/p/mzp1e
2    220  @comeagainjen http://twitpic.com/2y2lx - http:…
3    288  @lapcat Need to send 'em to my accountant tomo…
4    540      ADD ME ON MYSPACE!!!  myspace.com/LookThunder

   label (depression result)
0                          0
1                          0
2                          0
3                          0
4                          0
```

[214]:
```
df.shape
```

[214]:
```
(10314, 3)
```

[215]:
```
df.isnull().sum()
```

[215]:
```
Index                        0
message to examine           0
label (depression result)    0
dtype: int64
```

[216]:
```
#encoding
tfid=TfidfVectorizer(max_features=5000)
x=tfid.fit_transform(df["message to examine"])
y=df['label (depression result)']
```

[217]:
```
x=x.toarray()
```

[218]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪2,random_state=42)
```

[227]:
```
(x_train.shape[1],)
```

[227]:
```
(5000,)
```

[220]:
```
model=keras.Sequential([ keras.layers.Dense(128,activation='relu',input_shape␣
 ↪=(x_train.shape[1],)),
                        keras.layers.Dense(64,activation='relu'),
                        keras.layers.Dense(64,activation='relu'),
                        keras.layers.Dense(64,activation='relu'),
```

```
                        keras.layers.Dense(64,activation='relu'),
                        keras.layers.Dense(1,activation='sigmoid'),

])
#compile
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

[228]:
```
history=model.
 ↪fit(x_train,y_train,epochs=5,batch_size=16,validation_data=(x_test,y_test))
```

```
Epoch 1/5
516/516 [==============================] - 5s 5ms/step - loss: 0.1602 -
accuracy: 0.9349 - val_loss: 0.0559 - val_accuracy: 0.9859
Epoch 2/5
516/516 [==============================] - 3s 5ms/step - loss: 0.0089 -
accuracy: 0.9973 - val_loss: 0.0633 - val_accuracy: 0.9816
Epoch 3/5
516/516 [==============================] - 3s 6ms/step - loss: 0.0024 -
accuracy: 0.9998 - val_loss: 0.0769 - val_accuracy: 0.9845
Epoch 4/5
516/516 [==============================] - 3s 7ms/step - loss: 0.0020 -
accuracy: 0.9998 - val_loss: 0.0817 - val_accuracy: 0.9825
Epoch 5/5
516/516 [==============================] - 4s 7ms/step - loss: 0.0017 -
accuracy: 0.9998 - val_loss: 0.0747 - val_accuracy: 0.9850
```

[222]:
```
#evaluate the model on the set
test_loss,test_acc=model.evaluate(x_test,y_test,verbose=0)
test_loss
```

[222]: 0.6925400495529175

[223]:
```
test_acc
```

[223]: 0.6873485445976257

[26]:
```
#save the model
#model.save('senti.keras')
```

[106]:
```
#load the save model
#loaded_model=keras.models.load_model('senti.keras')
```

[107]:
```
#loaded_model
```

Prediction of MNSIT using CNN

[15]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
```

[16]:
```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

[17]:
```
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

[18]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

[19]:
```
# Build the CNN model
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax'),
])
```

[20]:
```
model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[21]:
```
history=model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.
 ↪2)
```

```
Epoch 1/5
750/750 [==============================] - 15s 18ms/step - loss: 0.2057 -
accuracy: 0.9388 - val_loss: 0.1025 - val_accuracy: 0.9693
Epoch 2/5
750/750 [==============================] - 13s 17ms/step - loss: 0.0593 -
accuracy: 0.9814 - val_loss: 0.0514 - val_accuracy: 0.9848
Epoch 3/5
750/750 [==============================] - 12s 16ms/step - loss: 0.0421 -
accuracy: 0.9861 - val_loss: 0.0503 - val_accuracy: 0.9857
Epoch 4/5
750/750 [==============================] - 13s 17ms/step - loss: 0.0320 -
accuracy: 0.9897 - val_loss: 0.0410 - val_accuracy: 0.9877
Epoch 5/5
750/750 [==============================] - 12s 17ms/step - loss: 0.0253 -
accuracy: 0.9918 - val_loss: 0.0364 - val_accuracy: 0.9881
```

[22]:

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test Accuracy: {test_acc}')
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0309 -
accuracy: 0.9910
Test Accuracy: 0.9909999966621399
```

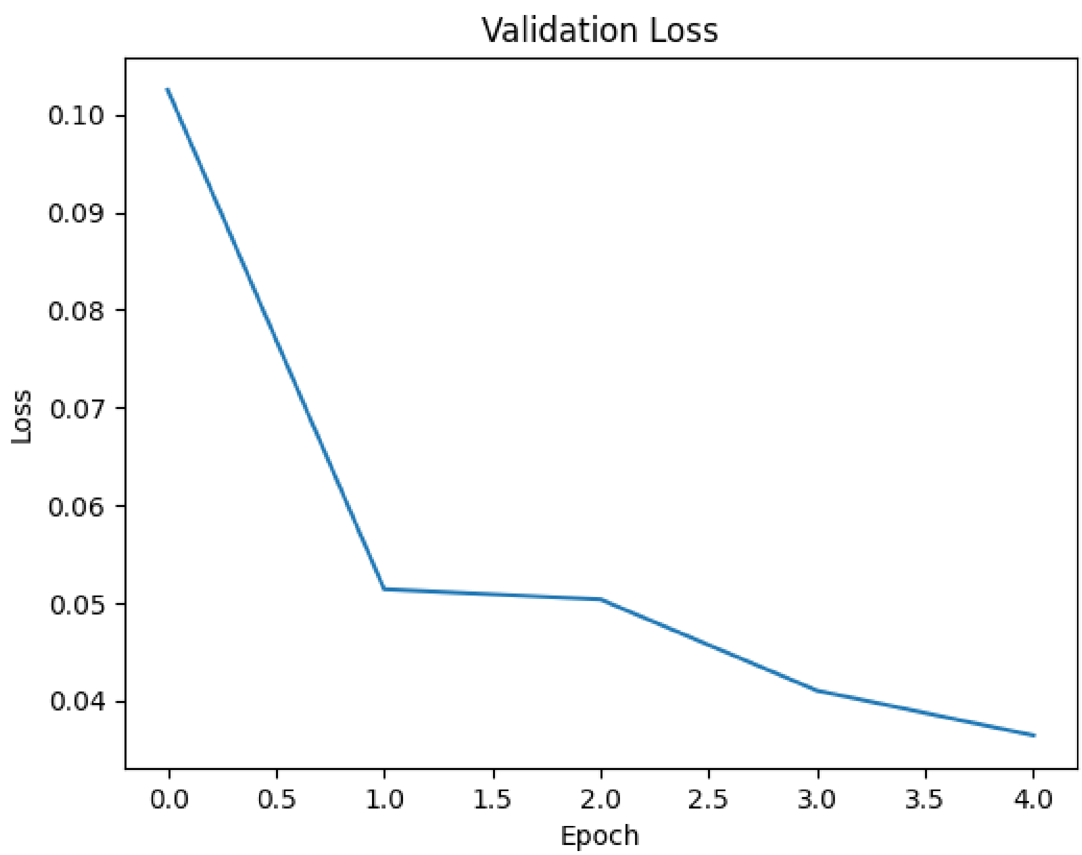
[23]:

```
print(f'Test Loss: {test_loss}')
```

```
Test Loss: 0.030942896381020546
```

[24]:

```
import matplotlib.pyplot as plt
plt.plot(history.history['val_loss'])
plt.title('Validation Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



[25]:

```
plt.plot(history.history['val_accuracy'])
plt.title('Validation accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()
```