model

May 2, 2025

1 Classification of Breast Cancer using ANN

[ ]:
```python
# Importing Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import seaborn as sns
```

[2]:
```python
# Load Dataset
data = pd.read_csv('data.csv')
data.head()
```

[2]:
```
   id  x.radius_mean  x.texture_mean  x.perimeter_mean  x.area_mean  \
0   1         13.540           14.36             87.46        566.3
1   2         13.080           15.71             85.63        520.0
2   3          9.504           12.44             60.34        273.9
3   4         13.030           18.42             82.61        523.8
4   5          8.196           16.84             51.71        201.9

   x.smoothness_mean  x.compactness_mean  x.concavity_mean  \
0            0.09779             0.08129           0.06664
1            0.10750             0.12700           0.04568
2            0.10240             0.06492           0.02956
3            0.08983             0.03766           0.02562
4            0.08600             0.05943           0.01588

   x.concave_pts_mean  x.symmetry_mean  …  x.texture_worst  \
0            0.047810           0.1885  …            19.26
1            0.031100           0.1967  …            20.49
2            0.020760           0.1815  …            15.66
3            0.029230           0.1467  …            22.81
```

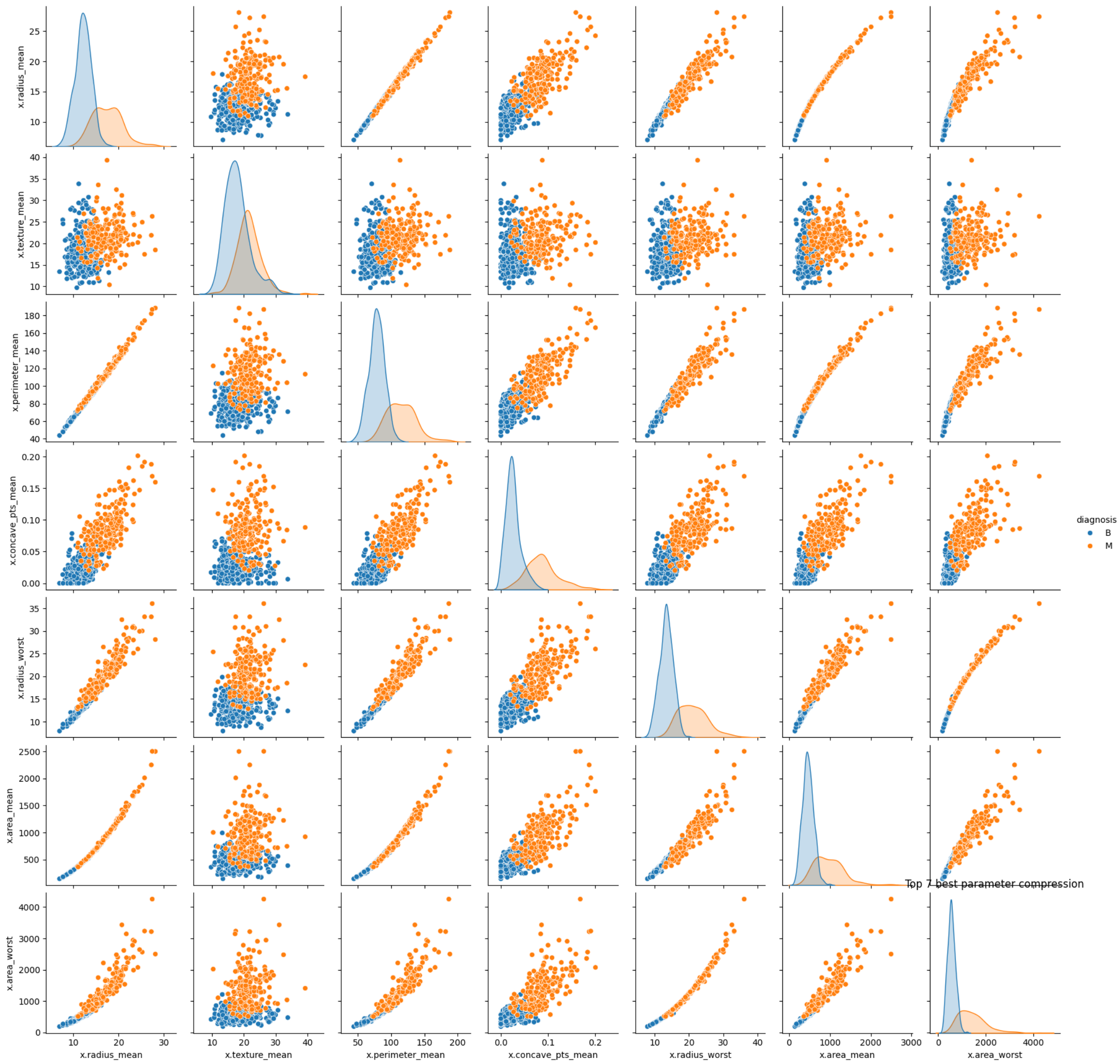
```
4           0.005917          0.1769  …            21.96

   x.perimeter_worst  x.area_worst  x.smoothness_worst  x.compactness_worst  \
0              99.70         711.2             0.14400              0.17730
1              96.09         630.5             0.13120              0.27760
2              65.13         314.9             0.13240              0.11480
3              84.46         545.9             0.09701              0.04619
4              57.26         242.2             0.12970              0.13570

   x.concavity_worst  x.concave_pts_worst  x.symmetry_worst  \
0            0.23900              0.12880            0.2977
1            0.18900              0.07283            0.3184
2            0.08867              0.06227            0.2450
3            0.04833              0.05013            0.1987
4            0.06880              0.02564            0.3105

   x.fractal_dim_worst  diagnosis
0              0.07259          B
1              0.08183          B
2              0.07773          B
3              0.06169          B
4              0.07409          B

[5 rows x 32 columns]
```

[3]:
```python
# Drop Unnecessary Columns if present
if 'Unnamed: 32' in data.columns:
    data.drop('Unnamed: 32', axis=1, inplace=True)
if 'id' in data.columns:
    data.drop('id', axis=1, inplace=True)
```

[4]:
```python
data.describe().T.style.background_gradient(cmap = sns.color_palette("ch:s=-.
 ↪2,r=.6", as_cmap=True))
```

[4]:
```
<pandas.io.formats.style.Styler at 0x1a04b4f9720>
```

[5]:
```python
sns.pairplot(data[['diagnosis', 'x.radius_mean', 'x.texture_mean', 'x.
 ↪perimeter_mean', 'x.concave_pts_mean', 'x.radius_worst', 'x.area_mean', 'x.
 ↪area_worst' ]], hue='diagnosis')
plt.title('Top 7 best parameter compression')
plt.savefig("results/data_variation_plot.png")
plt.show()
```



[6]:
```
# Encode Labels
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
data['diagnosis']
```

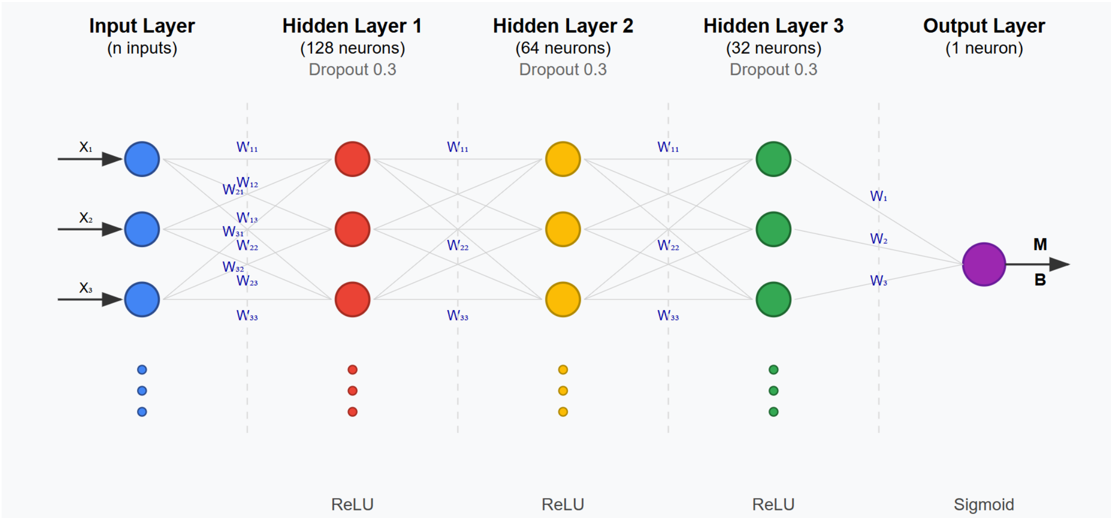
[6]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    1
565    1
566    1
567    1
```

```
568    1
Name: diagnosis, Length: 569, dtype: int64
```

[7]:
```python
# Split features and labels
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

# Standardize the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)
```

[8]:
```python
# Build Deep Learning Model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.
↪shape[1],),name='Input_layer'),
    Dropout(0.3),
    Dense(64, activation='relu', name='First_hidden_layer'),
    Dropout(0.3),
    Dense(32, activation='relu', name='Second_hidden_layer'),
    Dropout(0.3),
    Dense(1, activation='sigmoid', name='Output_layer')
])
```

```
c:\Users\Uditya\Desktop\Deep Learning\Deep Learning for Beginner\venv\lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```



Deep Neural Network Architecture (ANN)

[9]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
Input_layer (Dense)	(None, 128)	3,968
dropout (Dropout)	(None, 128)	0
First_hidden_layer (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
Second_hidden_layer (Dense)	(None, 32)	2,080
dropout_2 (Dropout)	(None, 32)	0
Output_layer (Dense)	(None, 1)	33

Total params: 14,337 (56.00 KB)

Trainable params: 14,337 (56.00 KB)

Non-trainable params: 0 (0.00 B)

[10]:
```python
# Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

# Train Model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
↪validation_split=0.2, verbose=1)

# Evaluate Model
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Epoch 1/100
12/12                 3s 46ms/step -
accuracy: 0.6819 - loss: 0.6648 - val_accuracy: 0.9341 - val_loss: 0.4199
Epoch 2/100
12/12                 0s 11ms/step -
accuracy: 0.8480 - loss: 0.4184 - val_accuracy: 0.9451 - val_loss: 0.2363
Epoch 3/100
12/12                 0s 11ms/step -
accuracy: 0.9410 - loss: 0.2373 - val_accuracy: 0.9560 - val_loss: 0.1425
Epoch 4/100
12/12                 0s 12ms/step -
accuracy: 0.9652 - loss: 0.1586 - val_accuracy: 0.9560 - val_loss: 0.1018
Epoch 5/100
12/12                 0s 13ms/step -
accuracy: 0.9724 - loss: 0.1061 - val_accuracy: 0.9780 - val_loss: 0.0760
Epoch 6/100
12/12                 0s 13ms/step -
accuracy: 0.9688 - loss: 0.1290 - val_accuracy: 0.9780 - val_loss: 0.0634
Epoch 7/100
12/12                 0s 12ms/step -
accuracy: 0.9585 - loss: 0.0968 - val_accuracy: 0.9780 - val_loss: 0.0662
Epoch 8/100
12/12                 1s 32ms/step -
accuracy: 0.9860 - loss: 0.0494 - val_accuracy: 0.9780 - val_loss: 0.0648
Epoch 9/100
12/12                 0s 23ms/step -
accuracy: 0.9787 - loss: 0.0922 - val_accuracy: 0.9780 - val_loss: 0.0609
Epoch 10/100
12/12                 0s 18ms/step -
accuracy: 0.9792 - loss: 0.0584 - val_accuracy: 0.9780 - val_loss: 0.0676
Epoch 11/100
12/12                 0s 13ms/step -
accuracy: 0.9906 - loss: 0.0612 - val_accuracy: 0.9780 - val_loss: 0.0716
Epoch 12/100
```

```
12/12            0s 12ms/step -
accuracy: 0.9898 - loss: 0.0512 - val_accuracy: 0.9780 - val_loss: 0.0702
Epoch 13/100
12/12            0s 11ms/step -
accuracy: 0.9890 - loss: 0.0599 - val_accuracy: 0.9780 - val_loss: 0.0744
Epoch 14/100
12/12            0s 12ms/step -
accuracy: 0.9952 - loss: 0.0309 - val_accuracy: 0.9780 - val_loss: 0.0828
Epoch 15/100
12/12            0s 13ms/step -
accuracy: 0.9964 - loss: 0.0247 - val_accuracy: 0.9780 - val_loss: 0.0863
Epoch 16/100
12/12            0s 12ms/step -
accuracy: 0.9888 - loss: 0.0400 - val_accuracy: 0.9780 - val_loss: 0.0900
Epoch 17/100
12/12            0s 11ms/step -
accuracy: 0.9966 - loss: 0.0260 - val_accuracy: 0.9780 - val_loss: 0.0963
Epoch 18/100
12/12            0s 12ms/step -
accuracy: 0.9873 - loss: 0.0572 - val_accuracy: 0.9670 - val_loss: 0.0981
Epoch 19/100
12/12            0s 12ms/step -
accuracy: 0.9855 - loss: 0.0315 - val_accuracy: 0.9670 - val_loss: 0.0953
Epoch 20/100
12/12            0s 12ms/step -
accuracy: 0.9868 - loss: 0.0460 - val_accuracy: 0.9780 - val_loss: 0.0873
Epoch 21/100
12/12            0s 13ms/step -
accuracy: 0.9858 - loss: 0.0359 - val_accuracy: 0.9780 - val_loss: 0.0963
Epoch 22/100
12/12            0s 12ms/step -
accuracy: 0.9968 - loss: 0.0174 - val_accuracy: 0.9780 - val_loss: 0.1000
Epoch 23/100
12/12            0s 12ms/step -
accuracy: 0.9912 - loss: 0.0218 - val_accuracy: 0.9780 - val_loss: 0.1047
Epoch 24/100
12/12            1s 25ms/step -
accuracy: 0.9982 - loss: 0.0186 - val_accuracy: 0.9780 - val_loss: 0.1062
Epoch 25/100
12/12            0s 15ms/step -
accuracy: 0.9898 - loss: 0.0421 - val_accuracy: 0.9780 - val_loss: 0.1055
Epoch 26/100
12/12            0s 15ms/step -
accuracy: 0.9949 - loss: 0.0169 - val_accuracy: 0.9670 - val_loss: 0.1365
Epoch 27/100
12/12            0s 15ms/step -
accuracy: 0.9942 - loss: 0.0168 - val_accuracy: 0.9670 - val_loss: 0.1424
Epoch 28/100
```

```
12/12            0s 14ms/step -
accuracy: 0.9903 - loss: 0.0529 - val_accuracy: 0.9670 - val_loss: 0.2061
Epoch 29/100
12/12            0s 12ms/step -
accuracy: 0.9936 - loss: 0.0170 - val_accuracy: 0.9670 - val_loss: 0.1972
Epoch 30/100
12/12            0s 13ms/step -
accuracy: 0.9951 - loss: 0.0177 - val_accuracy: 0.9670 - val_loss: 0.1421
Epoch 31/100
12/12            0s 12ms/step -
accuracy: 0.9885 - loss: 0.0253 - val_accuracy: 0.9780 - val_loss: 0.1125
Epoch 32/100
12/12            0s 12ms/step -
accuracy: 0.9799 - loss: 0.0388 - val_accuracy: 0.9670 - val_loss: 0.1626
Epoch 33/100
12/12            0s 12ms/step -
accuracy: 0.9935 - loss: 0.0223 - val_accuracy: 0.9670 - val_loss: 0.1690
Epoch 34/100
12/12            0s 13ms/step -
accuracy: 0.9923 - loss: 0.0275 - val_accuracy: 0.9670 - val_loss: 0.1467
Epoch 35/100
12/12            0s 11ms/step -
accuracy: 0.9926 - loss: 0.0185 - val_accuracy: 0.9670 - val_loss: 0.1461
Epoch 36/100
12/12            0s 13ms/step -
accuracy: 0.9964 - loss: 0.0167 - val_accuracy: 0.9670 - val_loss: 0.1451
Epoch 37/100
12/12            0s 12ms/step -
accuracy: 0.9991 - loss: 0.0071 - val_accuracy: 0.9670 - val_loss: 0.1463
Epoch 38/100
12/12            0s 13ms/step -
accuracy: 0.9959 - loss: 0.0138 - val_accuracy: 0.9670 - val_loss: 0.1533
Epoch 39/100
12/12            0s 11ms/step -
accuracy: 0.9953 - loss: 0.0113 - val_accuracy: 0.9670 - val_loss: 0.1540
Epoch 40/100
12/12            0s 13ms/step -
accuracy: 0.9973 - loss: 0.0118 - val_accuracy: 0.9670 - val_loss: 0.1480
Epoch 41/100
12/12            0s 26ms/step -
accuracy: 0.9949 - loss: 0.0183 - val_accuracy: 0.9670 - val_loss: 0.1561
Epoch 42/100
12/12            0s 12ms/step -
accuracy: 1.0000 - loss: 0.0066 - val_accuracy: 0.9670 - val_loss: 0.1660
Epoch 43/100
12/12            0s 12ms/step -
accuracy: 0.9967 - loss: 0.0095 - val_accuracy: 0.9670 - val_loss: 0.1767
Epoch 44/100
```

```
12/12              0s 12ms/step -
accuracy: 0.9982 - loss: 0.0109 - val_accuracy: 0.9670 - val_loss: 0.1673
Epoch 45/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.9670 - val_loss: 0.1685
Epoch 46/100
12/12              0s 25ms/step -
accuracy: 0.9936 - loss: 0.0201 - val_accuracy: 0.9670 - val_loss: 0.2187
Epoch 47/100
12/12              0s 12ms/step -
accuracy: 0.9923 - loss: 0.0133 - val_accuracy: 0.9670 - val_loss: 0.2615
Epoch 48/100
12/12              0s 12ms/step -
accuracy: 0.9972 - loss: 0.0113 - val_accuracy: 0.9670 - val_loss: 0.2702
Epoch 49/100
12/12              0s 21ms/step -
accuracy: 0.9886 - loss: 0.0220 - val_accuracy: 0.9780 - val_loss: 0.1297
Epoch 50/100
12/12              0s 12ms/step -
accuracy: 0.9938 - loss: 0.0122 - val_accuracy: 0.9780 - val_loss: 0.1242
Epoch 51/100
12/12              0s 12ms/step -
accuracy: 0.9873 - loss: 0.0223 - val_accuracy: 0.9670 - val_loss: 0.1472
Epoch 52/100
12/12              0s 12ms/step -
accuracy: 0.9989 - loss: 0.0089 - val_accuracy: 0.9670 - val_loss: 0.1528
Epoch 53/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0104 - val_accuracy: 0.9670 - val_loss: 0.1667
Epoch 54/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0052 - val_accuracy: 0.9670 - val_loss: 0.1900
Epoch 55/100
12/12              0s 11ms/step -
accuracy: 0.9967 - loss: 0.0054 - val_accuracy: 0.9670 - val_loss: 0.2206
Epoch 56/100
12/12              0s 12ms/step -
accuracy: 0.9994 - loss: 0.0037 - val_accuracy: 0.9670 - val_loss: 0.2385
Epoch 57/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0075 - val_accuracy: 0.9670 - val_loss: 0.2112
Epoch 58/100
12/12              0s 12ms/step -
accuracy: 0.9947 - loss: 0.0071 - val_accuracy: 0.9670 - val_loss: 0.2317
Epoch 59/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.9670 - val_loss: 0.2430
Epoch 60/100
```

```
12/12            0s 12ms/step -
accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.9670 - val_loss: 0.2505
Epoch 61/100
12/12            0s 12ms/step -
accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.9670 - val_loss: 0.2518
Epoch 62/100
12/12            0s 12ms/step -
accuracy: 1.0000 - loss: 0.0017 - val_accuracy: 0.9670 - val_loss: 0.2569
Epoch 63/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.9670 - val_loss: 0.2669
Epoch 64/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.9670 - val_loss: 0.2729
Epoch 65/100
12/12            0s 12ms/step -
accuracy: 0.9986 - loss: 0.0033 - val_accuracy: 0.9670 - val_loss: 0.2683
Epoch 66/100
12/12            0s 11ms/step -
accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 0.9670 - val_loss: 0.2597
Epoch 67/100
12/12            0s 13ms/step -
accuracy: 0.9991 - loss: 0.0045 - val_accuracy: 0.9670 - val_loss: 0.3156
Epoch 68/100
12/12            0s 11ms/step -
accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 0.9670 - val_loss: 0.3948
Epoch 69/100
12/12            1s 35ms/step -
accuracy: 0.9921 - loss: 0.0086 - val_accuracy: 0.9670 - val_loss: 0.3585
Epoch 70/100
12/12            1s 31ms/step -
accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.9670 - val_loss: 0.3285
Epoch 71/100
12/12            0s 17ms/step -
accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9670 - val_loss: 0.3388
Epoch 72/100
12/12            0s 13ms/step -
accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.9670 - val_loss: 0.3546
Epoch 73/100
12/12            0s 12ms/step -
accuracy: 0.9978 - loss: 0.0063 - val_accuracy: 0.9670 - val_loss: 0.3741
Epoch 74/100
12/12            0s 13ms/step -
accuracy: 0.9978 - loss: 0.0037 - val_accuracy: 0.9670 - val_loss: 0.3688
Epoch 75/100
12/12            0s 11ms/step -
accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.9670 - val_loss: 0.3472
Epoch 76/100
```

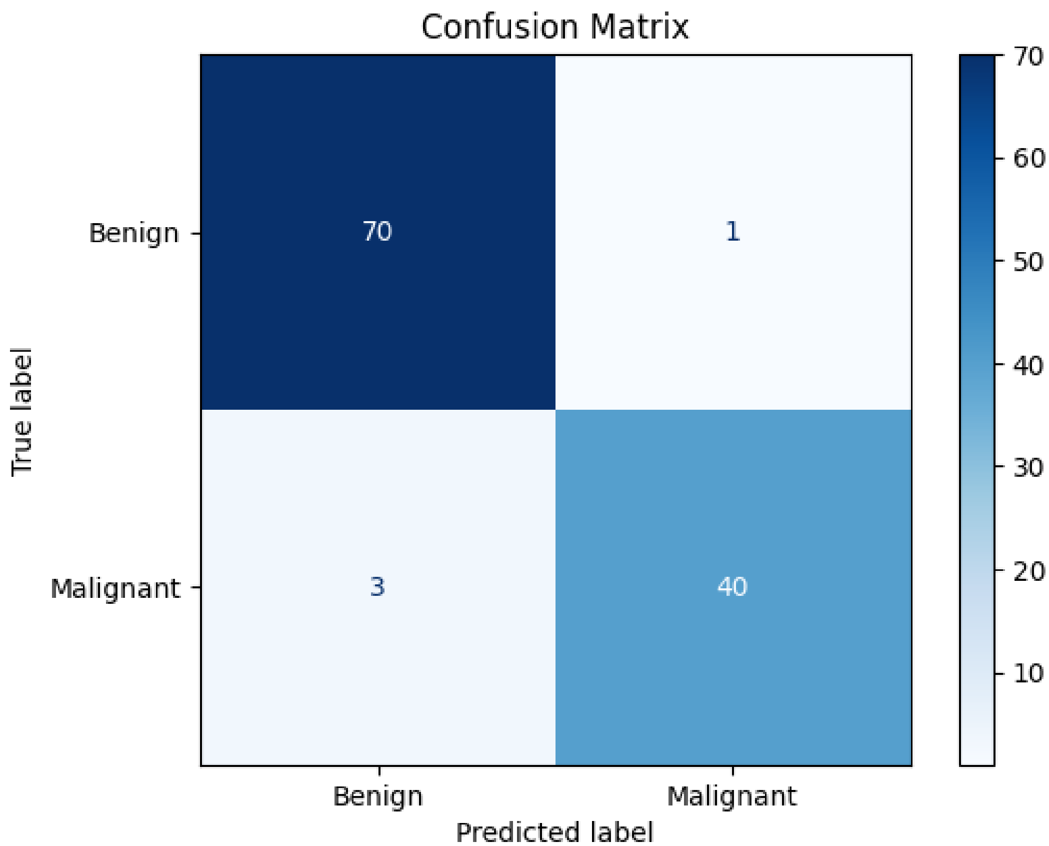
```
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0024 - val_accuracy: 0.9670 - val_loss: 0.2840
Epoch 77/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 7.7155e-04 - val_accuracy: 0.9670 - val_loss: 0.2706
Epoch 78/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.9670 - val_loss: 0.2790
Epoch 79/100
12/12              0s 12ms/step -
accuracy: 0.9986 - loss: 0.0037 - val_accuracy: 0.9670 - val_loss: 0.2889
Epoch 80/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.9670 - val_loss: 0.3002
Epoch 81/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 0.9670 - val_loss: 0.3157
Epoch 82/100
12/12              0s 11ms/step -
accuracy: 1.0000 - loss: 7.5450e-04 - val_accuracy: 0.9670 - val_loss: 0.3226
Epoch 83/100
12/12              0s 11ms/step -
accuracy: 0.9923 - loss: 0.0218 - val_accuracy: 0.9670 - val_loss: 0.4364
Epoch 84/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0026 - val_accuracy: 0.9670 - val_loss: 0.4322
Epoch 85/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 0.9670 - val_loss: 0.4069
Epoch 86/100
12/12              0s 13ms/step -
accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 0.9670 - val_loss: 0.3798
Epoch 87/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0010 - val_accuracy: 0.9670 - val_loss: 0.3549
Epoch 88/100
12/12              0s 15ms/step -
accuracy: 1.0000 - loss: 9.2520e-04 - val_accuracy: 0.9670 - val_loss: 0.3379
Epoch 89/100
12/12              0s 17ms/step -
accuracy: 1.0000 - loss: 9.7941e-04 - val_accuracy: 0.9670 - val_loss: 0.3255
Epoch 90/100
12/12              0s 12ms/step -
accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.9670 - val_loss: 0.3086
Epoch 91/100
12/12              0s 30ms/step -
accuracy: 1.0000 - loss: 6.6085e-04 - val_accuracy: 0.9670 - val_loss: 0.3082
Epoch 92/100
```

```
12/12            0s 17ms/step -
accuracy: 0.9973 - loss: 0.0035 - val_accuracy: 0.9670 - val_loss: 0.3130
Epoch 93/100
12/12            1s 37ms/step -
accuracy: 1.0000 - loss: 0.0024 - val_accuracy: 0.9670 - val_loss: 0.3076
Epoch 94/100
12/12            0s 14ms/step -
accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.9670 - val_loss: 0.3206
Epoch 95/100
12/12            0s 19ms/step -
accuracy: 0.9923 - loss: 0.0091 - val_accuracy: 0.9560 - val_loss: 0.3580
Epoch 96/100
12/12            0s 32ms/step -
accuracy: 1.0000 - loss: 4.5755e-04 - val_accuracy: 0.9560 - val_loss: 0.3905
Epoch 97/100
12/12            0s 12ms/step -
accuracy: 0.9986 - loss: 0.0025 - val_accuracy: 0.9560 - val_loss: 0.3798
Epoch 98/100
12/12            0s 22ms/step -
accuracy: 0.9989 - loss: 0.0031 - val_accuracy: 0.9670 - val_loss: 0.3746
Epoch 99/100
12/12            0s 16ms/step -
accuracy: 0.9978 - loss: 0.0113 - val_accuracy: 0.9670 - val_loss: 0.2186
Epoch 100/100
12/12            0s 15ms/step -
accuracy: 0.9967 - loss: 0.0062 - val_accuracy: 0.9670 - val_loss: 0.2027
4/4            0s 28ms/step

Confusion Matrix:
 [[70  1]
 [ 3 40]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        71
           1       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

[11]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Predict on test data
y_pred_probs = model.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype("int32")
```

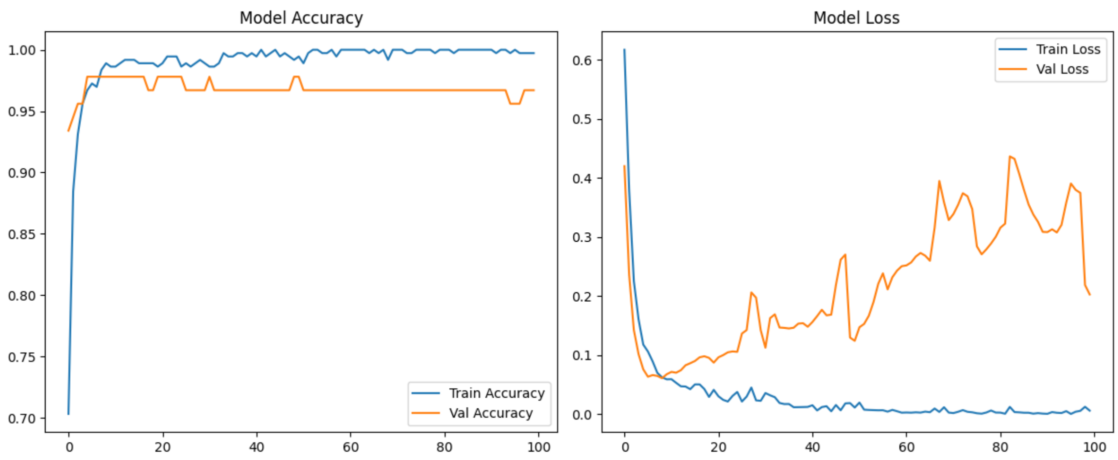
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Benign",␣
 ↪"Malignant"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.savefig("results/confusion_metrics.png")
plt.show()
```

```
4/4              0s 13ms/step
```



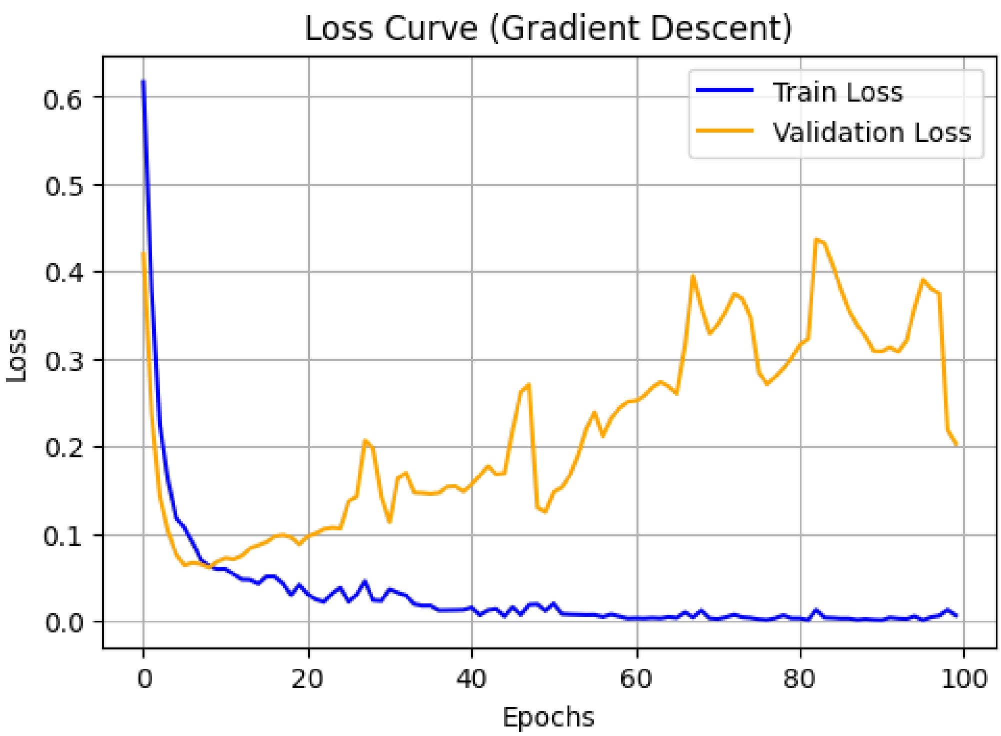
[12]:
```
import os

# Create results folder if not exists
os.makedirs("results", exist_ok=True)

# Plot Accuracy and Loss
plt.figure(figsize=(12, 5))
```

```python
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.legend()

plt.tight_layout()

# Save the plot
plt.savefig("results/training_metrics.png")
plt.show()
```



[13]:
```python
# Plotting only Loss (Gradient Descent Visualization)
import os
os.makedirs("results", exist_ok=True)

plt.figure(figsize=(6, 4))
plt.plot(history.history['loss'], label='Train Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Loss Curve (Gradient Descent)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
```

```
plt.savefig("results/loss_gradient_descent.png")
plt.show()
```



[14]:

```
# Prepare Dataset
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(16)

# Optimizer & Loss
loss_fn = tf.keras.losses.BinaryCrossentropy()
optimizer = tf.keras.optimizers.Adam()

# For Tracking
gradient_norms = {layer.name: [] for layer in model.layers if len(layer.
 ↪trainable_variables) > 0}
losses = []

# Training Loop
epochs = 100
for epoch in range(epochs):
    print(f"\nEpoch {epoch+1}/{epochs}")
    epoch_losses = []

    for step, (x_batch, y_batch) in enumerate(train_dataset):
```

```python
        with tf.GradientTape() as tape:
            logits = model(x_batch, training=True)
            loss_value = loss_fn(y_batch, logits)

        grads = tape.gradient(loss_value, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        epoch_losses.append(loss_value.numpy())

        # Save gradient norms
        idx = 0
        for layer in model.layers:
            if len(layer.trainable_variables) > 0:
                for var in layer.trainable_variables:
                    grad = grads[idx]
                    norm = tf.norm(grad).numpy() if grad is not None else 0
                    gradient_norms[layer.name].append(norm)
                    idx += 1

    # Average loss of epoch
    epoch_loss = np.mean(epoch_losses)
    losses.append(epoch_loss)
    print(f"Loss: {epoch_loss:.4f}")

# Plotting
plt.figure(figsize=(14, 5), dpi=200)

# Loss vs Epoch
plt.subplot(1, 2, 1)
plt.plot(range(1, epochs+1), losses, marker='o', color='blue')
plt.title("Loss vs Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.grid(True)

# Gradient Norm vs Epoch (per layer)
plt.subplot(1, 2, 2)
steps_per_epoch = len(train_dataset)

for layer_name, norms in gradient_norms.items():
    norms = np.array(norms)
    avg_per_epoch = [np.mean(norms[i*steps_per_epoch:(i+1)*steps_per_epoch]) for i in range(epochs)]
    plt.plot(range(1, epochs+1), avg_per_epoch, label=layer_name)

plt.title("Gradient Norm vs Epochs (per layer)")
plt.xlabel("Epochs")
plt.ylabel("Avg Gradient Norm")
```

```python
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.savefig("results/gradient_norm_per_epoch.png")
plt.show()
```

```
Epoch 1/100
Loss: 0.1018

Epoch 2/100
Loss: 0.0315

Epoch 3/100
Loss: 0.0341

Epoch 4/100
Loss: 0.0560

Epoch 5/100
Loss: 0.0232

Epoch 6/100
Loss: 0.0213

Epoch 7/100
Loss: 0.0190

Epoch 8/100
Loss: 0.0188

Epoch 9/100
Loss: 0.0310

Epoch 10/100
Loss: 0.0112

Epoch 11/100
Loss: 0.0127

Epoch 12/100
Loss: 0.0122

Epoch 13/100
Loss: 0.0045

Epoch 14/100
```

```
Loss: 0.0070

Epoch 15/100
Loss: 0.0042

Epoch 16/100
Loss: 0.0114

Epoch 17/100
Loss: 0.0031

Epoch 18/100
Loss: 0.0209

Epoch 19/100
Loss: 0.0022

Epoch 20/100
Loss: 0.0046

Epoch 21/100
Loss: 0.0093

Epoch 22/100
Loss: 0.0066

Epoch 23/100
Loss: 0.0111

Epoch 24/100
Loss: 0.0096

Epoch 25/100
Loss: 0.0034

Epoch 26/100
Loss: 0.0047

Epoch 27/100
Loss: 0.0050

Epoch 28/100
Loss: 0.0063

Epoch 29/100
Loss: 0.0059

Epoch 30/100
```

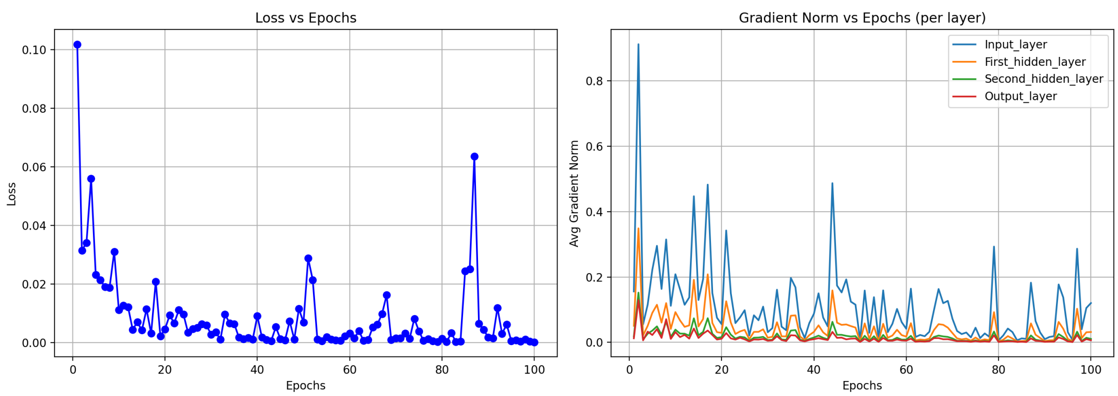
```
Loss: 0.0027

Epoch 31/100
Loss: 0.0036

Epoch 32/100
Loss: 0.0010

Epoch 33/100
Loss: 0.0097

Epoch 34/100
Loss: 0.0066

Epoch 35/100
Loss: 0.0064

Epoch 36/100
Loss: 0.0018

Epoch 37/100
Loss: 0.0013

Epoch 38/100
Loss: 0.0016

Epoch 39/100
Loss: 0.0011

Epoch 40/100
Loss: 0.0091

Epoch 41/100
Loss: 0.0017

Epoch 42/100
Loss: 0.0009

Epoch 43/100
Loss: 0.0006

Epoch 44/100
Loss: 0.0053

Epoch 45/100
Loss: 0.0012

Epoch 46/100
```

```
Loss: 0.0008

Epoch 47/100
Loss: 0.0073

Epoch 48/100
Loss: 0.0011

Epoch 49/100
Loss: 0.0115

Epoch 50/100
Loss: 0.0069

Epoch 51/100
Loss: 0.0288

Epoch 52/100
Loss: 0.0214

Epoch 53/100
Loss: 0.0011

Epoch 54/100
Loss: 0.0005

Epoch 55/100
Loss: 0.0019

Epoch 56/100
Loss: 0.0011

Epoch 57/100
Loss: 0.0009

Epoch 58/100
Loss: 0.0007

Epoch 59/100
Loss: 0.0021

Epoch 60/100
Loss: 0.0032

Epoch 61/100
Loss: 0.0015

Epoch 62/100
```

```
Loss: 0.0039

Epoch 63/100
Loss: 0.0007

Epoch 64/100
Loss: 0.0009

Epoch 65/100
Loss: 0.0052

Epoch 66/100
Loss: 0.0062

Epoch 67/100
Loss: 0.0097

Epoch 68/100
Loss: 0.0162

Epoch 69/100
Loss: 0.0010

Epoch 70/100
Loss: 0.0015

Epoch 71/100
Loss: 0.0016

Epoch 72/100
Loss: 0.0032

Epoch 73/100
Loss: 0.0014

Epoch 74/100
Loss: 0.0082

Epoch 75/100
Loss: 0.0039

Epoch 76/100
Loss: 0.0006

Epoch 77/100
Loss: 0.0012

Epoch 78/100
```

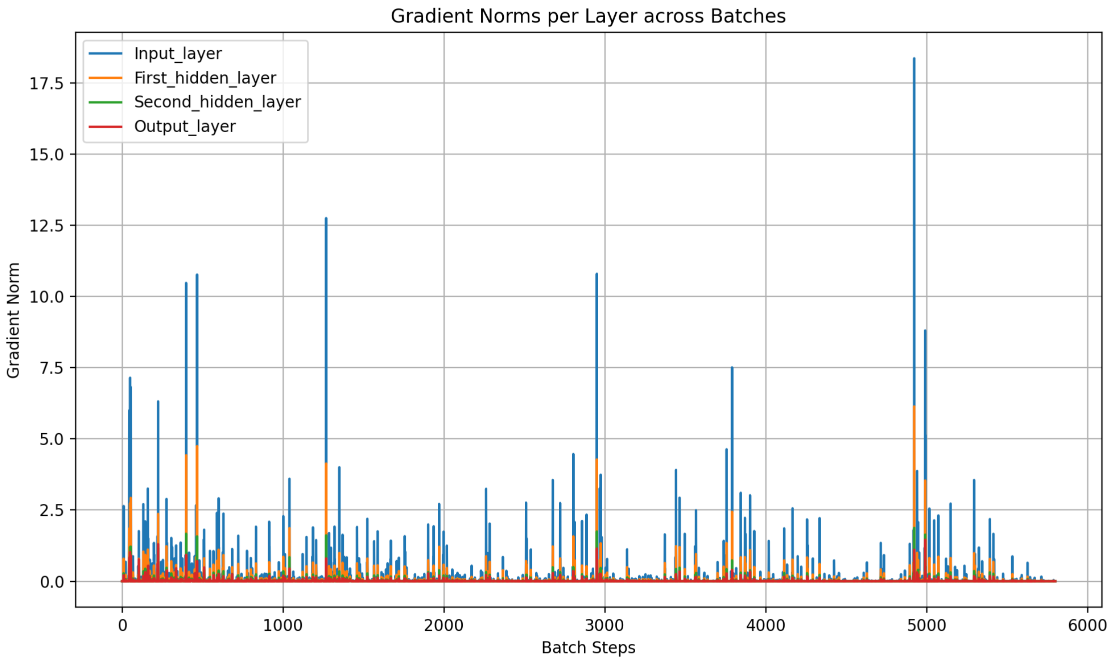
```
Loss: 0.0005

Epoch 79/100
Loss: 0.0003

Epoch 80/100
Loss: 0.0014

Epoch 81/100
Loss: 0.0003

Epoch 82/100
Loss: 0.0032

Epoch 83/100
Loss: 0.0003

Epoch 84/100
Loss: 0.0004

Epoch 85/100
Loss: 0.0243

Epoch 86/100
Loss: 0.0250

Epoch 87/100
Loss: 0.0635

Epoch 88/100
Loss: 0.0065

Epoch 89/100
Loss: 0.0044

Epoch 90/100
Loss: 0.0018

Epoch 91/100
Loss: 0.0014

Epoch 92/100
Loss: 0.0119

Epoch 93/100
Loss: 0.0031

Epoch 94/100
```

```
Loss: 0.0062

Epoch 95/100
Loss: 0.0006

Epoch 96/100
Loss: 0.0008

Epoch 97/100
Loss: 0.0004

Epoch 98/100
Loss: 0.0011

Epoch 99/100
Loss: 0.0004

Epoch 100/100
Loss: 0.0002
```



[15]:
```python
# Create result directory if not exists
import os
os.makedirs("results", exist_ok=True)

# Plot gradient norms for each layer
plt.figure(figsize=(10, 6), dpi= 200)
for layer_name, norms in gradient_norms.items():
    plt.plot(norms, label=layer_name)

plt.title("Gradient Norms per Layer across Batches")
plt.xlabel("Batch Steps")
plt.ylabel("Gradient Norm")
plt.legend()
```

```
plt.grid(True)
plt.tight_layout()
plt.savefig("results/gradient_flow_per_layer.png")
plt.show()
```



1.1 Binary Cross Entropy Loss

core of **gradient descent update:**

$$W_{t+1} = W_t - \eta \cdot \frac{\partial L}{\partial W}$$

core of **gradient descent update:**

$$W_{t+1} = W_t - \eta \cdot \frac{\partial L}{\partial W}$$

core of **gradient descent update:**

$$W_{t+1} = W_t - \eta \cdot \frac{\partial L}{\partial W}$$

[16]:

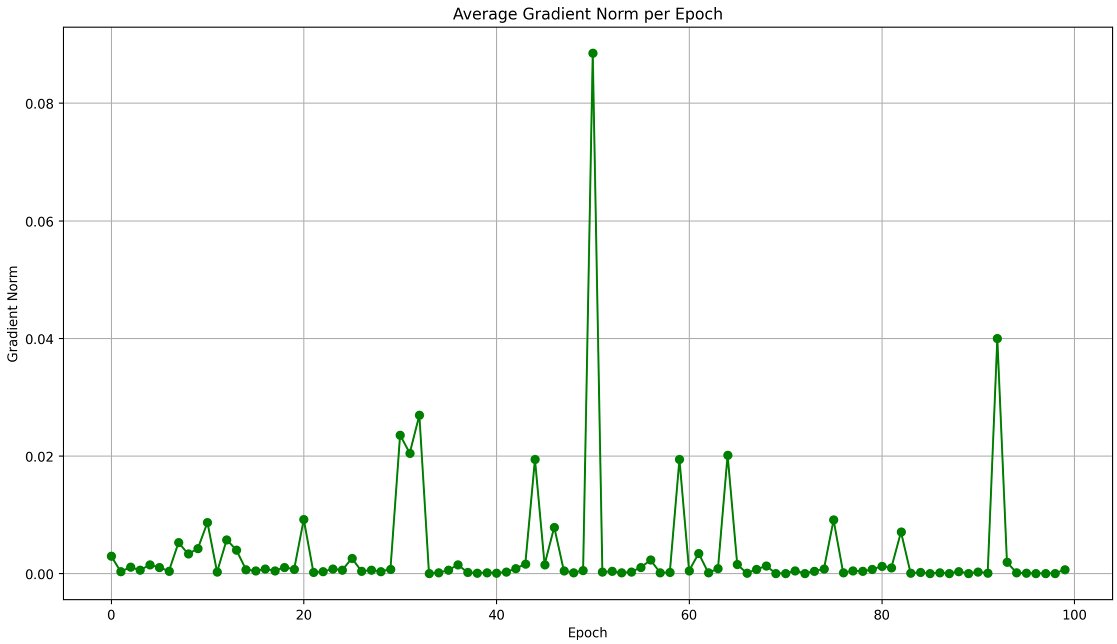
```
loss_fn = tf.keras.losses.BinaryCrossentropy()
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# To store gradient norms
gradient_norms = []
loss_values = []

# Convert data to tensors
X_tensor = tf.convert_to_tensor(X_train, dtype=tf.float32)
y_tensor = tf.convert_to_tensor(y_train.values.reshape(-1, 1), dtype=tf.float32)

# Training Loop for 100 epochs
for epoch in range(100):
    with tf.GradientTape() as tape:
        predictions = model(X_tensor, training=True)
        loss_value = loss_fn(y_tensor, predictions)

    # Compute gradients
    grads = tape.gradient(loss_value, model.trainable_weights)

    # Record gradient norms
    loss_values.append(loss_value.numpy())
    grad_norm = np.mean([tf.norm(g).numpy() for g in grads if g is not None])
    gradient_norms.append(grad_norm)

    # Apply gradients
    optimizer.apply_gradients(zip(grads, model.trainable_weights))

    print(f"Epoch {epoch+1}: Loss = {loss_value:.4f}, Avg. Grad Norm =␣
 ↪{grad_norm:.4f}")

# Plot gradient norms
plt.figure(figsize=(12, 7), dpi=250)
plt.plot(gradient_norms, marker='o', color='green')
plt.title("Average Gradient Norm per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Gradient Norm")
plt.grid(True)
```

```python
plt.tight_layout()
plt.savefig("results/gradient_flow.png")
plt.show()
```

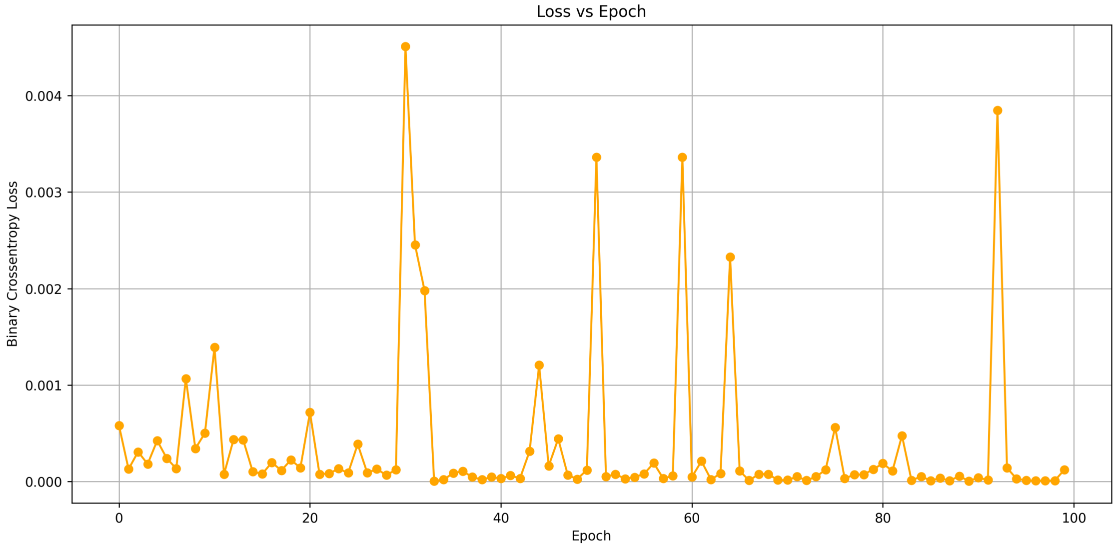
```
Epoch 1: Loss = 0.0006, Avg. Grad Norm = 0.0030
Epoch 2: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 3: Loss = 0.0003, Avg. Grad Norm = 0.0011
Epoch 4: Loss = 0.0002, Avg. Grad Norm = 0.0006
Epoch 5: Loss = 0.0004, Avg. Grad Norm = 0.0015
Epoch 6: Loss = 0.0002, Avg. Grad Norm = 0.0011
Epoch 7: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 8: Loss = 0.0011, Avg. Grad Norm = 0.0053
Epoch 9: Loss = 0.0003, Avg. Grad Norm = 0.0033
Epoch 10: Loss = 0.0005, Avg. Grad Norm = 0.0042
Epoch 11: Loss = 0.0014, Avg. Grad Norm = 0.0087
Epoch 12: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 13: Loss = 0.0004, Avg. Grad Norm = 0.0058
Epoch 14: Loss = 0.0004, Avg. Grad Norm = 0.0040
Epoch 15: Loss = 0.0001, Avg. Grad Norm = 0.0007
Epoch 16: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 17: Loss = 0.0002, Avg. Grad Norm = 0.0008
Epoch 18: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 19: Loss = 0.0002, Avg. Grad Norm = 0.0011
Epoch 20: Loss = 0.0001, Avg. Grad Norm = 0.0007
Epoch 21: Loss = 0.0007, Avg. Grad Norm = 0.0092
Epoch 22: Loss = 0.0001, Avg. Grad Norm = 0.0002
Epoch 23: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 24: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 25: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 26: Loss = 0.0004, Avg. Grad Norm = 0.0026
Epoch 27: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 28: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 29: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 30: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 31: Loss = 0.0045, Avg. Grad Norm = 0.0236
Epoch 32: Loss = 0.0025, Avg. Grad Norm = 0.0205
Epoch 33: Loss = 0.0020, Avg. Grad Norm = 0.0269
Epoch 34: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 35: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 36: Loss = 0.0001, Avg. Grad Norm = 0.0006
Epoch 37: Loss = 0.0001, Avg. Grad Norm = 0.0015
Epoch 38: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 39: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 40: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 41: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 42: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 43: Loss = 0.0000, Avg. Grad Norm = 0.0009
Epoch 44: Loss = 0.0003, Avg. Grad Norm = 0.0016
```

```
Epoch 45: Loss = 0.0012, Avg. Grad Norm = 0.0194
Epoch 46: Loss = 0.0002, Avg. Grad Norm = 0.0015
Epoch 47: Loss = 0.0004, Avg. Grad Norm = 0.0079
Epoch 48: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 49: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 50: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 51: Loss = 0.0034, Avg. Grad Norm = 0.0885
Epoch 52: Loss = 0.0001, Avg. Grad Norm = 0.0003
Epoch 53: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 54: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 55: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 56: Loss = 0.0001, Avg. Grad Norm = 0.0011
Epoch 57: Loss = 0.0002, Avg. Grad Norm = 0.0023
Epoch 58: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 59: Loss = 0.0001, Avg. Grad Norm = 0.0002
Epoch 60: Loss = 0.0034, Avg. Grad Norm = 0.0194
Epoch 61: Loss = 0.0000, Avg. Grad Norm = 0.0005
Epoch 62: Loss = 0.0002, Avg. Grad Norm = 0.0034
Epoch 63: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 64: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 65: Loss = 0.0023, Avg. Grad Norm = 0.0202
Epoch 66: Loss = 0.0001, Avg. Grad Norm = 0.0015
Epoch 67: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 68: Loss = 0.0001, Avg. Grad Norm = 0.0007
Epoch 69: Loss = 0.0001, Avg. Grad Norm = 0.0013
Epoch 70: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 71: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 72: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 73: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 74: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 75: Loss = 0.0001, Avg. Grad Norm = 0.0008
Epoch 76: Loss = 0.0006, Avg. Grad Norm = 0.0091
Epoch 77: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 78: Loss = 0.0001, Avg. Grad Norm = 0.0005
Epoch 79: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 80: Loss = 0.0001, Avg. Grad Norm = 0.0007
Epoch 81: Loss = 0.0002, Avg. Grad Norm = 0.0012
Epoch 82: Loss = 0.0001, Avg. Grad Norm = 0.0010
Epoch 83: Loss = 0.0005, Avg. Grad Norm = 0.0071
Epoch 84: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 85: Loss = 0.0001, Avg. Grad Norm = 0.0002
Epoch 86: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 87: Loss = 0.0000, Avg. Grad Norm = 0.0002
Epoch 88: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 89: Loss = 0.0001, Avg. Grad Norm = 0.0004
Epoch 90: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 91: Loss = 0.0000, Avg. Grad Norm = 0.0003
Epoch 92: Loss = 0.0000, Avg. Grad Norm = 0.0001
```

```
Epoch 93: Loss = 0.0038, Avg. Grad Norm = 0.0400
Epoch 94: Loss = 0.0001, Avg. Grad Norm = 0.0019
Epoch 95: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 96: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 97: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 98: Loss = 0.0000, Avg. Grad Norm = 0.0001
Epoch 99: Loss = 0.0000, Avg. Grad Norm = 0.0000
Epoch 100: Loss = 0.0001, Avg. Grad Norm = 0.0007
```



[17]:
```
plt.figure(figsize=(12, 6), dpi=250)
plt.plot(loss_values, marker='o', color='orange')
plt.title("Loss vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Binary Crossentropy Loss")
plt.grid(True)
plt.tight_layout()
plt.savefig("results/loss_vs_epoch.png")
plt.show()
```



[18]:
```
model.save('model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

[20]:
```
import pickle
with open('scaler.pkl', 'wb') as file:
        pickle.dump(scaler, file)
```

2 Thankyou