Projektarbeit

Brustkrebs - Binäre Klassifizierung eines Datensatzes

Quelle: Kaggle

Volkan Korunan

11. Oktober 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Brustkrebs ist weltweit die häufigste Krebserkrankung bei Frauen. Er macht 25 % aller Krebsfälle aus und betraf allein im Jahr 2022 über 2,3 Millionen Menschen. Das entspricht etwa 1 von 8 aller Krebserkrankungen weltweit. Er beginnt, wenn Zellen in der Brust unkontrolliert zu wachsen beginnen. Diese Zellen bilden in der Regel Tumore, die auf Röntgenbildern zu sehen oder als Knoten im Brustbereich zu fühlen sind. Die größte Herausforderung bei der Erkennung von Brustkrebs ist die Klassifizierung der Tumore in bösartig (krebsartig) und gutartig (nicht krebsartig).

1.1 Ziele dieser Arbeit

Das Hauptziel dieser Arbeit ist die Entwicklung eines effizienten und zuverlässigen neuronalen Netzes zur Erkennung von Brustkrebs auf Basis medizinischer Daten. Brustkrebs stellt eine der häufigsten Krebserkrankungen bei Frauen weltweit dar und frühzeitige Diagnosen sind entscheidend für erfolgreiche Behandlungen und höhere Überlebensraten. Traditionelle Diagnosemethoden, wie Mammographien und Biopsien, können zeitaufwendig sein und sind oft von der subjektiven Interpretation des medizinischen Personals abhängig. Mit dem Aufkommen großer medizinischer Datensätze und Fortschritten im Bereich des maschinellen Lernens bieten neuronale Netze die Möglichkeit, komplexe Muster in Daten zu erkennen, die für das menschliche Auge schwer zu identifizieren sind. Insbesondere tiefe neuronale Netze haben sich in der Bild- und Mustererkennung als äußerst effektiv erwiesen und können nichtlineare Beziehungen zwischen Variablen modellieren.Da ich keine Grafikkarte habe, arbeite ich mit einem Datensatz, welcher nicht aus Bildern besteht, sondern mir in Tabellenform vorliegt und bestimmte medizinische Kenndaten beinhaltet, welche das neuronale Netz für das Training verwendet.

Die Entwicklung eines solchen Modells zielt darauf ab, die Genauigkeit und Zuverlässigkeit der Brustkrebsdiagnose zu verbessern. Durch den Einsatz von maschinellem Lernen (Neuronalen Netztes) können wir:

- **Objektivität erhöhen**: Automatisierte Modelle reduzieren subjektive Fehler und liefern konsistente Ergebnisse.
- **Früherkennung fördern**: Durch genaue Vorhersagen können potenzielle Krebsfälle früher identifiziert und behandelt werden.
- **Ressourcen optimieren**: Effiziente Diagnosewerkzeuge entlasten medizinisches Personal und ermöglichen eine fokussierte Patientenbetreuung.

In dieser Arbeit wird ein neuronales Netz konzipiert, trainiert und validiert, um zwischen gutartigen und bösartigen Tumoren zu unterscheiden. Dazu werden moderne Techniken wie Datenvorverarbeitung, Regularisierung und Kreuzvalidierung eingesetzt, um ein robustes Modell zu entwickeln. Das übergeordnete Ziel ist es, meine Kenntnisse in neuronalen Netzen einzusetzen, um ein Modell zu entwickeln, das einen Beitrag zur Brustkrebsdiagnose leistet und dabei eine hohe Genauigkeit gewährleistet.

1.2 Begründung des Themas

1.2.1 Warum das Thema wichtig ist und behandelt werden sollte

- **Hohe Prävalenz von Brustkrebs**: Brustkrebs ist eine der häufigsten Krebserkrankungen bei Frauen weltweit, was die Notwendigkeit effektiver Diagnosemethoden unterstreicht.
- **Lebensrettende Früherkennung**: Eine frühzeitige Diagnose kann die Überlebensrate deutlich erhöhen und ermöglicht weniger invasive Behandlungen.
- **Verbesserungsbedarf bei Diagnosemethoden**: Traditionelle Diagnoseverfahren können zeitaufwendig sein und sind anfällig für menschliche Fehler und subjektive Interpretationen.
- **Wirtschaftliche Auswirkungen**: Effektivere Diagnosemethoden können Gesundheitskosten reduzieren und Ressourcen im Gesundheitssystem effizienter nutzen.
- **Psychosoziale Vorteile**: Schnellere und genauere Diagnosen reduzieren die psychische Belastung für Patientinnen durch Unsicherheit und Wartezeiten.

1.2.2 Warum neuronale Netze dafür angewendet werden

- **Erkennung komplexer Muster**: Neuronale Netze können komplexe, nichtlineare Beziehungen in medizinischen Daten erkennen, die für herkömmliche statistische Methoden schwer zugänglich sind.
- **Hohe Genauigkeit**: Sie haben das Potenzial, die Genauigkeit von Diagnosen zu erhöhen, indem sie subtilere Unterschiede zwischen gutartigen und bösartigen Tumoren identifizieren.
- **Automatisierung und Effizienz**: Neuronale Netze ermöglichen die automatisierte Analyse großer Datensätze, was den Diagnoseprozess beschleunigt.
- **Anpassungsfähigkeit**: Sie können mit neuen Daten kontinuierlich trainiert werden, um ihre Leistungsfähigkeit und Genauigkeit im Laufe der Zeit zu verbessern.
- **Reduzierung menschlicher Fehler**: Durch den Einsatz von KI können subjektive Fehler minimiert und eine konsistente Diagnosestellung gewährleistet werden.

1.3 Darstellung eines persönlichen Erkenntnisinteresses

Persönlich finde dich diese Untersuchung spannend, da sie ein Thema behandelt, das viele Frauen direkt betrifft und jeder Fortschritt hier einen unmittelbaren Einfluss auf die Gesundheit vieler Menschen haben kann. Besonders für Menschen aus Staaten welche nicht die Möglichkeit haben teure Untersuchungen zu bezahlen. Ich sehe allgemein großes Potenzial in der Anwendung von neuronalen Netzen für die Diagnose von Krankheiten, insbesondere durch die Fortschritte beim Training von Convolutional-Layern. Ich freue mich darauf, mit dem untersuchten Datensatz ein solches Problem selbst angehen zu können.

2 Nachvollziehbare Schritte und Erklärung der Vorgehensweise im Code

Als ersten Schritt werden die notwendigen Python Bibliotheken importiert. Da ich es recht kompakt halten möchte sind nur die wichtigsten erklärt und diese lauten wie folgt:

- Pandas für das Einlesen und Analysieren der Daten
- Matplotlib für Grafik Plots
- Numpy für die Bearbeitung von Arrays (Setze den Zufallswert für scikit-learn!)
- Scikit-Learn für das Preprocessing der Daten (Encoding, Scaling, Train/Test-Split)
- Tensorflow & Keras-API zur Erstellung der Modelle

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import class_weight
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,
BatchNormalization,LeakyReLU
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
```

Zuerst setzte ich den Seed auf 42. Dadurch wird gewährleistet, dass alle Zufallszahlen, die in meinem Code generiert werden, konsistent und reproduzierbar sind. Durch die Verwendung eines festen Seeds können Entwickler und Forscher sicherstellen, dass ihre Modelle und Analysen unter den gleichen Bedingungen durchgeführt werden. Dies erleichtert die Vergleichbarkeit und Validierung der Ergebnisse erheblich, da man sicher sein kann,

dass unterschiedliche Ausführungen des Codes zu den gleichen Resultaten führen. So können Fehlerquellen minimiert und die Zuverlässigkeit der Ergebnisse erhöht werden.

```
# Zufallskeim f r Reproduzierbarkeit setzen
tf.random.set_seed(42)
np.random.seed(42)
```

Nach dem Setzen des Seeds folgt das Laden und die Verarbeitung der Daten aus einer Excel-Datei, die Informationen über Brustkrebs enthält. Zunächst wird die Datei mit dem Befehl `df=pd.read_csv("breast-cancer.xls")` in einen DataFrame geladen. Dieser DataFrame enthält verschiedene Merkmale der Patienten sowie die Zielvariable, die angibt, ob der Tumor bösartig (M) oder gutartig (B) ist.

Um die Zielvariable für die weitere Analyse nutzbar zu machen, wird sie mit dem `LabelEncoder` encodiert. Dabei wird die Diagnose "Mïn den Wert 1 und "Bïn den Wert 0 umgewandelt. Dies geschieht durch den Befehl `y = le.fit_transform(df['diagnosis'])`, wobei die encodierten Labels anschließend wieder in den DataFrame eingefügt werden.

Im nächsten Schritt werden nicht benötigte Spalten, wie die ID der Patienten und die ursprüngliche Diagnose, entfernt, um den DataFrame zu bereinigen. Dies geschieht mit `X = df.drop(['id', 'diagnosis'], axis=1)`, wodurch nur die relevanten Merkmale für die Analyse übrig bleiben.

Abschließend werden die Daten mit dem `StandardScaler` skaliert. Dies ist wichtig, um sicherzustellen, dass alle Merkmale im gleichen Maßstab liegen, was die Leistung vieler maschineller Lernalgorithmen verbessert. Der Befehl `X_scaled = scaler.fit_transform(X)` führt die Skalierung durch und erstellt einen neuen DataFrame, der die skalierten Merkmale enthält.

Insgesamt bereitet dieser Code die Daten für die weitere Analyse oder das Training eines Modells vor, indem er die Zielvariable encodiert, nicht benötigte Informationen entfernt und die verbleibenden Daten skaliert werden.

```
# Korrelationen berechnen, um wichtige Features zu identifizieren
corr_matrix = df.corr()
corr_target = abs(corr_matrix["diagnosis"])
relevant_features = corr_target[corr_target > 0.5]
relevant_features = relevant_features.drop('diagnosis')
selected_features = relevant_features.index.tolist()
print("Ausgew hlte Features:", selected_features)

# Datensatz mit ausgew hlten Features erstellen
X_selected = df[selected_features]
X_scaled_selected = scaler.fit_transform(X_selected)
```

Danach wird eine Korrelation zwischen den Merkmalen eines Datensatzes berechnet, um wichtige Features zu identifizieren, die einen signifikanten Einfluss auf die Zielvariable haben. In diesem Fall die `"diagnosis"`, haben. Zunächst wird eine Korrelationsmatrix erstellt, die die Beziehungen zwischen allen numerischen Variablen im DataFrame `df` darstellt. Dies geschieht durch den Befehl `df.corr()`, der eine Matrix zurückgibt, in der jede Zelle den Korrelationskoeffizienten zwischen zwei Variablen anzeigt.

Anschließend wird die absolute Korrelation der Merkmale mit der Zielvariable `"diagnosis"` extrahiert. Dies geschieht durch `abs(corr_matrix["diagnosis"])`,

was sicherstellt, dass sowohl positive als auch negative Korrelationen berücksichtigt werden. Die nächsten Schritte beinhalten das Filtern der Merkmale, die eine Korrelation von mehr als 0.5 aufweisen. Diese Schwelle deutet darauf hin, dass nur Merkmale mit einer starken Beziehung zur Zielvariable ausgewählt werden.

Die Zeile `relevant_features = relevant_features.drop('diagnosis')` entfernt die Zielvariable selbst aus der Liste der relevanten Merkmale, da sie nicht als Feature betrachtet werden sollte. Die ausgewählten Merkmale werden dann in einer Liste `selected_features` gespeichert, die anschließend ausgegeben wird. Dies geschieht durch den Befehl `print(Äusgewählte Features:", selected_features)`.

Im letzten Teil des Codes wird ein neuer Datensatz `X_selected` erstellt, der nur die ausgewählten Merkmale enthält. Um sicherzustellen, dass die Daten für die weitere Analyse oder Modellierung geeignet sind, wird eine Skalierung der ausgewählten Merkmale durchgeführt. Dies geschieht durch den Befehl `scaler.fit_transform(X_selected)`, der die Merkmale standardisiert, sodass sie einen Mittelwert von 0 und eine Standardabweichung von 1 haben.

Zusammengefasst ermöglicht dieser Codeabschnitt, die wichtigsten Merkmale zu identifizieren und diese für die weitere Analyse oder Modellierung vorzubereiten, indem er sie skaliert.

```
1
2  df = pd.read_csv("breast-cancer.xls")
3
4  # Zielvariable encodieren (M = 1, B = 0)
5  le = LabelEncoder()
6  y = le.fit_transform(df['diagnosis'])
7
8  # Encodierte Labels in den DataFrame einf gen
9  df['diagnosis'] = y
10
11 # Nicht ben tigte Spalten entfernen
12 X = df.drop(['id', 'diagnosis'], axis=1)
13
14 # Daten skalieren
15 scaler = StandardScaler()
16 X_scaled = scaler.fit_transform(X)
```

Nun folgt die Bereitstellung des Datensatzes für maschinelles Lernen vor, indem er ihn in Trainings- und Testdaten aufteilt und die Trainingsdaten mit SMOTE (*Synthetic Minority Over-sampling Technique*) ausgleicht. Dies ist besonders wichtig bei unausgewogenen Klassen, um Verzerrungen in der Modellleistung zu vermeiden.

Die Aufteilung erfolgt mit der Funktion `train_test_split`, wobei 20 % der Daten als Testdaten verwendet werden. Der Parameter `random_state=42` sorgt für Reproduzierbarkeit, und `stratify=y` stellt sicher, dass die Klassenverteilung in beiden Datensätzen gleich bleibt.

Mit SMOTE werden synthetische Beispiele der unterrepräsentierten Klasse erzeugt, um ein ausgewogenes Trainingsset zu erstellen. Obwohl die Daten nun ausgeglichen sind, werden optional Klassen-Gewichte berechnet, um das Modell weiter zu optimieren.

Zusammengefasst sorgt der Code dafür, dass die Klassenverteilung ausgeglichen ist, was zu einer besseren Modellleistung und Generalisierbarkeit führt.

```
1
2  # Daten in Trainings- und Testdaten aufteilen
3  X_train, X_test, y_train, y_test = train_test_split(
4  X_scaled_selected, y,
5  test_size=0.2, random_state=42, stratify=y)
6
7  # Datensatz mit SMOTE balancieren
8  sm = SMOTE(random_state=42)
9  X_train_resampled, y_train_resampled = sm.fit_resample(X_train,
10 y_train)
11
12 #Klassen-Gewichte berechnen (optional, da Daten balanciert sind)
13 class_weights = class_weight.compute_class_weight('balanced',
14 classes=np.unique(y_train_resampled), y=y_train_resampled)
15 class_weights = dict(enumerate(class_weights))
```

Nach der sorgfältigen Vorbereitung der Daten folgt nun der finale Schritt, in dem ein neuronales Netzwerk-Modell definiert und trainiert wird. Der Code verwendet die `Keras`-Bibliothek, um ein einfaches, aber effektives Modell für die binäre Klassifikation zu erstellen. Zunächst wird ein sequentielles Modell erstellt, das aus mehreren Schichten besteht. Die erste Schicht ist eine `Dense`-Schicht mit 512 Neuronen, die die Eingabedaten erwartet, deren Form durch `input_shape=(X_train.shape[1],)` definiert wird. Um Überanpassung zu vermeiden, wird ein L2-Regularisierer mit einem Wert von 0.001 hinzugefügt.

Nach dieser Schicht folgt eine `BatchNormalization`, die hilft, die Trainingsgeschwindigkeit zu erhöhen und die Stabilität des Modells zu verbessern. Die Aktivierungsfunktion ist `LeakyReLU` mit einem Alpha-Wert von 0.1, die es ermöglicht, dass auch negative Werte durch die Schicht fließen, was die Lernfähigkeit des Modells verbessert. Schließlich wird eine `Dropout`-Schicht mit einer Rate von 0.3 hinzugefügt, um Überanpassung zu reduzieren, indem zufällig 30 % der Neuronen während des Trainings deaktiviert werden.

Diese Struktur wird durch mehrere weitere `Dense`-Schichten mit abnehmender Neuronenanzahl (`256`, `128`, `64` und `32`) fortgesetzt, wobei jede Schicht die gleichen Regularisierungs-, Normalisierungs- und Aktivierungsmechanismen verwendet. Die letzte Schicht ist eine `Dense`-Schicht mit einem Neuron und der `sigmoid`-Aktivierungsfunktion, die für die binäre Klassifikation geeignet ist, da sie Ausgaben zwischen 0 und 1 erzeugt.

Nachdem das Modell definiert wurde, erfolgt die Kompilierung. Hierbei wird der `Adam`-Optimierer mit einer Lernrate von 0.001 verwendet, und die Verlustfunktion ist `binary_crossentropy`, die für binäre Klassifikationsprobleme geeignet ist. Zusätzlich wird die Metrik `accuracy` zur Bewertung der Modellleistung während des Trainings verwendet.

Um das Training zu optimieren, werden zwei Callbacks definiert:
`EarlyStopping` und `ReduceLROnPlateau`. Der `EarlyStopping`-Callback überwacht den Validierungsverlust (`val_loss`) und stoppt das Training, wenn sich dieser über 10 Epochen nicht verbessert, wobei die besten Gewichte des Modells wiederhergestellt werden. Der `ReduceLROnPlateau`-Callback reduziert die Lernrate um den Faktor 0.5, wenn der Validierungsverlust über 5 Epochen stagnierend ist, und stellt sicher, dass die Lernrate nicht unter 1e-5 fällt.

Schließlich wird das Modell mit der Methode `fit` trainiert, wobei die resampelten Trainingsdaten `X_train_resampled` und `y_train_resampled` verwendet werden. Der Datensatz wird in Trainings- und Validierungsdaten aufgeteilt, wobei 20 % für die Validierung reserviert sind. Das Training erfolgt über maximal 100 Epochen mit einer Batch-Größe von 32, und die definierten Callbacks werden aktiviert, um das Training zu optimieren.

```
# Modell definieren (vereinfachte Architektur)
model = Sequential([
    Dense(512, input_shape=(X_train.shape[1],),
    kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.3),

    Dense(256, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.3),

    Dense(128, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.3),

    Dense(64, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.3),

    Dense(32, kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    LeakyReLU(alpha=0.1),
    Dropout(0.3),

    Dense(1, activation='sigmoid')  # For binary classification
])

# Modell kompilieren mit angepasstem Optimierer und Lernrate
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Callbacks f r fr hes Stoppen und Lernratenreduzierung
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=5,
min_lr=1e-5)

# Modell trainieren
history = model.fit(X_train_resampled, y_train_resampled,
                    validation_split=0.2,
                    epochs=100,
                    batch_size=32,
                    callbacks=[early_stopping, reduce_lr],
                    verbose=1)
```

Nachdem das Modell erstellt und trainiert wurde, folgt die Evaluierung der Modellleistung anhand der Testdaten. In diesem Schritt werden verschiedene Leistungskennzahlen wie **Genauigkeit** (`accuracy`), **Präzision** (`precision`), **Recall** (`recall`) und der **F1-Score** berechnet, um die Effektivität des Modells bei der Klassifizierung zu bewerten. Diese Metriken bieten einen umfassenden Überblick darüber, wie gut das Modell sowohl positive als auch

negative Klassen erkennt und wie ausgewogen die Ergebnisse sind.

Anschließend werden Vorhersagen auf den Testdaten durchgeführt, um die Fähigkeit des Modells zu demonstrieren, neue, ungesehene Daten korrekt zu klassifizieren. Dies erfolgt durch die Methode `model.predict()`, die Wahrscheinlichkeiten für jede Klasse generiert. Diese Wahrscheinlichkeiten werden dann in binäre Klassifikationen umgewandelt, indem ein Schwellenwert (`threshold`) definiert wird.

Die Evaluierung ermöglicht es, die Stärken und Schwächen des Modells zu identifizieren und gegebenenfalls weitere Optimierungen vorzunehmen. Beispielsweise können fehlklassifizierte Fälle analysiert werden, um Muster oder spezifische Merkmale zu erkennen, die zu Fehlentscheidungen führen. Basierend auf diesen Erkenntnissen können Anpassungen am Modell vorgenommen werden, wie die Änderung von Hyperparametern, die Hinzufügung weiterer Schichten oder die Verbesserung der Datenvorverarbeitung.

```
# Modell auf Testdaten evaluieren
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")
# Vorhersagen auf Testdaten treffen
y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)
```

In diesem Code Abschnitt werden zusätzliche Metriken zur Bewertung der Modellleistung berechnet. Zunächst wird der Klassifikationsbericht mit `print(classification_report( y_test, y_pred))` ausgegeben, der Präzision, Recall und F1-Score für jede Klasse anzeigt. Zudem wird der ROC AUC Score mit `auc = roc_auc_score(y_test, y_pred_proba)` berechnet und mit `print(f"ROC AUC Score: {auc}")` ausgegeben. Diese Metriken bieten eine umfassende Bewertung der Modellleistung und helfen, Stärken und Schwächen zu identifizieren, um die Vorhersagegenauigkeit zu verbessern.

Am ende wird die Konfusionsmatrix zur Bewertung der Modellleistung erstellt.
Die tatsächlichen Klassen (y_test)$werden mit den vorhergesagten Klassen$($y_pred$)
verglichen und als Heatmap visualisiert.
Dabei werden die Achsen beschriftet und das Diagramm dargestellt.

```
# Zus tzliche Metriken berechnen
print("Klassifikationsbericht:")
print(classification_report(y_test, y_pred))
# ROC AUC Score berechnen
auc = roc_auc_score(y_test, y_pred_proba)
print(f"ROC AUC Score: {auc}")
```

Zusätzlich wird die Trainingshistorie des Modells analysiert, indem die Entwicklung von Genauigkeit und Verlust über die Epochen visualisiert wird. Zwei Diagramme zeigen die Trainings- und Validierungsgenauigkeit sowie den Trainings- und Validierungsverlust, um den Lernfortschritt des Modells zu bewerten.

3 Ergebnisse

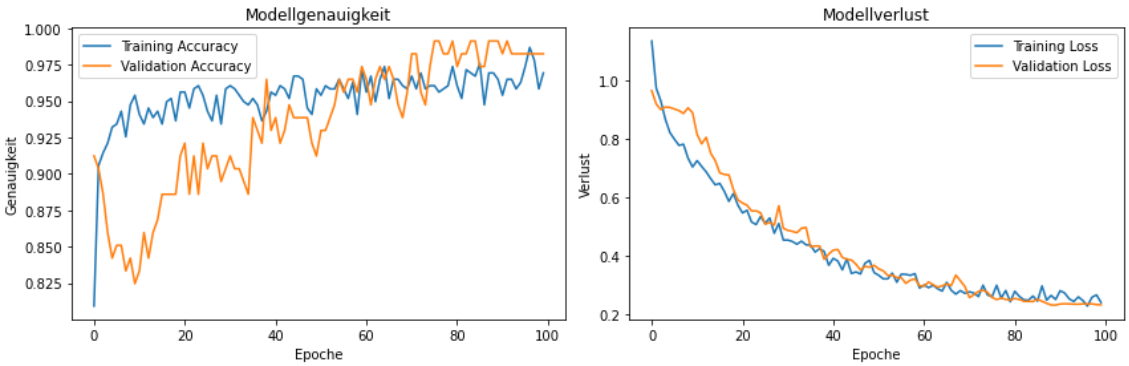

Abbildung 1: Modellgenauigkeit & Modellverlust

Die obere Abbildung zeigt die Modellgenauigkeit (links) und den Modellverlust (rechts) über die Epochen hinweg, dargestellt.

- Modellgenauigkeit (links):
  Die Grafik zeigt, dass die Trainingsgenauigkeit (blaue Linie) und die Validierungsgenauigkeit (orange Linie) zu Beginn schnell ansteigen, was auf ein effektives Lernen des Modells hindeutet. Die anfänglichen Schwankungen der Validierungsgenauigkeit stabilisieren sich im Verlauf des Trainings. Gegen Ende nähern sich beide Kurven an, mit einer maximalen Genauigkeit von etwa 0,98, was auf eine hohe Klassifikationsleistung und ein gutes Generalisierungsvermögen des Modells hinweist.
- Modellverlust (rechts): Der Verlust sinkt in den ersten Epochen schnell, was darauf hindeutet, dass das Modell in dieser Phase effektiv lernt. Ab etwa 40 Epochen verlaufen die Trainings- und Validierungsverlustkurven (blau und orange) parallel zueinander, was auf eine stabile Modellleistung hindeutet.
  Der Verlust stabilisiert sich bei etwa 0,2, was auf eine gute Anpassung des Modells an die Trainingsdaten schließen lässt.

Beide Abbildungen zeigen eine gute Übereinstimmung zwischen den Trainings- und Validierungsmetriken, ohne Hinweise auf Überanpassung.

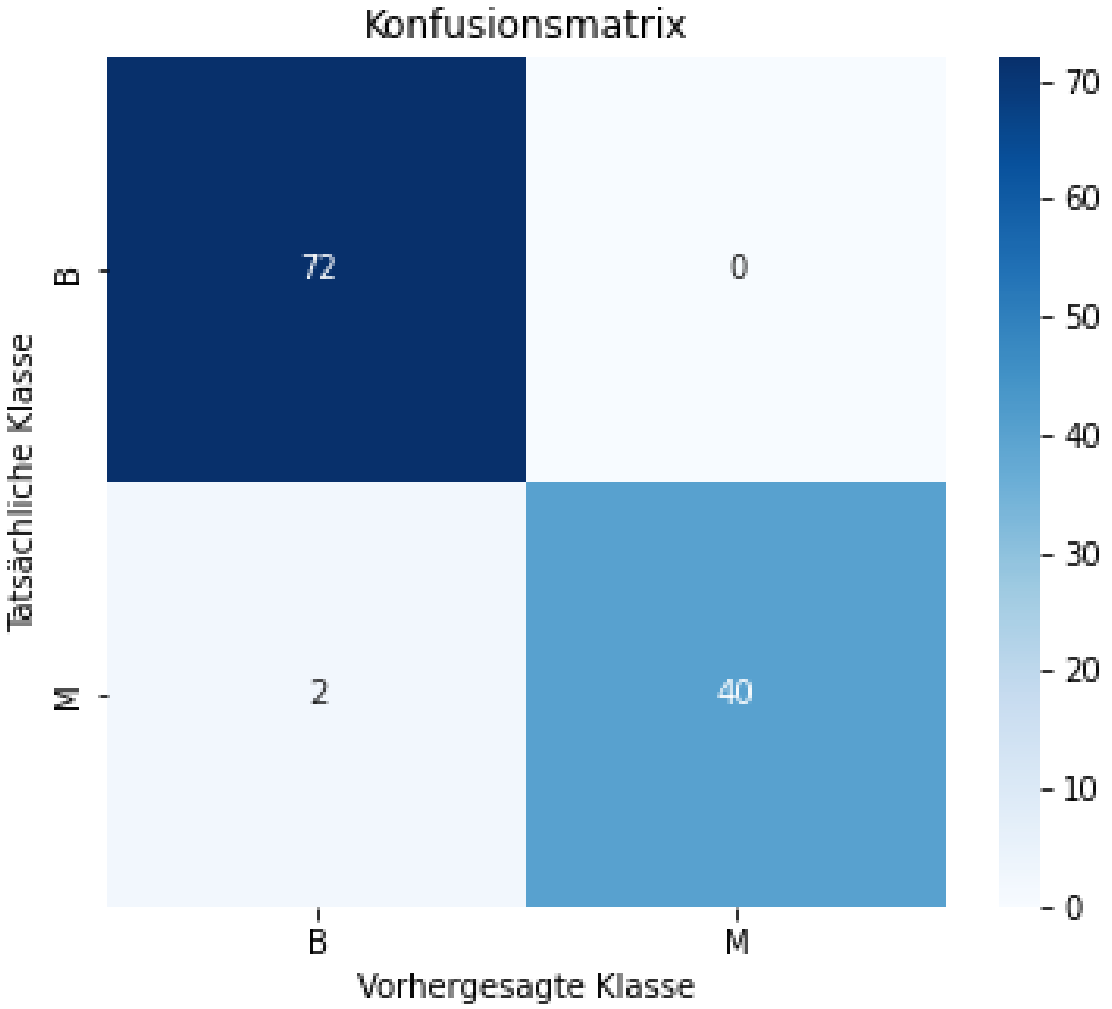

Abbildung 2: Konfusionsmatrix

Die obere Abbildung 2 zeigt eine Konfusionsmatrix. Diese gibt einen Überblick über die Leistung des Klassifikationsmodells bei der Vorhersage der Klassen "B"(benigne) und "M"(maligne).

- Richtig Positive (M korrekt als M klassifiziert): 40 Fälle.
- Richtig Negative (B korrekt als B klassifiziert): 72 Fälle.
- Falsch Positive (B fälschlicherweise als M klassifiziert): 0 Fälle.
- Falsch Negative (M fälschlicherweise als B klassifiziert): 2 Fälle.

Das Modell klassifiziert die meisten Beispiele korrekt. Es gab jedoch 2 Fälle, in denen maligne Tumore fälschlicherweise als benigne klassifiziert wurden (falsch negative). Keine benignen Tumore wurden fälschlicherweise als maligne klassifiziert (keine falsch positiven). Insgesamt zeigt das Modell eine hohe Genauigkeit und eine gute Fähigkeit, zwischen den Klassen zu unterscheiden, weist aber eine kleine Fehlerquote bei der Erkennung maligner Fälle auf.

```
Klassifikationsbericht:
              precision    recall  f1-score
support

           0       0.97      1.00      0.99
72
           1       1.00      0.95      0.98
42

    accuracy                           0.98
114
   macro avg       0.99      0.98      0.98
114
weighted avg       0.98      0.98      0.98
114

ROC AUC Score: 1.0
```

Abbildung 3: Klassifikationsbericht des Modells

Die Abbildung 3 zeigt die Ausgabe des Klassifikationsberichts auf der Konsole, welcher verschiedene Metriken zur Bewertung der Modellleistung präsentiert. Diese werden unten näher erläutert:

- Klasse 0 (benigne Tumore):
  - Precision (Präzision): 0,97 - Von allen als Klasse 0 vorhergesagten Fällen waren 97 % tatsächlich Klasse 0.
  - Recall (Empfindlichkeit): 1,00 - Alle tatsächlichen Klasse-0-Fälle wurden korrekt erkannt.
  - F1-Score: 0,99 - Harmonisches Mittel von Präzision und Recall, zeigt eine nahezu perfekte Klassifikation.
- Klasse 1 (maligne Tumore):
  - Precision: 1,00 - Alle als Klasse 1 vorhergesagten Fälle waren korrekt.
  - Recall: 0,95 - 95 % der tatsächlichen Klasse-1-Fälle wurden korrekt erkannt.
  - F1-Score: 0,98 - Hohe Leistung bei der Klassifikation maligner Tumore, aber leicht geringer als für Klasse 0.
- Gesamtgenauigkeit: 0,98 - 98 % der Vorhersagen des Modells waren korrekt.
- Durchschnittswerte:
  - Macro Average: 0,99 (Präzision), 0,98 (Recall), 0,98 (F1-Score) - Durchschnitt über beide Klassen hinweg, ohne Gewichtung.
  - Weighted Average: 0,98 für Präzision, Recall und F1-Score - Durchschnitt unter Berücksichtigung der Größe der Klassen.
- ROC AUC Score: 1,0 - Das Modell unterscheidet perfekt zwischen den Klassen.

4 Ausblick

Durch den Einsatz eines einfachen neuronalen Netzes und die Anwendung etablierter Methoden zur Vermeidung von Overfitting ist es gelungen, ein Modell zu erstellen, das die Daten der Feinnadelbiopsie zuverlässig in gesundes und krebsartiges Gewebe klassifiziert. Mit einer Genauigkeit von 97,8 % wurde das gesetzte Ziel erreicht. Aufgrund der Wichtigkeit (Kritikalität) des Themas sollte jedoch die weitere Erhöhung der Genauigkeit die oberste Prämisse für zukünftige Bestrebungen sein. Bereits eine einzelne falsche Diagnose kann ein Leben negativ beeinflussen und muss daher vermieden werden. In diesem Zusammenhang sollte bei weiteren Untersuchungen ein größeres Augenmerk auf die Unterscheidung zwischen „Falsch-Positiven“ und „Falsch-Negativen“ Befunden gelegt werden. Eine „Falsch-Negative“ Diagnose ist im Zweifelsfall deutlich kritischer zu bewerten als eine „Falsch-Positive“, da notwendige Behandlungsschritte ausbleiben. Diese Unterscheidung wurde in der vorliegenden Untersuchung aufgrund des begrenzten Zeitrahmens nicht getroffen. Ein erster wichtiger Schritt zur Verbesserung der Modelle sollte in der Vergrößerung des Datensatzes liegen. Obwohl der Datensatz mit 569 Einträgen eine gute Größe für unser erstes Projekt im Bereich des maschinellen Lernens darstellt, ist die Menge des Datensatzes nicht groß genug, um wirklich signifikante Modelle zu trainieren. Hierfür sollte eine deutliche Erhöhung der Datenmenge angestrebt werden. Ein guter Richtwert wäre eine Vergrößerung um das 100-Fache, um das Modell noch besser zu trainieren.