Neural Networks

Ninjaz

Agenda

1. Heart Disease Dataset
2. Neural Networks Models
3. Comparison of the best NN model results
4. Machine Learning Models
5. Comparison of the best ML model results

Datatset

Heart Disease Dataset

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V.

It contains 12 attributes and 918 records

The "target" field refers to the presence of heart disease in the patient. It is integer-valued 0 = no disease and 1 = disease

Data columns (total 12 columns):

#	Column	Non-Null Count	Dtype
0	Age	918 non-null	int64
1	Sex	918 non-null	object
2	ChestPainType	918 non-null	object
3	RestingBP	918 non-null	int64
4	Cholesterol	918 non-null	int64
5	FastingBS	918 non-null	int64
6	RestingECG	918 non-null	object
7	MaxHR	918 non-null	int64
8	ExerciseAngina	918 non-null	object
9	Oldpeak	918 non-null	float64
10	ST_Slope	918 non-null	object
11	HeartDisease	918 non-null	int64

Deep learning - ANN models

First NN Model

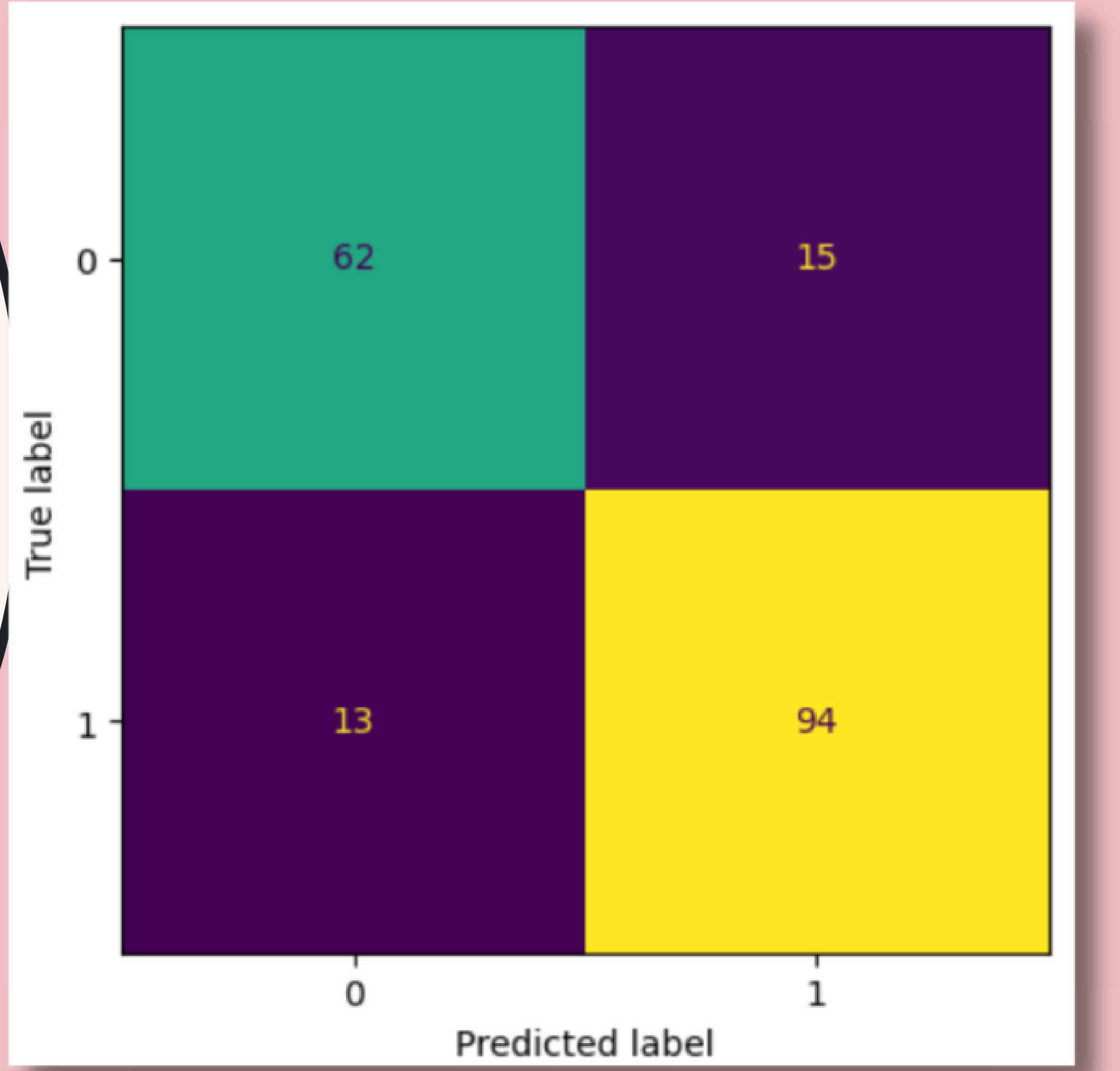
Two layers

First layer: 100 units   Activation='LeakyReLU'

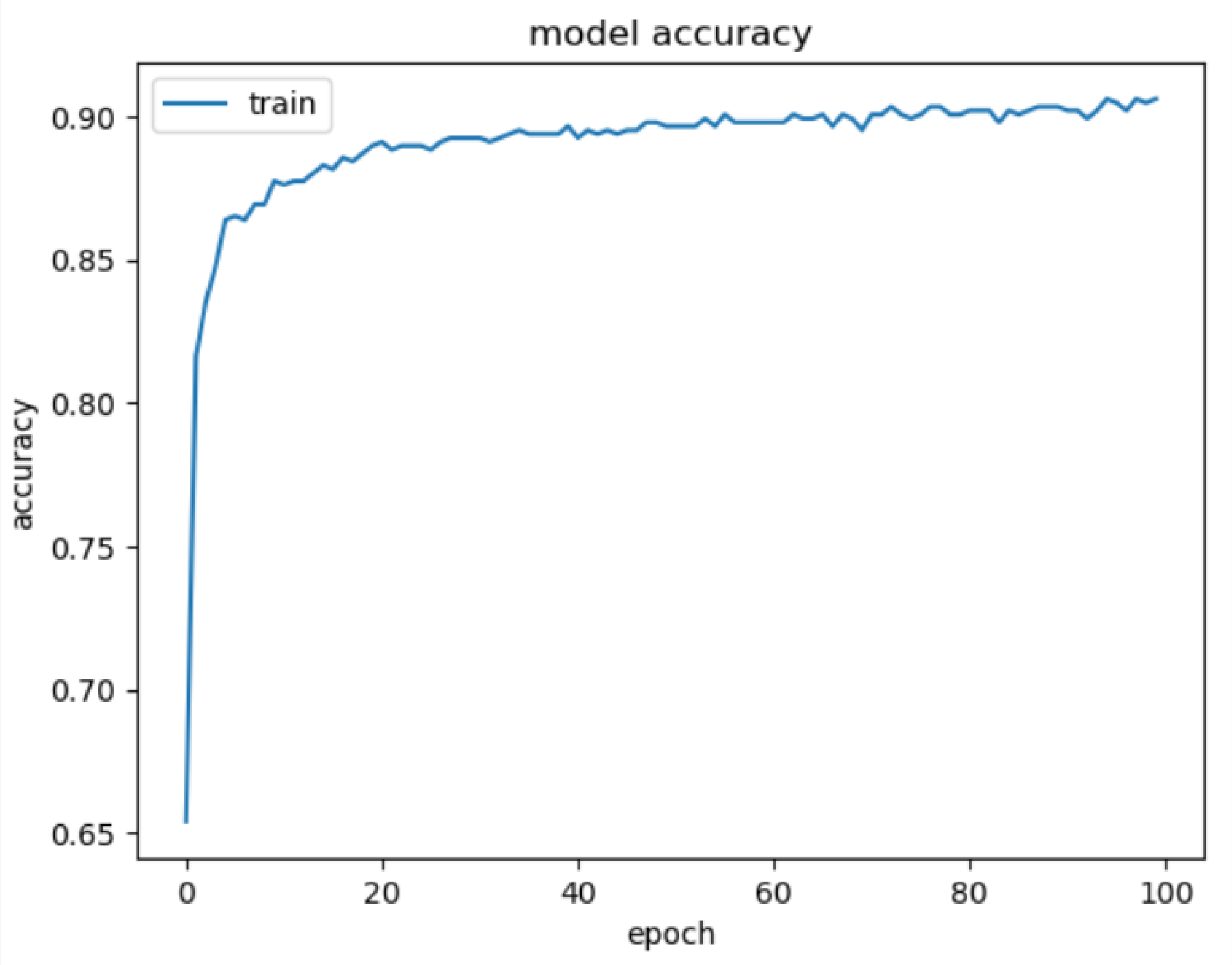
Second layer : 100 units   Activation='LeakyReLU'

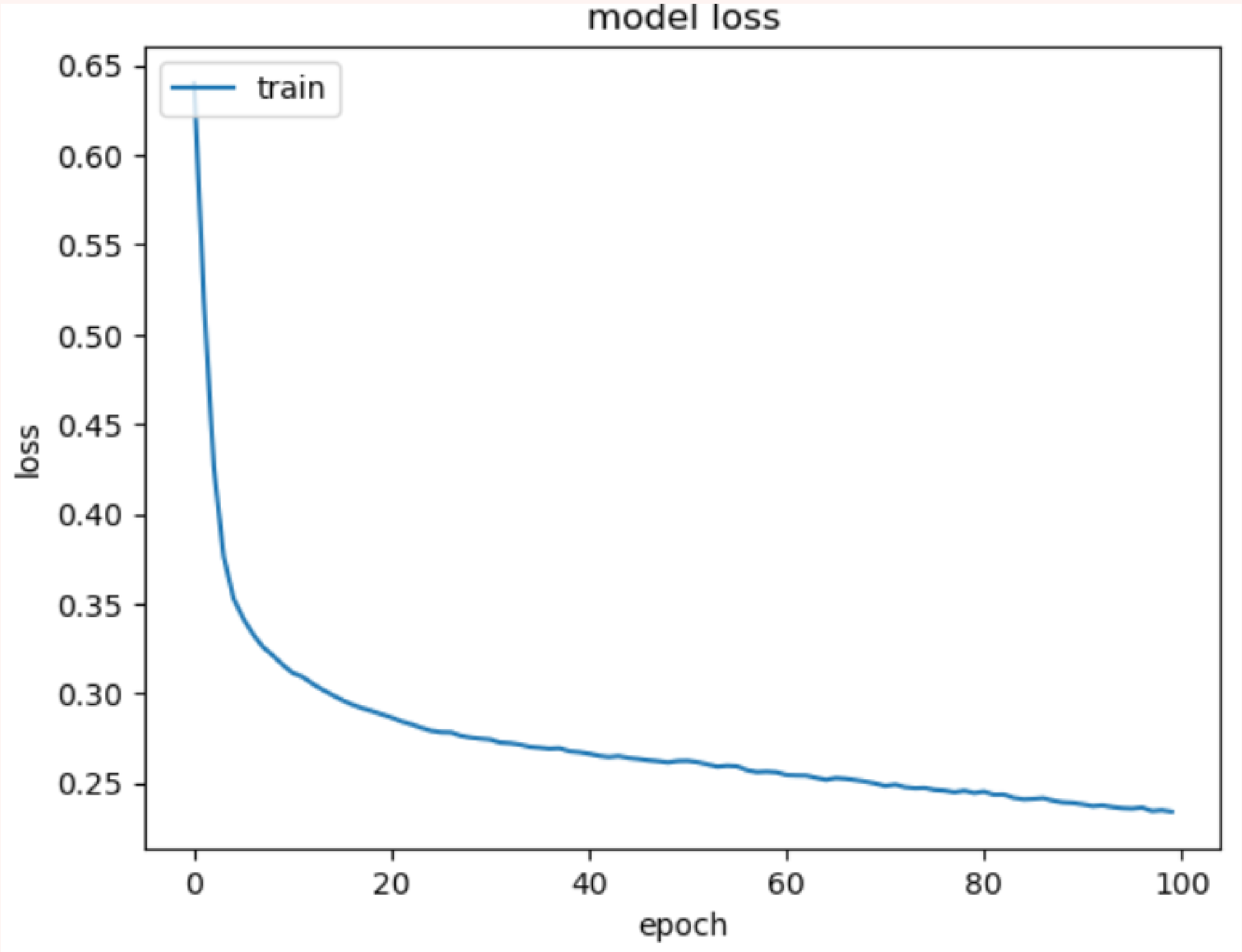
Output layer :one unit

Activation Function 'sigmoid'



Plot the model training history





Second NN Model

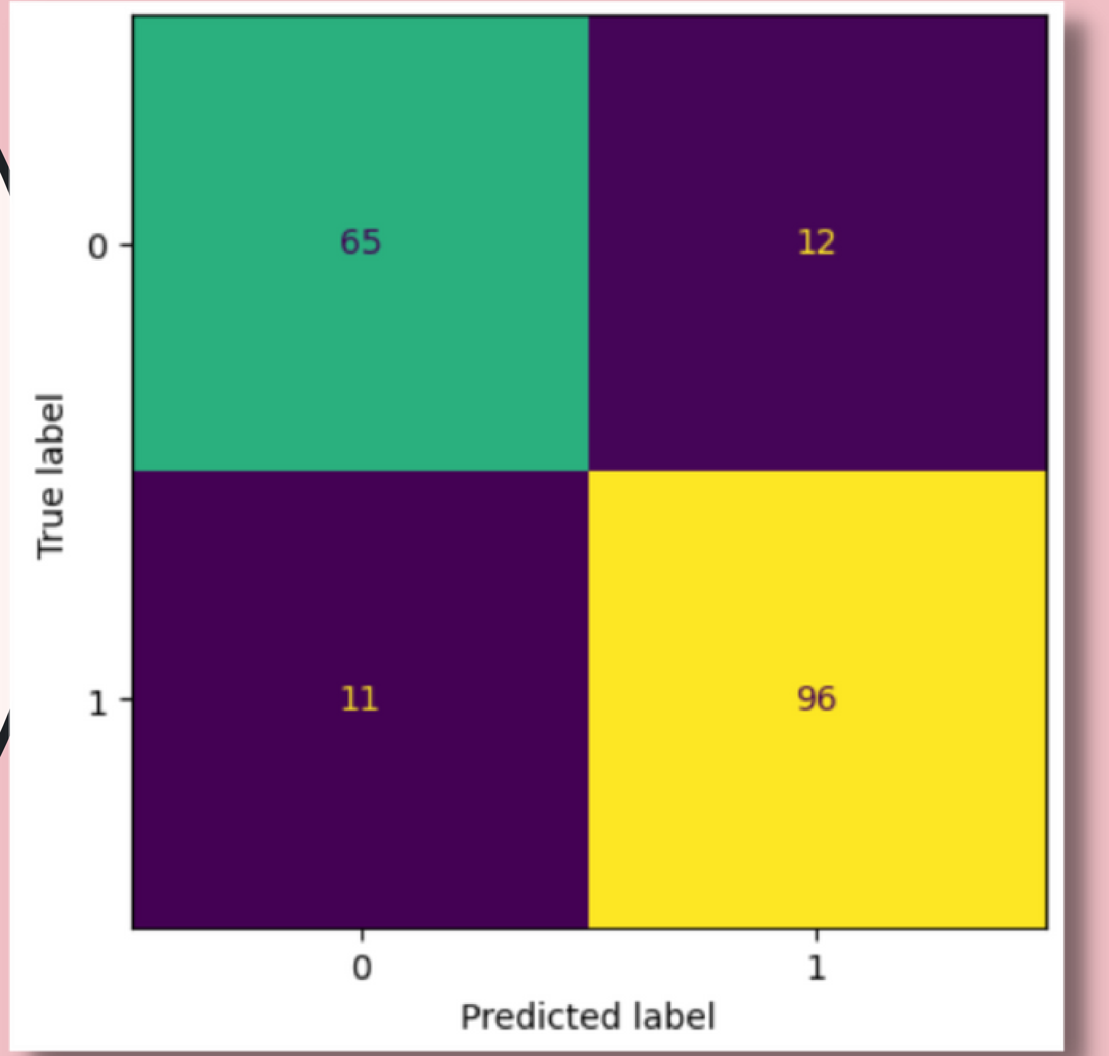
Two layers

First layer: 6 units   Activation='Relu'

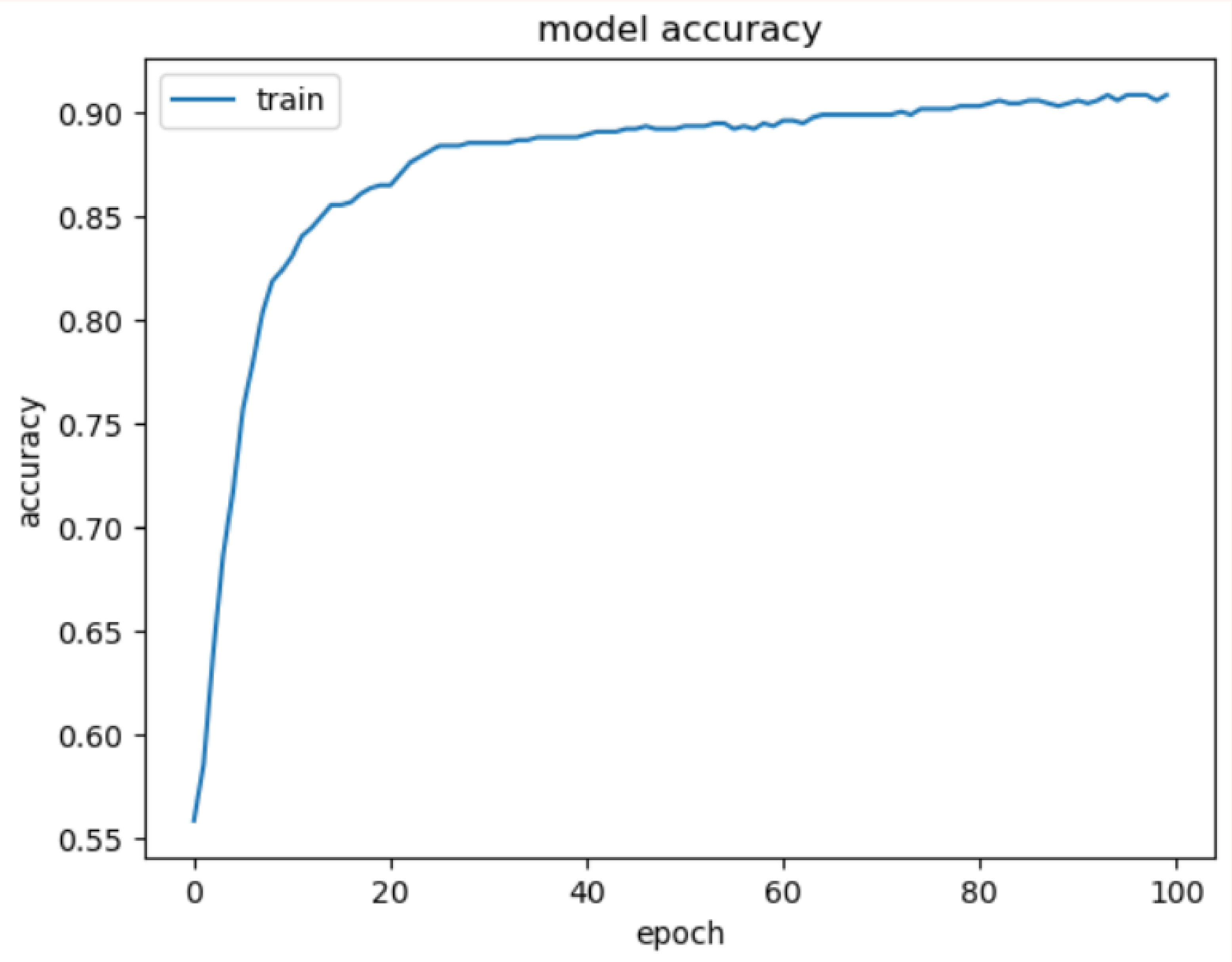
Second layer : 6 units   Activation='Relu'

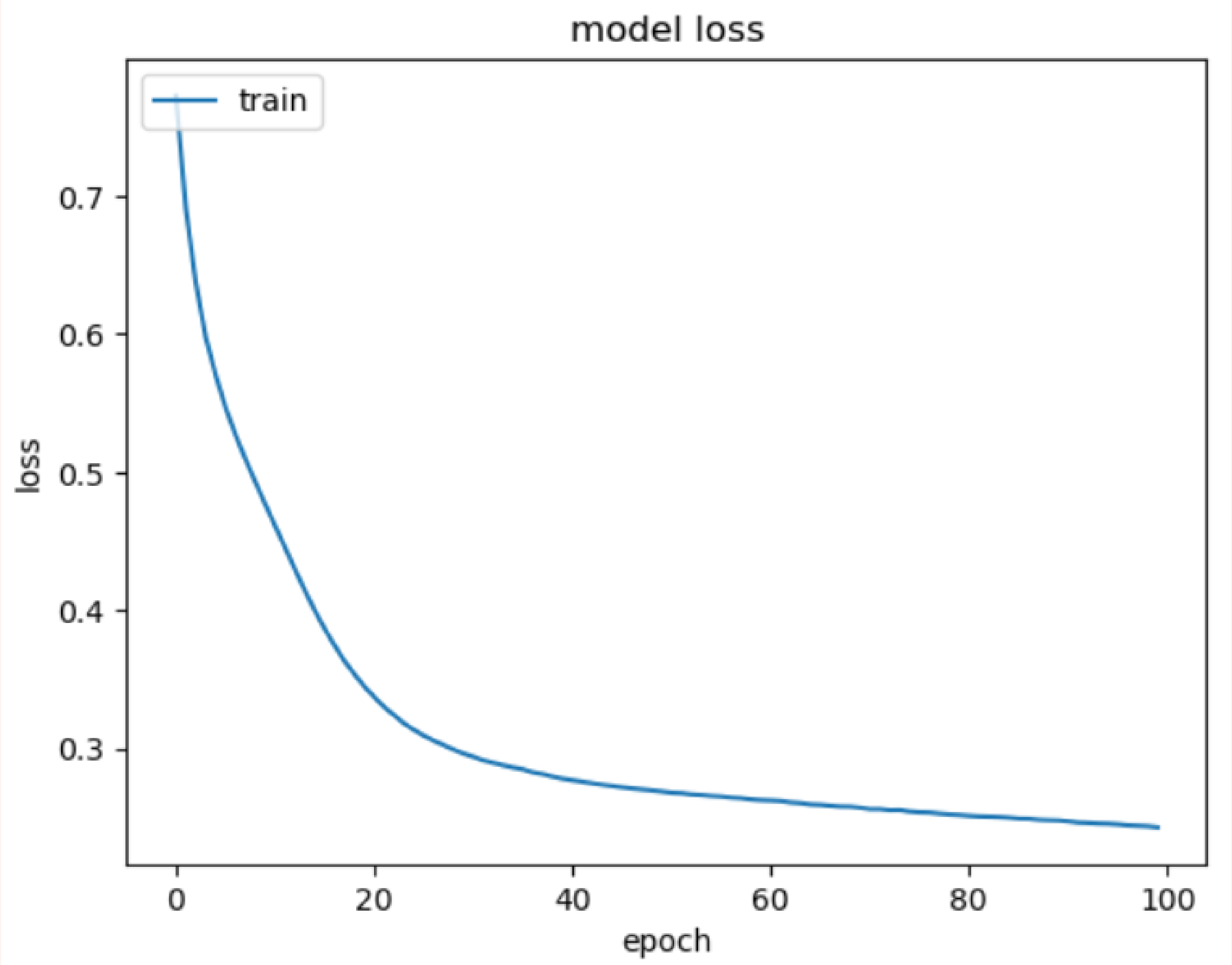
Output layer :one unit

Activation Function 'sigmoid'



Plot the model training history





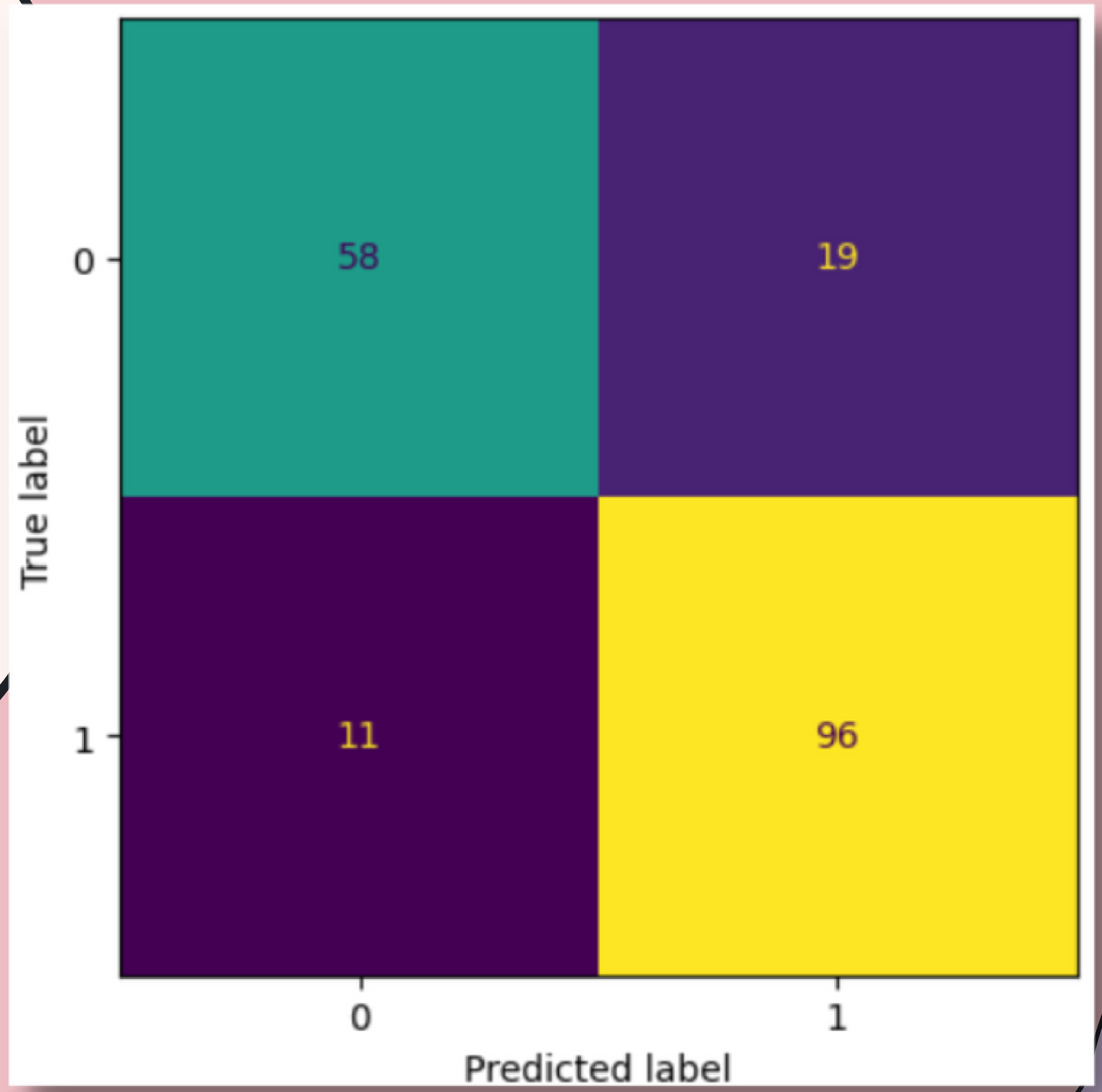
Third NN Model

One layer

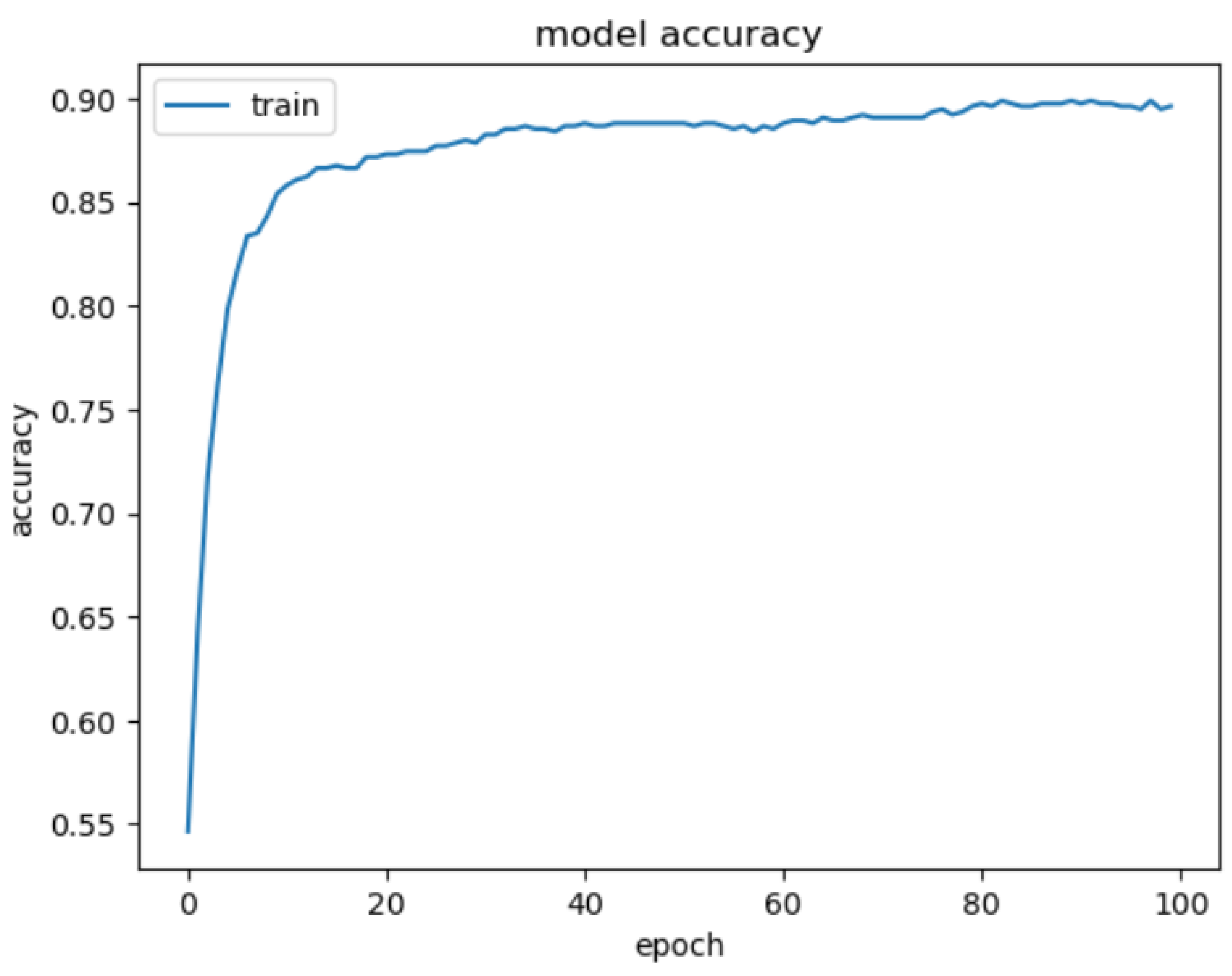
layer: 8 units   Activation='relu'

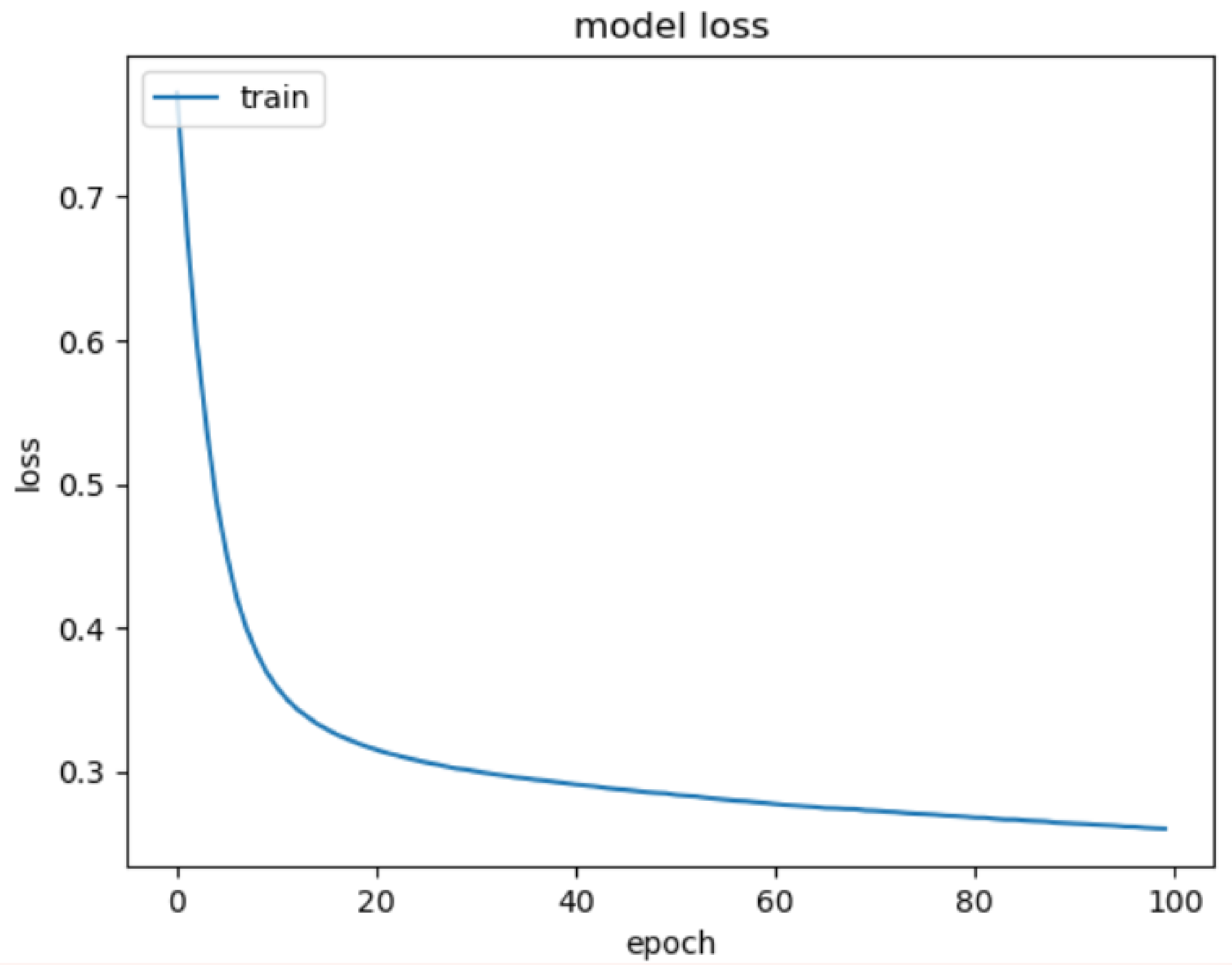
Output layer :one unit

Activation Function 'sigmoid'



Plot the model training history





Fourth NN Model

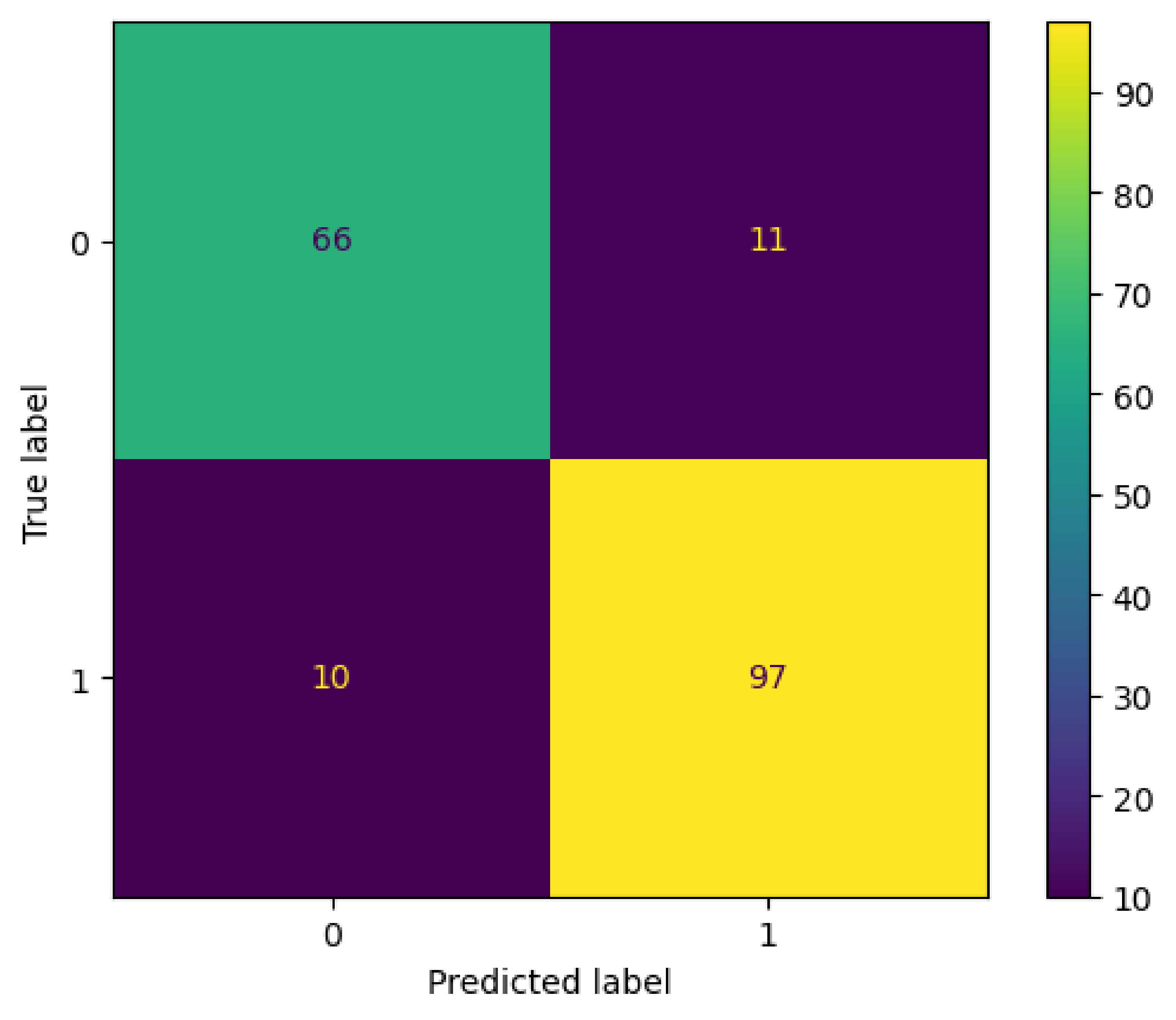
Two layers

First layer: 16 units Activation='swish'

Second layer : 16 units Activation='swish'

Output layer :one unit

Activation Function 'sigmoid'

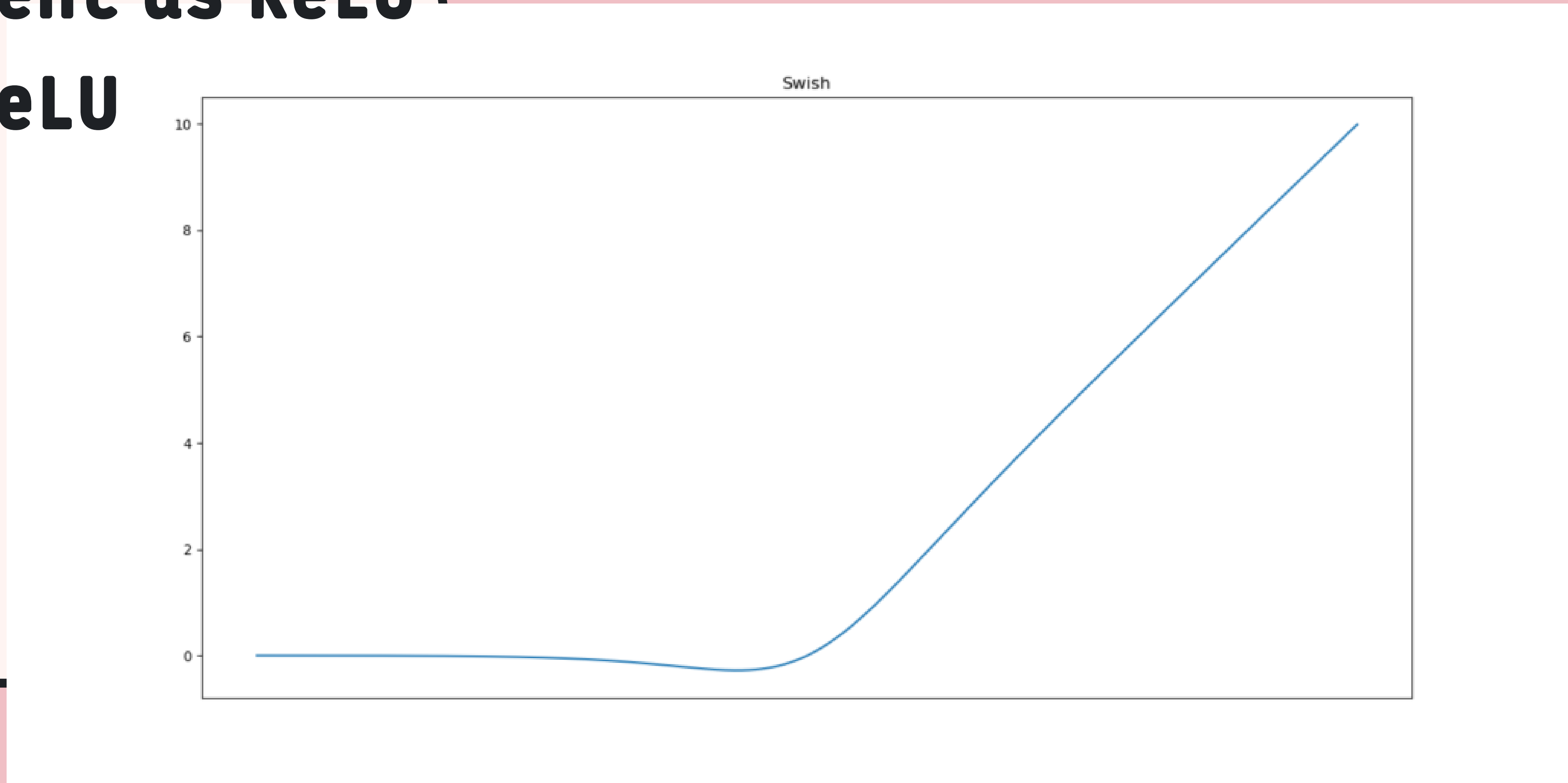


Swish activation function

- Swish is a activation function which was discovered by researchers at Google.

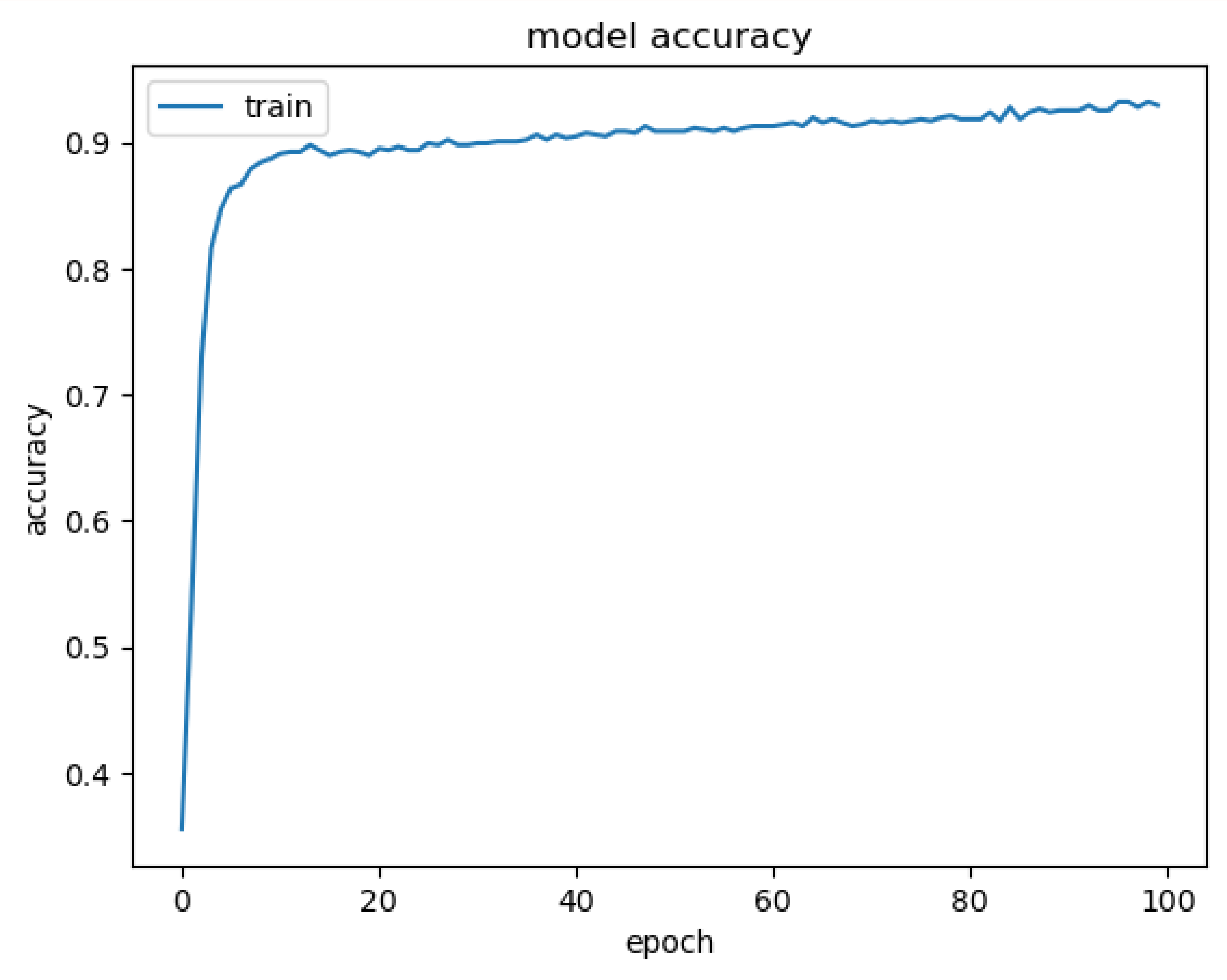
- Swish is as computationally efficient as ReLU and shows better performance than ReLU on deeper models.

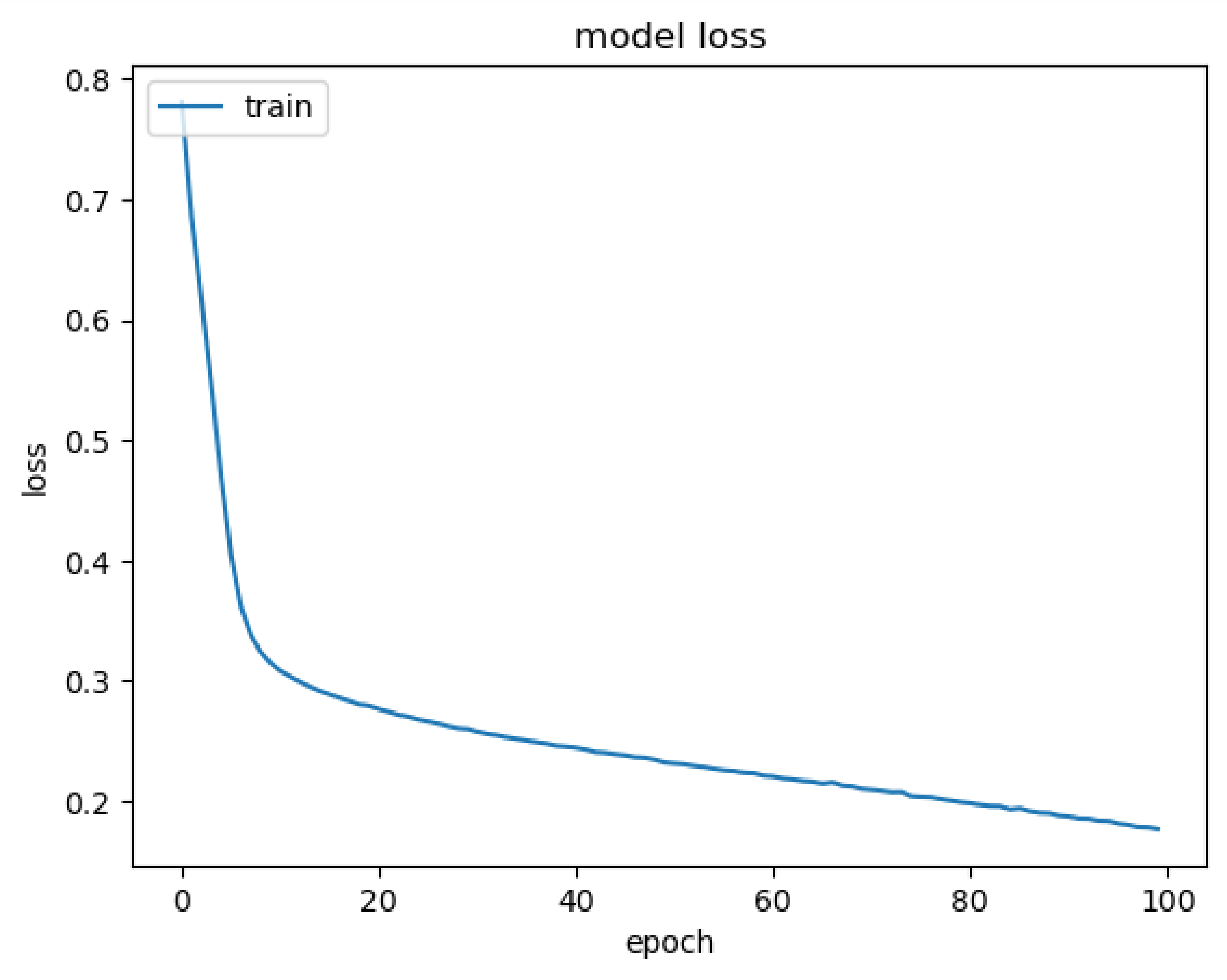
- The values for swish ranges from negative infinity to infinity.



source:www.analyticsvidhya.com

Plot the model training history





comparison of NN model results

model no.		precision	recall	f1-score	support	accuracy
1	0 1	0.83 0.86	0.81 0.88	0.82 0.87	77 107	0.85
2	0 1	0.86 0.89	0.84 0.90	0.85 0.89	77 107	0.88
3	0 1	0.84 0.83	0.75 0.90	0.79 0.86	77 107	0.84
4	0 1	0.87 0.90	0.86 0.91	0.86 0.90	77 107	0.89

Machine learning models

Logistic regression model

```
classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

```
accuracy_score(y_test,y_pred,normalize=True)
```

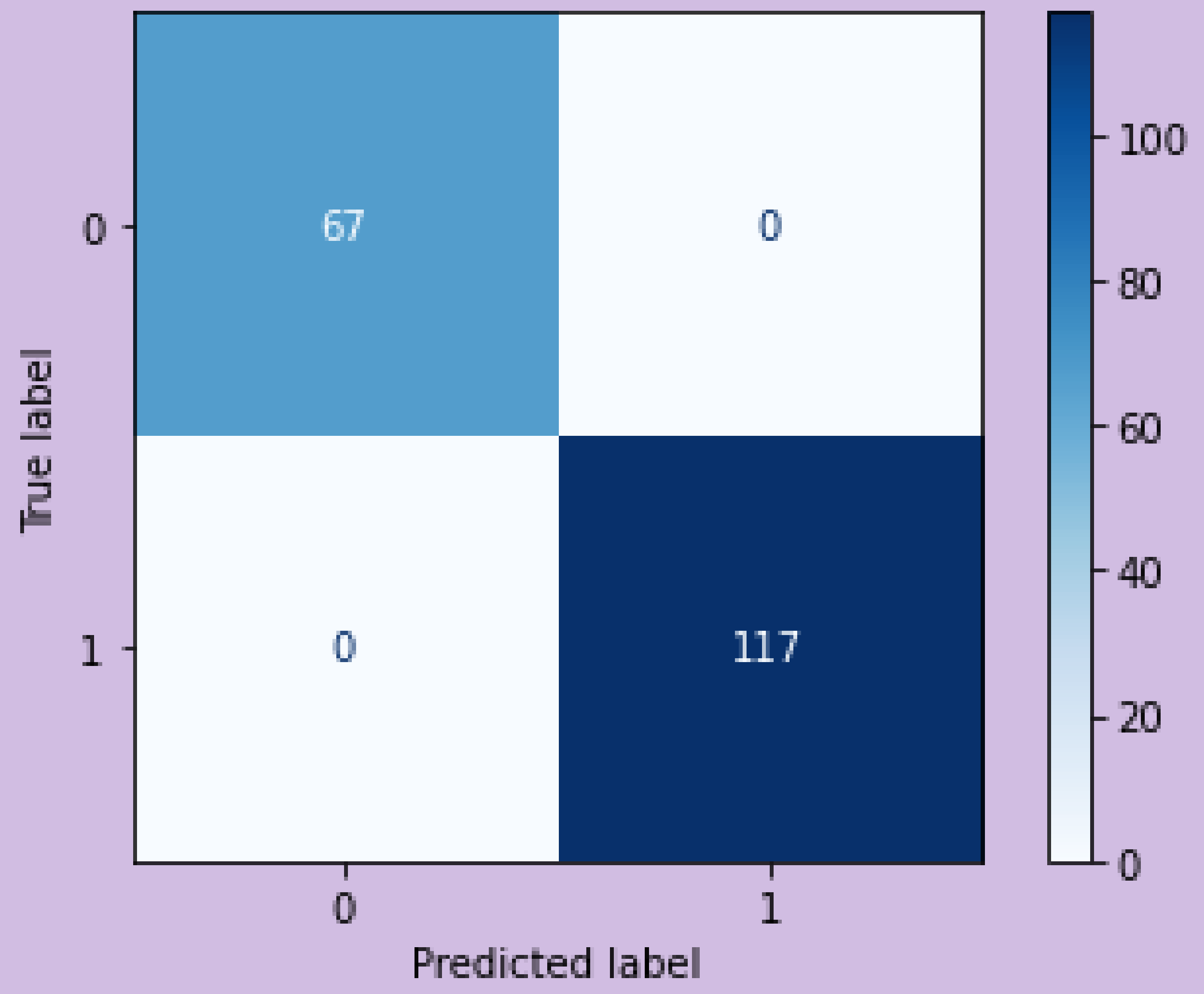
```
0.8260869565217391
```

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[56, 21],
       [11, 96]], dtype=int64)
```

```
pred = classifier.predict(X_test)
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.84	0.73	0.78	77
1	0.82	0.90	0.86	107
accuracy			0.83	184
macro avg	0.83	0.81	0.82	184
weighted avg	0.83	0.83	0.82	184



Logistic regression model-Tuning

```
# Instantiate Standard Scaler
scaler = StandardScaler()
# Fit & transform data.
scaled_df = scaler.fit_transform(X_train)
```

```
pca = PCA()
pca.fit(scaled_df)
```

```
PCA()
```

```
pipe = make_pipeline(
    PCA(n_components= 6),
    LogisticRegression())
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

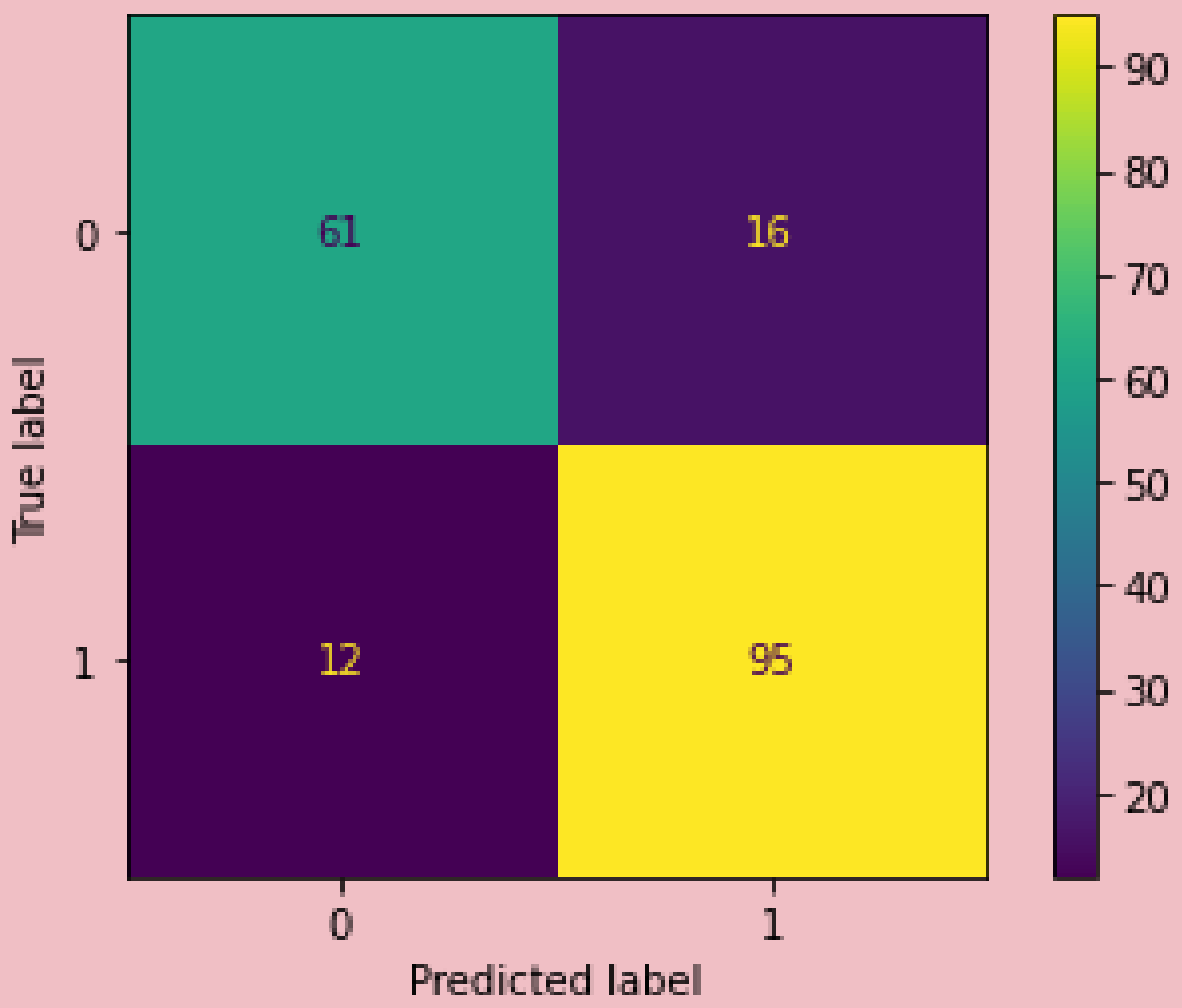
```
0.8478260869565217
```

```
pred = pipe.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.79      0.81        77
           1       0.86      0.89      0.87       107

    accuracy                           0.85       184
   macro avg       0.85      0.84      0.84       184
weighted avg       0.85      0.85      0.85       184
```



Desicion tree model

```
scaler = StandardScaler()
# Fit & transform data.
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

```
class_tree = DecisionTreeClassifier(criterion='gini', max_depth=4)
class_tree.fit(X_train_sc, y_train)
preds_class = class_tree.predict(X_test_sc)
```

```
val_train = round(class_tree.score(X_train_sc, y_train),2)*100
val_test = round(class_tree.score(X_test_sc, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Set Accuracy: {val_test}%')
```

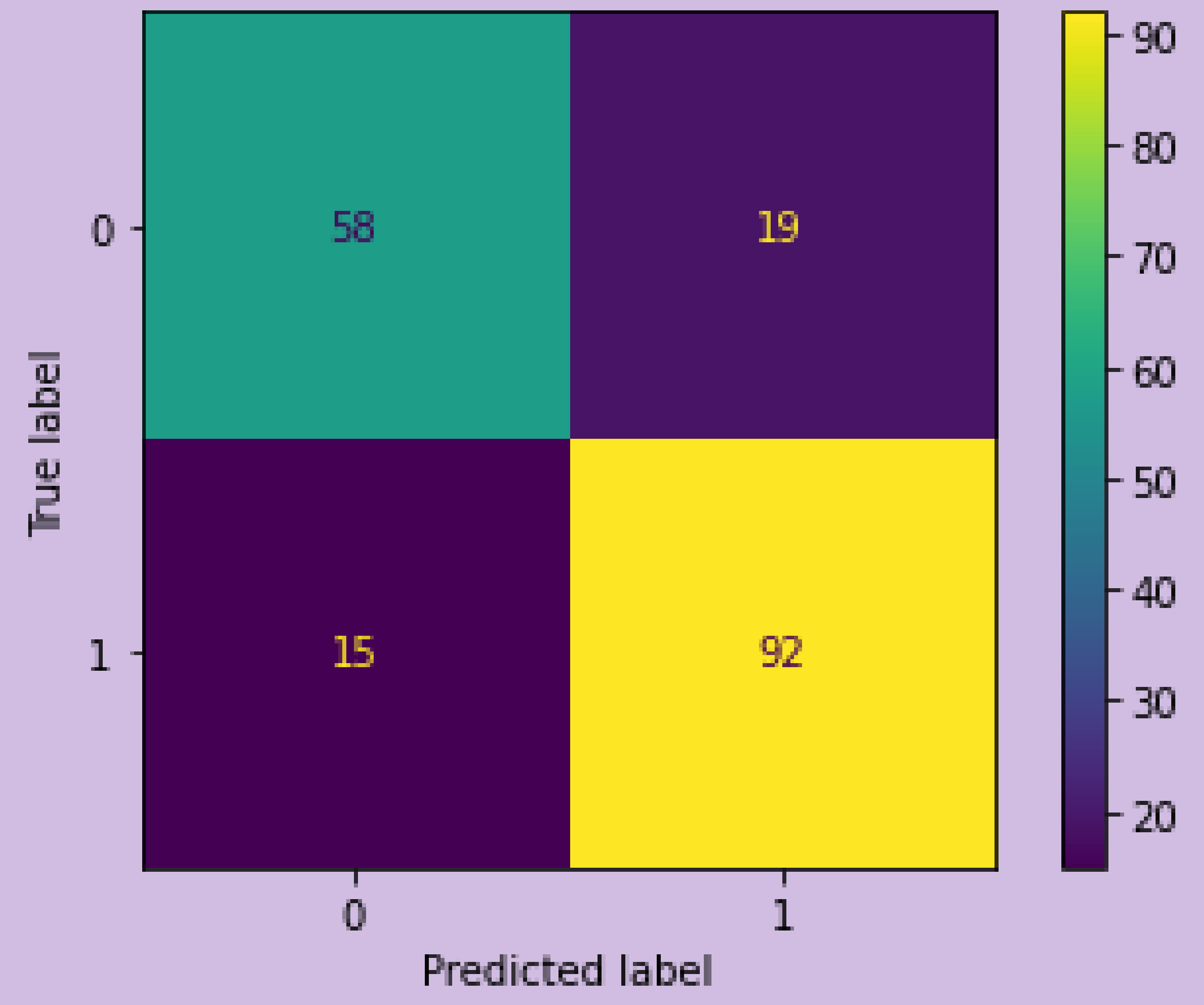
```
Training Accuracy: 89.0%
Test Set Accuracy: 82.0%
```

```
pred = class_tree.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.75      0.77        77
           1       0.83      0.86      0.84       107

    accuracy                           0.82       184
   macro avg       0.81      0.81      0.81       184
weighted avg       0.81      0.82      0.81       184
```



Desicion tree model - Tuning

```
# Classification
param_grid = {
    "criterion": ["gini", "entropy"],
    "max_depth": [2,4,6]
}
grid = GridSearchCV(
    class_tree,
    param_grid,
    cv = 5,
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train, y_train)
```

```
grid.best_score_
```

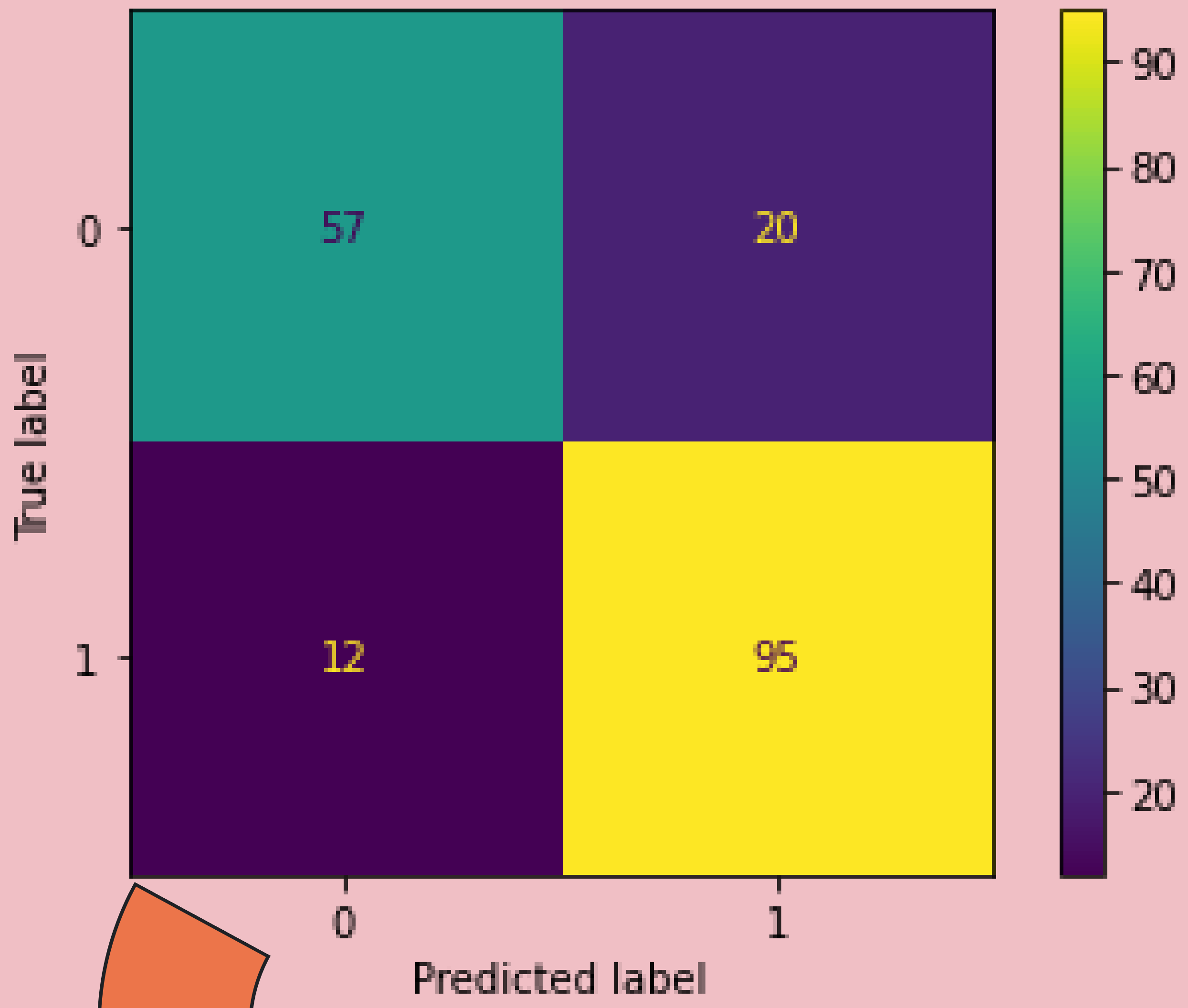
```
0.8487093467523996
```

```
grid.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 6}
```

```
pred = grid.predict(X_test)
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.77	0.79	0.78	77
1	0.85	0.83	0.84	107
accuracy			0.82	184
macro avg	0.81	0.81	0.81	184
weighted avg	0.82	0.82	0.82	184



Random forest model

```
class_forest = RandomForestClassifier(n_estimators = 6, criterion = 'gini', random_state = 0)
class_forest.fit(X_train_sc, y_train)
preds_class = class_forest.predict(X_test_sc)
```

```
val_train = round(class_forest.score(X_train_sc, y_train),2)*100
val_test = round(class_forest.score(X_test_sc, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Set Accuracy: {val_test}%')
```

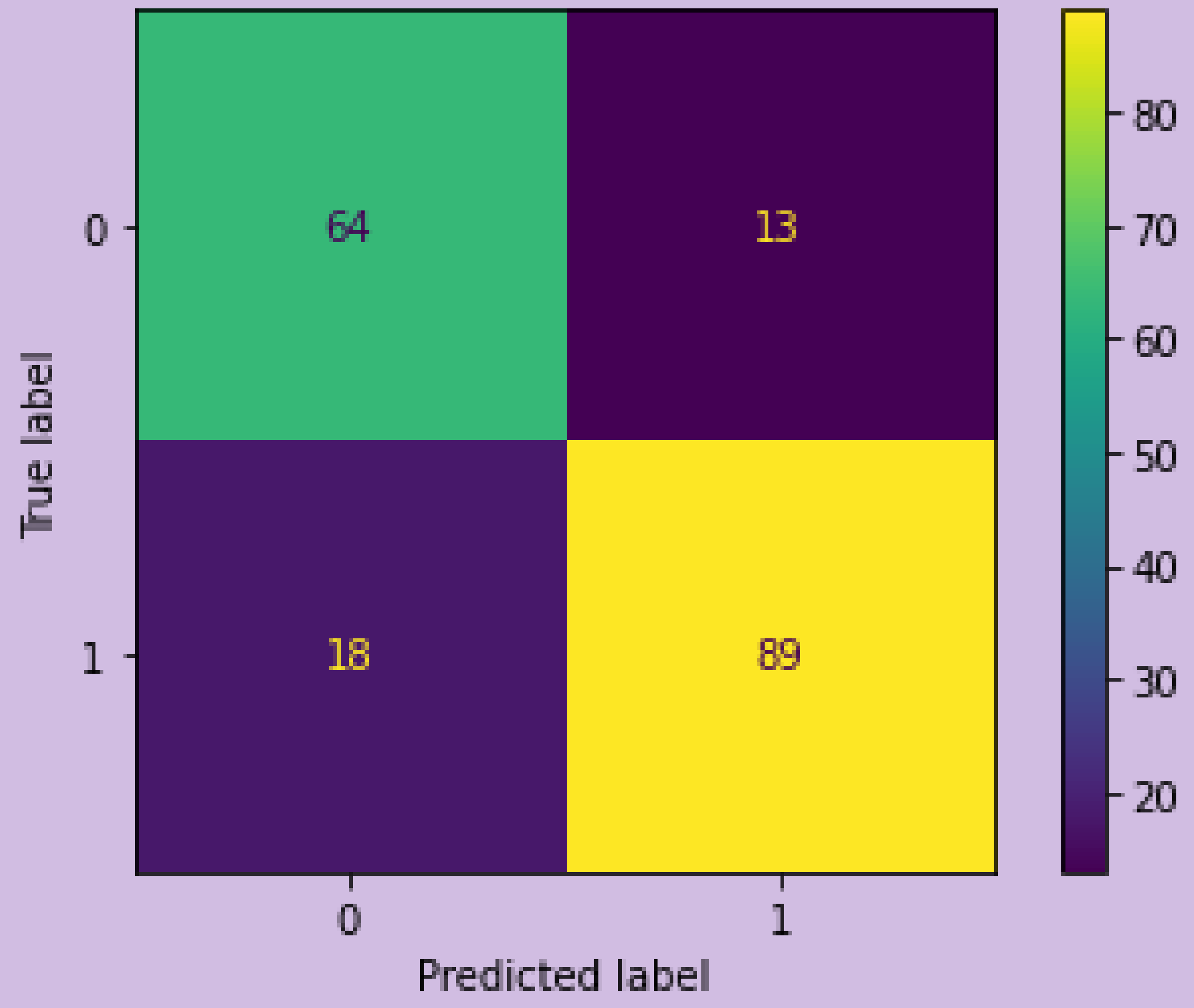
```
Training Accuracy: 99.0%
Test Set Accuracy: 83.0%
```

```
pred = class_forest.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.83      0.81        77
           1       0.87      0.83      0.85       107

    accuracy                           0.83       184
   macro avg       0.83      0.83      0.83       184
weighted avg       0.83      0.83      0.83       184
```



Random forest model-Tuning

```
# Classification
param_grid = {
    "n_estimators": [10,20,30],
    "criterion": ["gini", "entropy"],
    "max_depth": [2,4,6]
}
grid = GridSearchCV(
    class_forest,
    param_grid,
    cv = 5,
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train, y_train)
```

```
grid.best_score_
```

```
0.8814462771409934
```

```
grid.best_params_
```

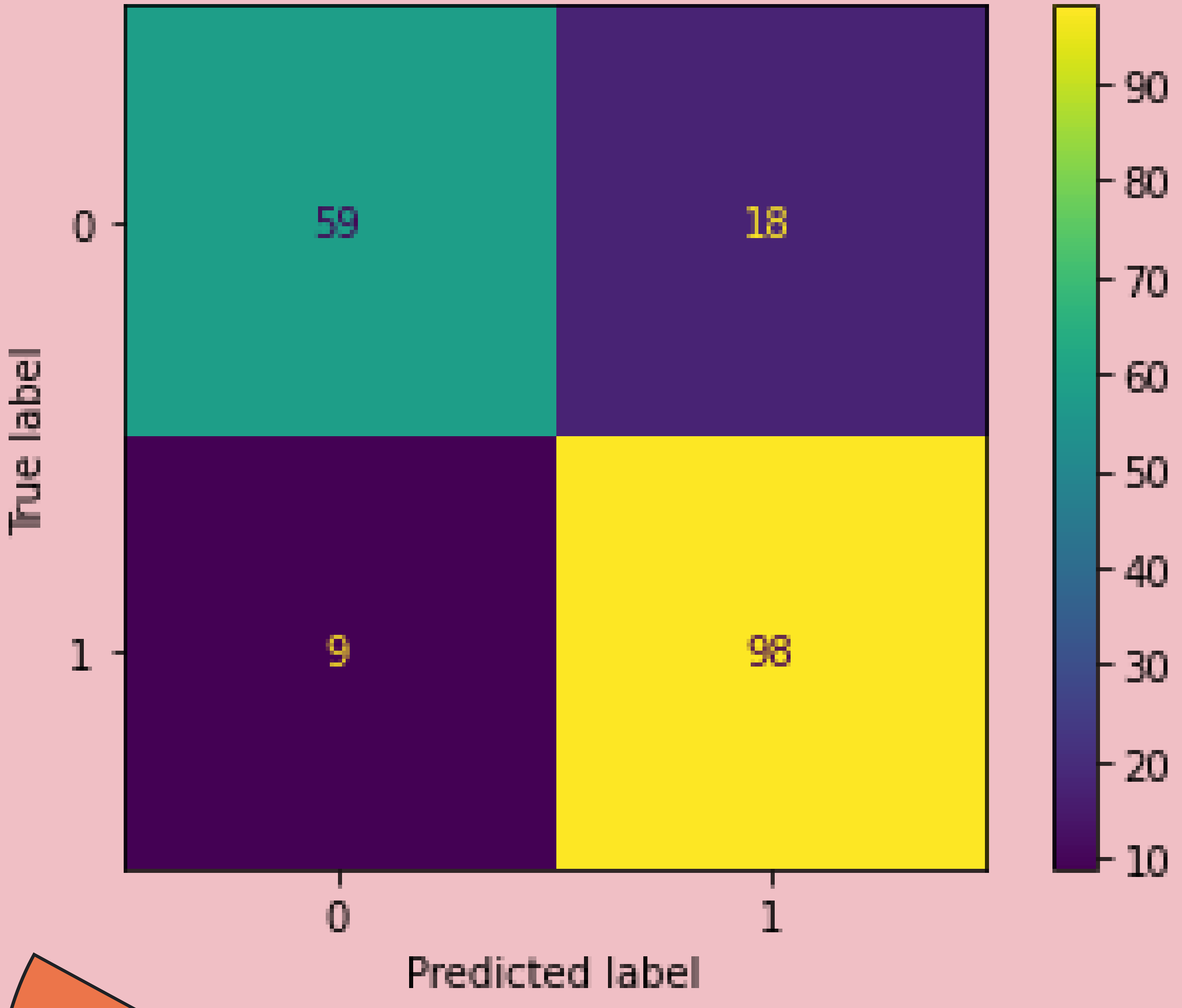
```
{'criterion': 'gini', 'max_depth': 6, 'n_estimators': 20}
```

```
pred = grid.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.77      0.81        77
           1       0.84      0.92      0.88       107

    accuracy                           0.85       184
   macro avg       0.86      0.84      0.85       184
weighted avg       0.85      0.85      0.85       184
```



comparison of model results

model no.		precision	recall	f1-score	support	accuracy
1	0 1	0.84 0.82	0.73 0.90	0.78 0.86	77 107	0.82
2	0 1	0.79 0.83	0.75 0.86	0.77 0.84	77 107	0.82
3	0 1	0.78 0.87	0.83 0.83	0.81 0.85	77 107	0.83

comparison of tuning model results

model no.		precision	recall	f1-score	support	accuracy
1	0 1	0.84 0.86	0.79 0.89	0.81 0.87	77 107	0.85
2	0 1	0.77 0.85	0.79 0.83	0.78 0.84	77 107	0.82
3	0 1	0.87 0.84	0.77 0.92	0.81 0.88	77 107	0.85

THANK YOU