



HAUTKREBS: ERSTELLUNG EINER DEEP LEARNING
MODELL FÜR DIE DIAGNOSE DER KREBSSTELLEN
AUF DER MENSCHLICHEN HAUT MITTELS PYTHON
UND INTEGRATION DES MODELLS IN EINER
RESPONSIVEN WEBSEITE MITTELS
DJANGO-FRAMEWORK

Abschlussarbeit

im Studiengang Medizintechnik
Hochschule Koblenz, RheinAhrCampus
Remagen

vorgelegt von **Ziad Bari**

Matrikelnummer: 537846
geboren am 01.01.1996 in Edlib

Erstprüfer: Prof. Dr. Jens Bongartz

Zweitprüfer: Prof. Dr. Matthias Kohl

Externer Gutachter: Ing. Mahmoud Ahmed

Durchgeführt bei: Mystro GmbH

Remagen, 19.02.2024

Zusammenfassung

Das Ziel des Projekts besteht darin, ein Deep Learning-Modell zur Diagnose von Hautkrebsläsionen auf der menschlichen Haut zu erstellen. Dies wird mit Hilfe der Programmiersprache Python realisiert. Zunächst wurde eine umfangreiche Menge von der Bilddaten **der bösartige Hautkrebsläsionen und gutartige Hautflecken (Hautveränderungen)** gesammelt. Diese Bilddaten dienen als Trainingsdatensatz für das Deep Learning-Modell.

In diesem Projekt wurde Python und TensorFlow-Bibliothek verwendet, um ein convolutional neuronales Netzwerk aufzubauen und zu trainieren. Das Modell wird so programmiert, dass es die Hautbilder analysiert und Krebsstellen erkennt. Durch die Verwendung von Deep Learning-Techniken kann das Modell lernen, Muster und Merkmale in den Hautläsionen zu erkennen, die auf Hautkrebs hinweisen.

Nachdem das Modell entwickelt und trainiert wurde, geht es darum, es in eine responsive Webseite zu integrieren. Hier kommt das Django-Framework ins Spiel. Django ermöglicht es dann, eine benutzerfreundliche und interaktive Webseite zu erstellen, auf der Benutzer Hautbilder hochladen können.

Die Webseite verwendet das entwickelte Deep Learning-Modell, um die hochgeladenen Bilder zu analysieren und eine Diagnose für Hautveränderung zu erstellen. Das Ergebnis wird den Benutzern präsentiert, möglicherweise mit zusätzlichen Informationen und Vorschlägen für weitere Schritte.

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Abkürzungsverzeichnis	VII
1 Grundlagen	1
1.1 Einführung in künstliche Intelligenz	1
1.2 Maschinelles Lernen	2
1.3 Deep Learning und neuronale Netzwerke	3
1.3.1 Neuron im Gehirn	3
1.3.2 Neuronale Netzwerke	4
2 Convolutional Neural Network (CNN)	7
2.1 Bilderkennung im conventionell neuronalen Netzwerken	7
2.2 Verlustbehaftete Schichten	8
2.2.1 Das Reduzieren der Dimension des Eingabebildes in einem CNN	8
2.2.2 Convolutional Layer	9
2.2.3 Pooling Layer	12
2.2.4 Dropout und Flatten-Techniken	13
2.2.5 Die Wiederholung von Convolutional Layers und Pooling Layers in einem CNN	15
2.3 Vollvernetzte (neuronale) Schichten	15
2.3.1 Aktivierungsfunktionen in CNN	17
3 Datensatz von Hautkrebsbildern	21
3.1 Entstehung des Hautkrebses	21
3.2 Hautkrebserkennung und Bilderquellen	23
4 Implementierung von CNNs in Deep Learning-Bilderklassifikation	26
4.1 Programmierwerkzeuge	26
4.1.1 TensorFlow	26
4.1.2 Keras	27

4.2	Binäre Hautkrebs-Bilderklassifikation	28
4.2.1	CNN-Modell für die binäre Hautbilderklassifikation	28
4.2.2	Training des Modells	34
4.2.3	Evaluation des CNN-Modells	37
4.2.4	Testen des CNN-Modells	38
5	Skinlib	41
5.1	Einführung in Skinlib	41
5.2	CNN-Modell in Skinlib	42
5.2.1	Importieren aller im Code verwendeten Bibliotheken	43
5.2.2	Abrufen der Training und Test-Bilderdaten von ISIC	44
5.2.3	Erstellung von Bild-data-Generator	45
5.2.4	Bilderanzeigen von einem Batch in Training-Bilddaten	45
5.2.5	Transferlernen mittels EfficientNetB5 von Keras	47
5.2.6	Erstellung des CNN (convolutional neuronales Netzwerk)-Modell	51
5.2.7	Trainieren des erstellten CNN-Modells	55
5.2.8	Darstellung der CNN-Modellleistung	56
5.2.9	Evaluation des Skinlib CNN-Modells	57
5.2.10	Speichern und Laden des CNN-Modells	59
5.3	Skinlib-Webseite	60
5.3.1	Skinlib-Frontend	60
5.3.2	Skinlib-Backend	61
5.3.3	Skinlib Django-Projekt	62
5.3.4	Integration des CNN-Modells in skinlib-Django-Projekt	66
6	Ergebnisse	71
7	Programmcode	72
Literatur		73

Abbildungsverzeichnis

1.1	Einfache Struktur des KI [1]	1
1.2	Einfache Struktur des Maschinellen Lernen [2]	2
1.3	Neuron im Gehirn [3]	3
1.4	Neurales Netzwerk [3]	4
1.5	Neuronales Netzwerk mit 1 hidden layers [3]	5
1.6	Neuronale Netzwerke in Deep learning [4]	5
2.1	Convolutional neurale Netzwerk [4]	7
2.2	Convolutional Layer 1 [5]	9
2.3	Convolutional Layer 2 [5]	10
2.4	Convolutional Layer 3 [5]	11
2.5	Ausgabematrix von Convolutional Layer [5]	11
2.6	Pooling Layer [6]	12
2.7	Dropout-Technik [7]	13
2.8	Flatten-Technik [8]	14
2.9	Die Wiederholung von Convolutional Layers und Pooling Layers in einem CNN [4]	15
2.10	Einfach vollvernetzte (neuronale) Schichten [9]	16
2.11	ReLU (Rectified Linear Unit) [10]	18
2.12	Sigmoid-Funktion [10]	18
2.13	Softmax-Funktion [11]	19
2.14	Softmax-Funktion Mathe-Function [11]	19
3.1	Gutartige und Bösartige Tumoren [15]	21
3.2	The International Skin Imaging Collaboration Webseite [12]	23
3.3	Melanocytic nevus	25
3.4	Melanoma	25
3.5	Squamous cell carcinoma	25
3.6	Vascular lesion	25
3.7	Dermatofibroma	25

3.8	Benign keratosis	25
3.9	Basal cell carcinoma	25
3.10	Actinic keratosi	25
4.1	TensorFlow	26
4.2	Keras	27
4.3	Dataset KerasAPI	29
4.4	4 Batch-Bildern von Datensatz	30
4.5	CNN-Algorithmus für die binäre Hautbilderklassifikation	31
4.6	Training des Modells	34
4.7	Ersten 10 Epochen bei Training des Modells	35
4.8	Loss und Accuracy bei Trainingsprozess	36
4.9	Evaluation des CNN-Modells	37
4.10	originales Testbild	38
4.11	Verarbeitetes Testbild	39
4.12	Vorhersage für das Testbild	40
5.1	Implementierungsplan des CNN-Modell in Skinlib	42
5.2	Importieren aller im Code verwendeten Bibliotheken	43
5.3	Abrufen der Training und Test-Bilderdaten von ISIC	44
5.4	Pandas-DataFrame-Objek	44
5.5	Erstellung von Bild-data-Generator	45
5.6	Bilderanzeigen von einem Batch in Training-Bilddaten	46
5.7	Plot-Figure Batch in Training-Bilddaten	46
5.8	KerasAPI-Modellen [18]	47
5.9	EfficientNetB5-Architektur [19]	48
5.10	AlexNet vs EfficientNet [20]	49
5.11	Transfer learning [21]	50
5.12	Erstellung des CNN-Modell basierend auf der Keras/EfficientNetB5-Architektur	51
5.13	Zusammenfassung des Modells-Skinlib	54
5.14	Trainieren des erstellten CNN-Modells	55
5.15	Epochen während des Trainierens des erstellten CNN-Modells	56
5.16	Darstellung der CNN-Modellleistung	57
5.17	Evaluation des Skinlib CNN-Modells	58
5.18	Confusion-Matrix des CNN-Modells	59
5.19	Speichern und Laden des CNN-Modells	59

5.20	Django – Model Views Template [14]	61
5.21	Skinlib Django virtuelle Umgebung Terminal 1	62
5.22	Skinlib Django virtuelle Umgebung Terminal 2	64
5.23	Skinlib Django server	65
5.24	skinlib Django Visual studio Code	65
5.25	skinlib Django models.py Prediction-class	66
5.26	skinlib Django views.py predict-Funktion	67
5.27	skinlib Django predict.html Image-Form	69
7.1	skinlib Github-repo	72

Abkürzungsverzeichnis

KI	Künstliche Intelligenz	2
ML	Maschinelles Lernen	2
DL	Deep learning	3
CNN	Convolutional Neural Network	9
ReLU	Rectified Linear Unit	17
HTML	(Hypertext Markup Language	60
CSS	Cascading Style Sheets	60
VS	Visual Studio Code	60

1 Grundlagen

1.1 Einführung in künstliche Intelligenz

Künstliche Intelligenz (KI) ist die Fähigkeit von Maschinen nachzubilden. Das bedeutet, die KI kann wie der Mensch aus Erfahrungen lernen, indem sie Muster erkennt und dadurch flexibel auf neue Situationen reagiert.

Das Ziel der künstlichen Intelligenz besteht darin, Maschinen so zu gestalten, dass sie lernen, Probleme lösen und Entscheidungen treffen können, ähnlich wie Menschen dies tun würden.

Künstliche Intelligenz (KI) ist die Simulation menschlicher Intelligenzprozesse durch Maschinen, insbesondere durch Computersysteme. Im Allgemeinen funktionieren KI-Systeme, indem sie große Mengen gelabelter Trainingsdaten aufnehmen, die Daten auf Korrelationen und Muster analysieren und diese Muster nutzen, um Vorhersagen über zukünftige Zustände zu treffen.

KI erfordert eine Grundlage aus spezialisierter Hardware und Software zum Schreiben und Trainieren von Algorithmen für maschinelles Lernen. Es gibt keine Programmiersprache, die gleichbedeutend mit KI ist, aber einige, darunter Python, R und Java, sind sehr beliebt.

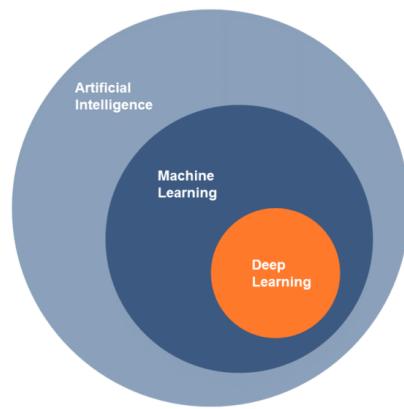


Abbildung 1.1: Einfache Struktur des KI [1]

Machine Learning (maschinelles Lernen, ML) und Deep Learning (tiefes Lernen, DL) in 1.1 sind Teilbereiche der Künstlichen Intelligenz. KI umfasst eine breite Palette von Techniken, darunter maschinelles Lernen und Deep Learning.

1.2 Maschinelles Lernen

Maschinelles Lernen (ML) spielt heutzutage technisch eine große Rolle. Darunter versteht man das Grundkonzept, dass der Computer die entsprechende Fähigkeiten zeigen soll und er in der Regel Sachen lernt und dann auch entsprechend wiedererkennen kann.



Abbildung 1.2: Einfache Struktur des Maschinellen Lernen [2]

Maschinelles Lernen ist ein Teilbereich der Künstliche Intelligenz (KI), bei dem computergestützte Systeme lernen, Muster und Zusammenhänge in Daten zu erkennen und daraus Vorhersagen oder Entscheidungen zu treffen. Siehe Abbildung 1.2. Es basiert auf Algorithmen und Statistik, wobei Computermodelle anhand von vorhandenen Daten trainiert werden, um bestimmte Aufgaben zu erfüllen. Das maschinelle Lernen ermöglicht es Maschinen, aus Erfahrungen zu lernen. Es findet in vielen Bereichen Anwendung, wie z.B. Spracherkennung, Bilderkennung, automatisierte Fahrzeuge und personalisierte Empfehlungssysteme.

1.3 Deep Learning und neuronale Netzwerke

Deep learning ist eine Methode der Informationsverarbeitung und ein Teilbereich des maschinelles Lernen. Deep learning (DL) nutzt neuronale Netzwerke, um große Datensätze zu analysieren. Die künstliche neurale Netze gehen hier wie die Menschen etwas vor, indem sie etwas wahrnehmen, darüber nachdenken und eine Schlussfolgerung daraus ziehen. Nur können die Maschinen und die Systeme viel schneller größere Datensätze untersuchen als wir Menschen möglich wäre.

1.3.1 Neuron im Gehirn

Die Idee neuronaler Netzwerke stammt von der Funktionsweise der Gehirnzellen 1.3, die das Denken und die Datenanalyse ermöglichen, um ähnliche Algorithmen zu entwerfen. Die Idee hat nichts mit menschlichem Denken zu tun, sondern mit der Art und Weise der Datenanalyse.

Im Gehirn sind Neuronen die grundlegenden Bausteine für die Verarbeitung von Informationen. Sie empfangen Signale von anderen Neuronen über Dendriten, verarbeiten diese Signale in ihrem Zellkörper, und geben dann ein Ausgangssignal über den Axon weiter. Diese Art der Informationsverarbeitung ermöglicht dem Gehirn, komplexe Muster zu erkennen und Aufgaben wie Sehen, Hören und Denken auszuführen.

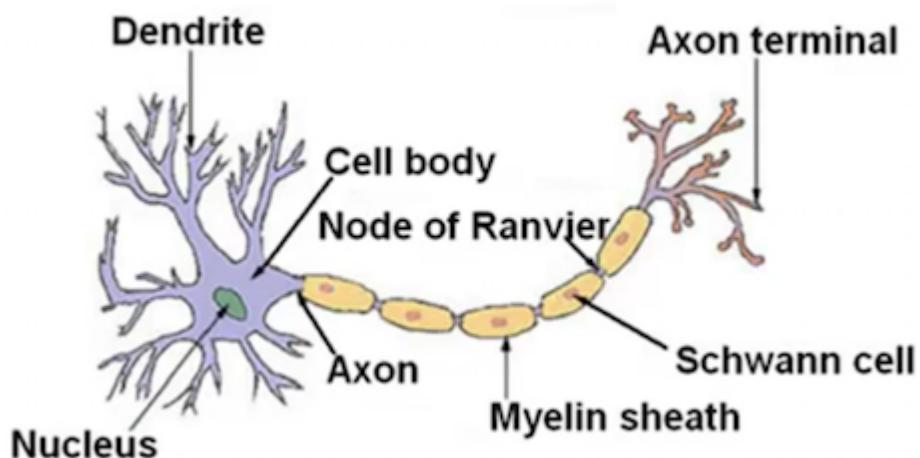


Abbildung 1.3: Neuron im Gehirn [3]

1.3.2 Neuronale Netzwerke

Um das oberen Struktur des Neurones im Gehirn als neuronale Netzwerke zu realisieren, muss dann das neuronale Netzwerke auf die gleiche Art und Weise des Neurones im Gehirn aufgebaut werden.

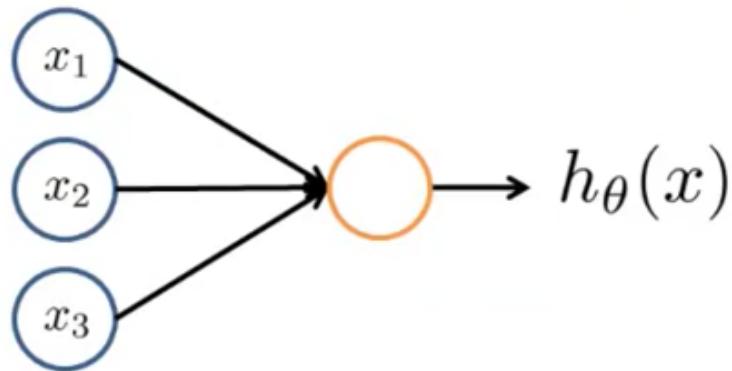


Abbildung 1.4: Neurales Netzwerk [3]

- Das neuronale Netzwerke ist so konzipiert, dass es dem Design einer neuronalen Zelle im Gehirn ähnelt.
- Der orange Kreis ist die Zelle.
- Im einfachen Modell 1.4 ähneln unsere Dendriten den Eingabemerkmalen ($X_1 \dots X_N$), und die Ausgabe ist das Ergebnis der Hypothesenfunktion $H_\theta(X)$ und hier kann ein oder mehrere Vorhersagen als Ergebnisse ausgegeben werden.

Ein neuronales Netzwerk ist also nichts anderes als ein Netzwerk von Neuronen, und jedes Neuron versucht, eine einfache Funktion zu lernen, und mit Hilfe dieser einfachen Neuronen versucht das Netzwerk, eine sehr komplexe Funktion zu lernen.

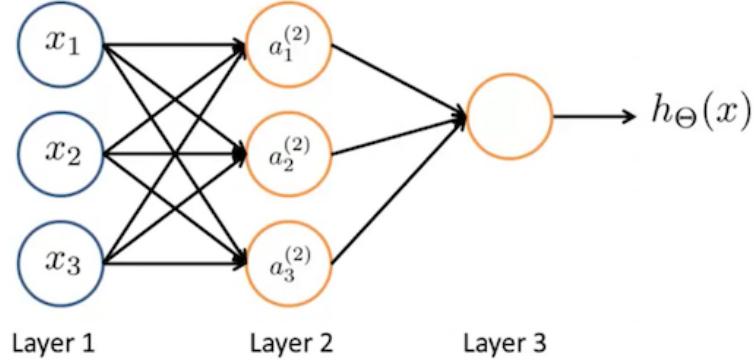


Abbildung 1.5: Neuronales Netzwerk mit 1 hidden layers [3]

Im obigen Bild 1.5 sieht man, dass es 3 Eingabemerkmale (Eingabeschichten oder Input-Layer) gibt und jede Eingabe durch jedes der Neuronen in den verborgenen (Hidden-Layers) Schicht geleitet wird, deren Ausgabe an die folgende Schicht (Aus-gabeschicht oder Output-Layer) weitergeleitet wird.

Es gibt generell 3 Arten von den neuronalen Netzwerke 1.6, die den Kern von Deep Learning Applikationen repräsentieren:

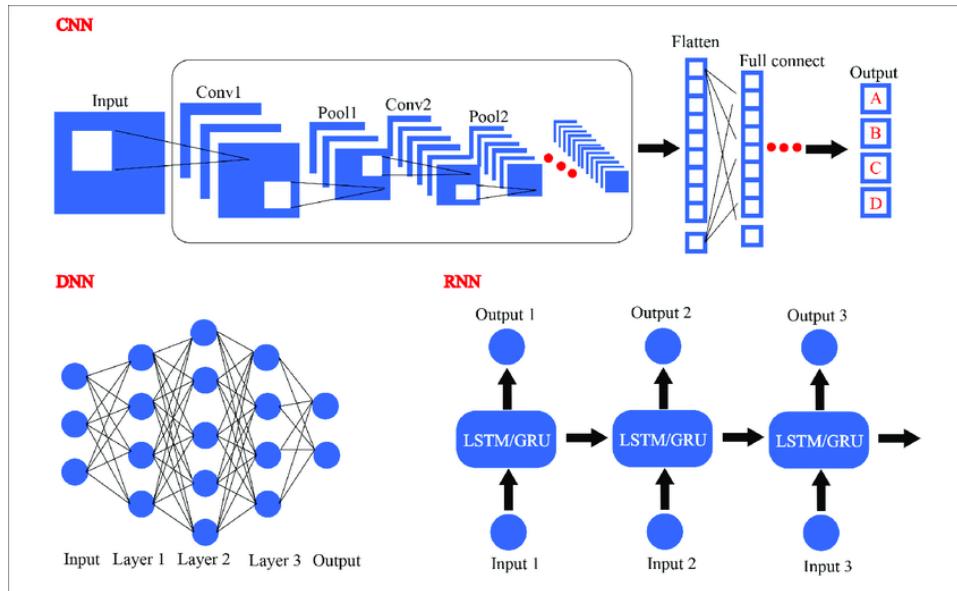


Abbildung 1.6: Neuronale Netzwerke in Deep learning [4]

- **RNN:** (Recurrent Neural Network): Ein rekurrentes neuronales Netzwerk ist eine Art von Netzwerk, das speziell für die Verarbeitung von Sequenzen entwi-

ckelt wurde. Im Gegensatz zu herkömmlichen neuronalen Netzwerken besitzt ein RNN eine Feedback-Verbindung, die es ermöglicht, Informationen über vorherige Zustände beizubehalten. Dadurch kann ein RNN langfristige Abhängigkeiten und Zusammenhänge in Sequenzen erfassen und zum Beispiel in der Sprachverarbeitung, Textgenerierung oder Zeitreihenanalyse eingesetzt werden.

- **DNN**: (Deep Neural Network): Ein tiefes neuronales Netzwerk ist ein Netzwerk, das aus mehreren Schichten von Neuronen besteht, die als Schichten bezeichnet werden. Jede Schicht besteht aus vielen künstlichen Neuronen, die mit Neuronen in den benachbarten Schichten verbunden sind. DNNs sind bekannt für ihre Fähigkeit, komplexe Funktionen zu lernen und in verschiedenen Bereichen wie Bilderkennung, Spracherkennung und natürlicher Sprachverarbeitung eingesetzt zu werden
- **CNN**: (Convolutional Neural Network): Ein faltendes neuronales Netzwerk ist eine spezialisierte Art von Netzwerk, die hauptsächlich in der Bildverarbeitung eingesetzt wird. CNNs verwenden spezielle Schichten, die als Faltungsschichten bezeichnet werden, um Merkmale aus den Eingabebildern zu extrahieren. Diese Schichten verwenden Faltungsoperationen, um lokale Muster und Strukturen in den Daten zu erkennen. Durch die Verwendung von CNNs können automatisch Merkmale wie Kanten, Texturen oder Formen aus den Bildern gelernt und für Klassifizierungsaufgaben verwendet werden.

2 Convolutional Neural Network (CNN)

2.1 Bilderkennung im conventionell neuronalen Netzwerken

Eine Convolutional Neural Network (CNN), auch als faltendes neuronales Netzwerk bezeichnet, ist eine Art von Deep Learning Modell, das häufig in der Bildverarbeitung verwendet wird. Es wurde entwickelt, um automatisch Muster in Bildern zu erkennen und zu klassifizieren. CNNs sind aufgrund ihrer Fähigkeit, räumliche Strukturen innerhalb von Daten zu erfassen, sehr effektiv bei der Verarbeitung von Bildern.

Faltungsneuronale Netzwerke lernen direkt aus den Daten und werden weit verbreitet für die Bilderkennung und Klassifizierung eingesetzt. CNNs gelten als eine der besten maschinellen Lernalgorithmen zur Analyse von gitterartig strukturierten Daten, wie sie in Bildern vorkommen. CNNs haben außergewöhnliche Leistungen bei Bildverarbeitungsaufgaben und Computer Vision-Aufgaben wie Lokalisierung und Segmentierung, Klassifizierung und Detektion gezeigt.

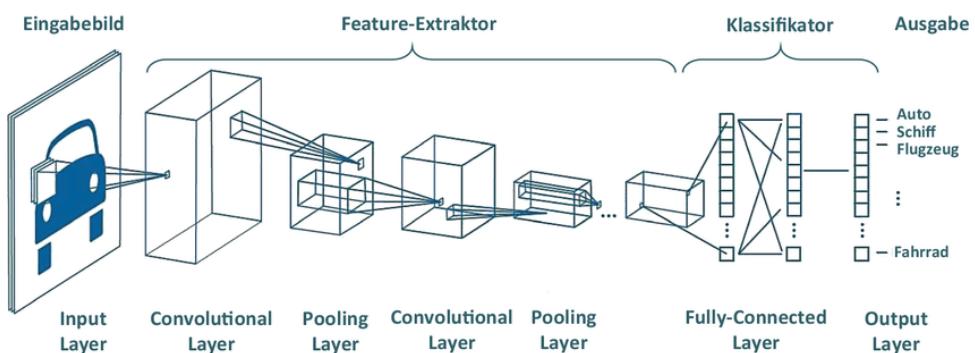


Abbildung 2.1: Convolutional neurale Netzwerk [4]

Ein typisches Faltungsneuronales Netzwerk wie in der Abbildung 2.1 enthält in der Regel zehn oder hunderte Schichten, von denen jede darauf trainiert werden kann, verschiedene Aspekte eines Bildes zu erkennen. Die Ausgabe jedes gefalteten Bildes wird nach der Anwendung von Filtern als Eingabe für die nächste Schicht verwendet,

während das Bild mit verschiedenen Auflösungen trainiert wird. Die Filter beginnen mit der Erkennung grundlegender Merkmale wie Helligkeit und Kanten und werden immer komplexer, bis sie Merkmale erkennen, die das Objekt eindeutig identifizieren. Zwischen den Eingabe- und Ausgabeschichten eines CNN befinden sich mehrere versteckte Schichten. Diese Schichten führen Operationen durch, die die Daten verändern, um spezifische Merkmale der Daten zu lernen. Faltung, Aktivierung (oder ReLU) und Pooling sind die am häufigsten verwendeten Schichten.

CNNs werden oft in Aufgaben wie Objekterkennung, Gesichtserkennung, Bildsegmentierung und Bildgenerierung eingesetzt. Sie haben zahlreiche Anwendungen in verschiedenen Bereichen wie Medical Imaging, autonomes Fahren, Überwachungssysteme und vieles mehr.

2.2 Verlustbehaftete Schichten

Die Architektur des Convolutional Neural Network hat einen convolutive Layer (Verlustbehaftete Schichten). Diese Verlustbehaftete Schichten sind typischerweise Verarbeitungsketten für das Eingabebild und haben verschiedene Stufen. Zunächst wird dann in Details erklärt, wie das Eingabebild bei jeder Schicht der Verlustbehafteten Schichten verarbeitet werden:

2.2.1 Das Reduzieren der Dimension des Eingabebildes in einem CNN

Das Reduzieren der Dimension des Eingabebildes in einem CNN (Convolutional Neural Network) vor der Weiterleitung an das Convolutional Layer hat mehrere Vorteile:

- Erlaubt das Lernen von allgemeineren Merkmalen: Durch die Reduzierung der Bildgröße werden kleine Details und Muster herausgefiltert, die möglicherweise für das Lernen allgemeinerer Merkmale weniger relevant sind. Dies ermöglicht dem CNN, sich auf bedeutendere Merkmale zu konzentrieren, die unabhängig von der spezifischen Größe des Bildes zu finden sind.
- Überwindung von Speicherproblemen: Größere Bilder beanspruchen mehr Speicherplatz. Durch die Reduzierung der Dimensionen können größere Datensätze oder mehr Schichten im CNN verwendet werden, ohne dass es zu Speicherproblemen kommt.

2.2.2 Convolutional Layer

Diese erste Schicht analysiert die eingegebenen Bilder und erkennt das Vorhandensein einer Reihe von Merkmalen.

Im Convolutional Neural Network (CNN) wird das Eingabebild als Matrix repräsentiert. Die Dimensionen der Matrix hängen von der Anzahl der Kanäle des Bildes ab. Für ein monochromes Graustufenbild wird die Matrix in der Regel eine zweidimensionale Matrix sein, in der jeder Eintrag den Grauwert eines einzelnen Pixels darstellt. Für ein Farbbild werden in der Regel drei Kanäle verwendet, um die Farbinformationen zu repräsentieren (z. B. Rot, Grün und Blau). Die Matrix wird dann eine dreidimensionale Matrix sein, wobei jede Dimension eine der Farbkomponenten repräsentiert. Jeder Eintrag in der Matrix enthält dann den Wert der entsprechenden Farbkomponente für einen bestimmten Pixel.

In den folgenden Beispiel 2.2 wird dann genau erklärt, wie das Eingabebild in der Faltungsstufe oder in englischen Convolutional Layer verarbeitet wird:

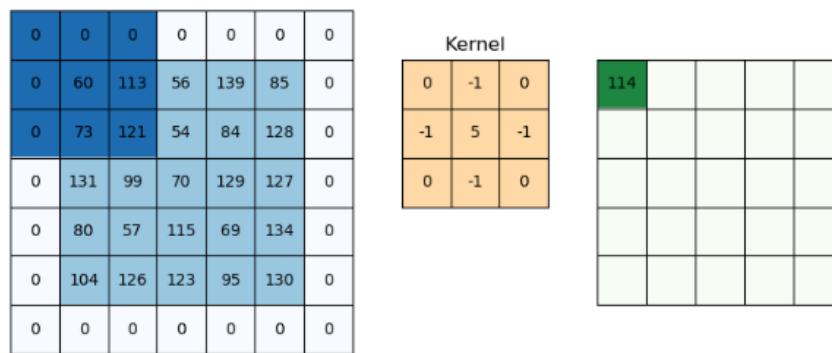


Abbildung 2.2: Convolutional Layer 1 [5]

Hier wird ein kleines Bildausschnitt (dunkel Blau) (3x3 Matrix) von der Eingabebild genommen und diese multiplizieren wir mit einer Matrix(eng. Kernel), die gleiche Größe (3x3 Matrix) hat und aus Koeffizienten besteht. Dann werden dann alle Terme addiert. Das ergibt sich dann eine neue Bildpunkt.

Die Elemente des Kernels oder die Faltungsmatrix sind Variable, die im Verlauf des Trainingsprozess angepasst und verbessert werden.

Im obigen Beispiel 2.2 resultiert das Ausgabearray durch folgendes:

$$\begin{aligned}
 y[0,0] &= (0 \cdot 0) + (0 \cdot -1) + (0 \cdot 0) \\
 &\quad + (0 \cdot -1) + (60 \cdot 5) + (113 \cdot -1) \\
 &\quad + (0 \cdot 0) + (73 \cdot -1) + (121 \cdot 0) = 114
 \end{aligned}$$

Diese Bildausschnitt in der Abbildung 2.3 und 2.4 wird dann in einer Pixel nach rechst zunächst verschoben, wobei dann weiter das ganzes Eingabebild mit dem Bildausschnitt abgedeckt wird und das gleiche mathematische Methode wird dann hier appliziert.

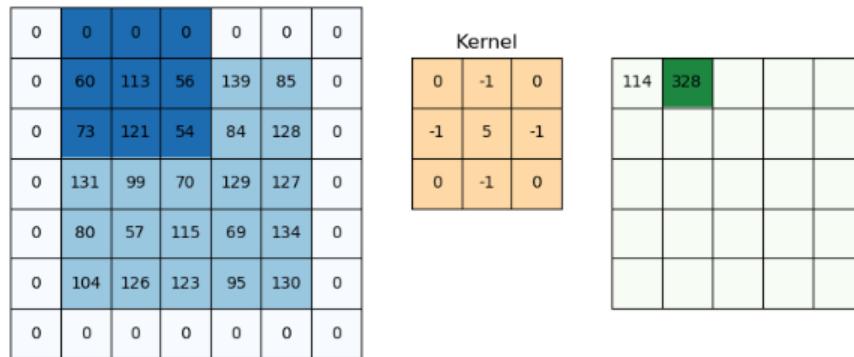


Abbildung 2.3: Convolutional Layer 2 [5]

$$\begin{aligned}
 y[0,1] &= (0 \cdot 0) + (0 \cdot -1) + (0 \cdot 0) \\
 &\quad + (60 \cdot -1) + (113 \cdot 5) + (56 \cdot -1) \\
 &\quad + (73 \cdot 0) + (121 \cdot -1) + (54 \cdot 0) = 328
 \end{aligned}$$

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26		

Abbildung 2.4: Convolutional Layer 3 [5]

$$\begin{aligned}
 y[0, 2] &= (0 \cdot 0) + (0 \cdot -1) + (0 \cdot 0) \\
 &\quad + (113 \cdot -1) + (56 \cdot 5) + (139 \cdot -1) \\
 &\quad + (121 \cdot 0) + (54 \cdot -1) + (64 \cdot 0) = -26
 \end{aligned}$$

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

Abbildung 2.5: Ausgabematrix von Convolutional Layer [5]

Bei jedem erreichten Bildausschnitt wird eine Konvolution berechnet, die eine Merkmalskarte (Feature Map) ergibt, um zu wissen, wo sich die Features im Bild befinden: Je höher die Feature Map ist, desto präziser ähnelt der gefilterte Bildausschnitt dem Feature.

Im Convolutional Layer (Faltungsschicht) wurden dann die Eingabebilddaten reduziert. Das macht man nicht mit einer Filter-Matrix, sondern mit vielen im Verlauf des Trainingsprozess, sodass die Eingabebilddaten letztendlich reduziert werden.

2.2.3 Pooling Layer

Der Pooling Layer ist eine Operation, die normalerweise zwischen zwei Convolutional Layers angewandt wird. Sie erhält als Eingabe die Feature Maps, die als Ausgabe der Convolutional Layers gebildet werden. Sie reduziert die Größe der Bilder und erhält gleichzeitig ihre wesentlichsten Merkmale. Zu den am häufigsten verwendeten Methoden gehören das bereits erwähnte Max-Pooling und das Average Pooling, bei dem der Durchschnittswert des Filterfensters in jedem Schritt beibehalten wird.

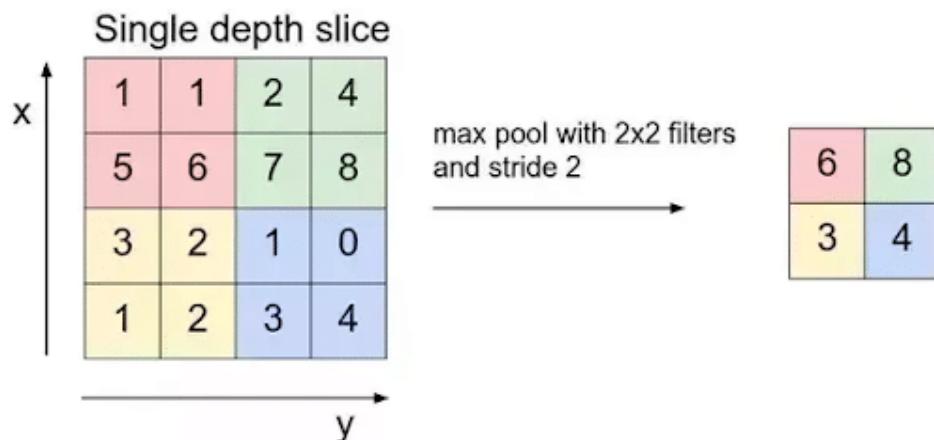


Abbildung 2.6: Pooling Layer [6]

Hier in diesem Beispiel 2.6 für Max-Pooling: Es ist eine 4×4 -Matrix, die die ursprüngliche Eingabe darstellt, und einen Filter eines Fensters der Größe 2×2 , den auf die Eingabe anwendet wird. Für jedes Subfeld, das der Filter abtastet, nimmt Max-Pooling das Maximum an. Dadurch entsteht gleichzeitig eine neue Ausgabematrix, in der jedes Element dem Maximum jedes abgetasteten Feldes entspricht.

2.2.4 Dropout und Flatten-Techniken

Die Dropout-Schicht ist eine Technik, die in Deep Learning-Modellen verwendet wird, um Überanpassung (Overfitting) zu reduzieren.

Typischerweise wird die Dropout-Schicht nach den Convolutional-Schichten und vor Vollvernetzte (neuronale) Schichten eingefügt.

Überanpassung tritt auf, wenn ein Modell zu stark auf die Trainingsdaten überlernt und Schwierigkeiten hat, gute Vorhersagen auf neuen, unbekannten Daten zu treffen. Dies kann zu einer geringen Allgemeinheit und schlechter Leistung des Modells führen.

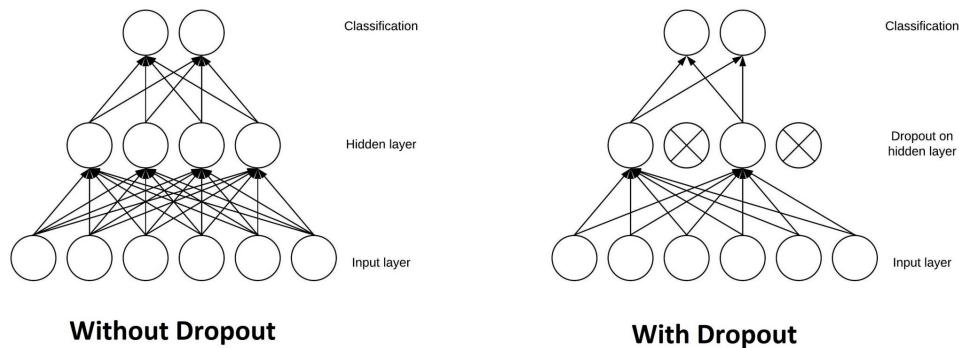


Abbildung 2.7: Dropout-Technik [7]

Die Dropout-Schicht in 2.7 beugt Überanpassung vor, indem sie während des Trainings zufällig eine bestimmte Anzahl von Neuronen in der vorhergehenden Schicht deaktiviert oder auslöscht. Dies bedeutet, dass während jedes Trainingsdurchlaufs ein Teil der Neuronen ignoriert wird und somit verschiedene Merkmale im Modell stärker berücksichtigt werden. Durch das Auslassen von Neuronen werden die Gewichte der verbleibenden Neuronen angepasst, um robustere Merkmale zu erfassen und die Abhängigkeit von bestimmten Merkmalen zu verringern.

Die Flatten-Schicht sollte in einem CNN-Modell normalerweise unmittelbar am Ende des Verlustbehaftete Schichten hinzugefügt werden.

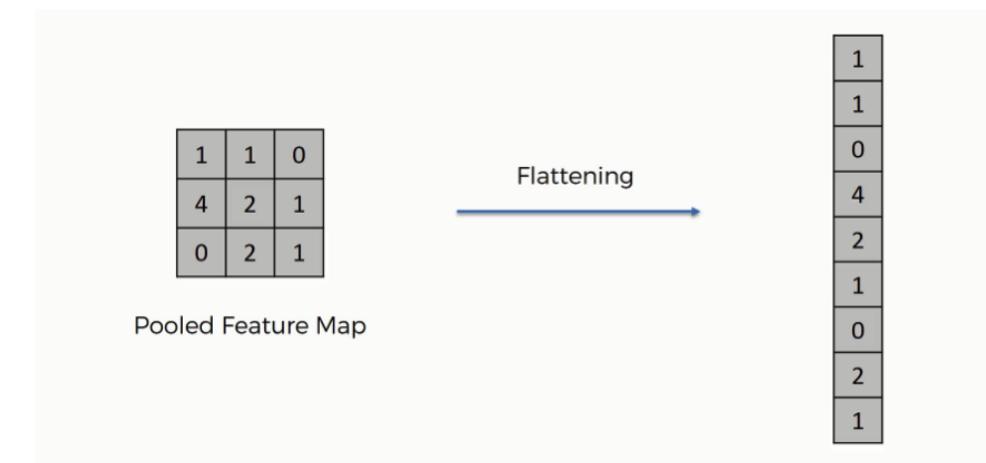


Abbildung 2.8: Flatten-Technik [8]

Die Flatten-Schicht in 2.8 wird verwendet, um diese mehrdimensionalen Merkmale in einen eindimensionalen Vektor umzuwandeln. Dieser Vektor wird dann den vollvernetzten (neuronale) Schichten übergeben, die normalerweise am Ende des Modells stehen.

2.2.5 Die Wiederholung von Convolutional Layers und Pooling Layers in einem CNN

Das Ziel der Wiederholung von Convolutional Layers und Pooling Layers in einem Convolutional Neural Network (CNN) bei der Bildklassifikation wie in 2.9 besteht darin, die hierarchische Darstellung von Merkmalen in den Eingangsbildern zu ermöglichen.

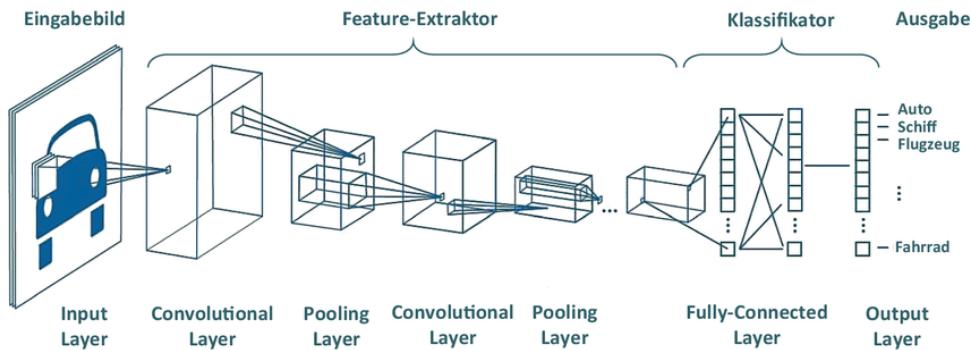


Abbildung 2.9: Die Wiederholung von Convolutional Layers und Pooling Layers in einem CNN [4]

Die wiederholte Anwendung von Convolutional Layers und Pooling Layers in einem CNN ermöglicht es dem Netzwerk, sowohl lokale als auch globale Merkmale zu erfassen und hierarchisch abstraktere Repräsentationen des Eingangsbildes zu erzeugen. Dies ist entscheidend für die erfolgreiche Bildklassifikation, da komplexe Muster und Strukturen in den Bildern erkannt werden können.

2.3 Vollvernetzte (neuronale) Schichten

Als nächstes kommt dann die Fully-connected Layer oder die vollvernetzte (neuronale) Schicht als die letzte Verarbeitungsschicht in der Aufbau einer CNN.

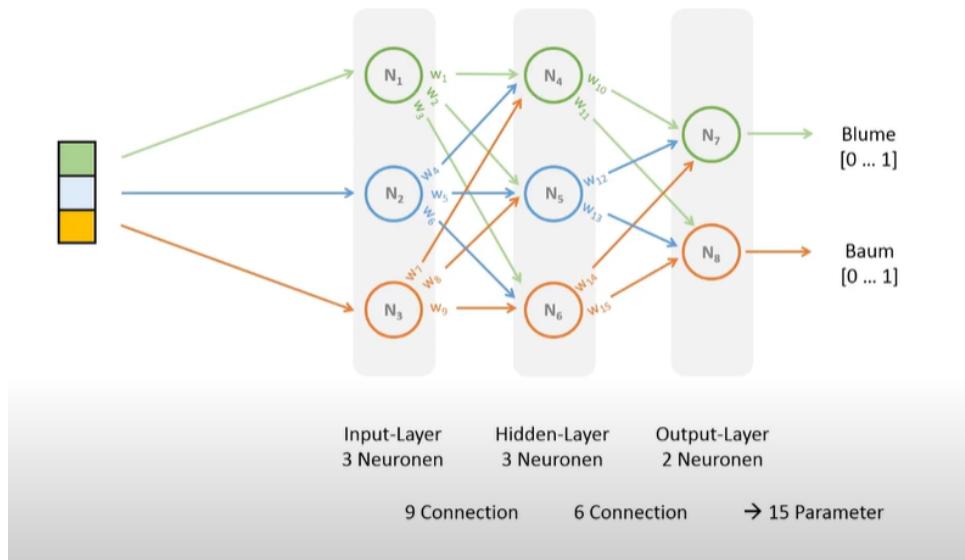


Abbildung 2.10: Einfach vollvernetzte (neuronale) Schichten [9]

Die Neuronen im Input-Layer N_1 , N_2 und N_3 in der Vollvernetzte (neuronale) 2.10 bekommen dann als Eingabe die Ausgabedaten, die vorher in der Verlustbehaftete Schichten generiert wurden sind.

Jedes Neuron des Input-Layers N_1 , N_2 und N_3 ist mit den Neuronen der nächsten Ebene(Hidden-Layer) N_4 , N_5 und N_6 verknüpft. Diese sind auch in Verbindung mit den Neuronen N_7 und N_8 im Output-Layer.

Im obigen Beispiel bekommt das Neuron N_1 im Input-Layer die Ausgabewert aus dem verlustbehafteten Schichten. Diese Wert wird dann mit dem Gewichtungsfaktor W_1 multipliziert und zunächst in den weiteren Neuronen N_4 , N_5 und N_6 in Hidden-Layers weitergeleitet. Insgesamt sind dann 9 Gewichtungsfaktoren aus den Neuronen im Input-Layer. Das Hidden-Layer hat dann als Eingabedaten die Ausgabewerte aus dem Input-Layer. Diese Werte werden mit dem Gewichtungsfaktoren im Hidden-Layer multipliziert und werden in den Output-Layer geliefert. Insgesamt sind dann 6 Gewichtungsfaktoren aus den Neuronen im Hidden-Layer. Am Ende liefert das Modell aus dem Output-Layer eine binäre Vorhersage zum Beispiel entweder ein 0 als Baum oder ein 1 als Blume.

2.3.1 Aktivierungsfunktionen in CNN

Die Aktivierungsfunktion ist ein wichtiger Bestandteil von CNN in Deep Learning, insbesondere bei der Bildklassifikation. Sie wird auf die Ausgabewerte der Neuronen in den Schichten des Netzwerks angewendet, um nicht-linearität einzuführen und komplexe Muster in den Daten zu erfassen.

Das Neuron in einer CNN bekommt die Inputdaten (Zahlenwerte) aus den Verknüpfungen und darin werden Zahlenwerte aufsummiert, aber die Summe wird dann nicht eins zu eins in den weiteren Neuronen in den nächsten Layers geleitet, sondern wird die Summe über eine bestimmte Schwelle weitergegeben.

Eine Aktivierungsfunktion bestimmt, ob ein bestimmtes Neuron über die Informationen verfügt, die zu einer korrekten Vorhersage auf der Ausgabeebene führen. Aktivierungsfunktionen ähneln der Art und Weise, wie Neuronen im Gehirn kommunizieren und Informationen übertragen, indem sie feuern, wenn die Aktivierung einen bestimmten Schwellenwert überschreitet.

In CNN für Bildklassifikation gibt es verschiedene Hauptaktivierungsfunktionen, die verwendet werden. Hier sind einige der gängigsten:

- **Rectified Linear Unit (ReLU)**: Die ReLU-Funktion ist eine nichtlineare Aktivierungsfunktion, die am häufigsten in CNNs verwendet wird. Sie berechnet den Ausgabewert als Null für negative Eingaben und gibt die gleiche positive Eingabe zurück. ReLU wird in Hidden-Layers und direkt nach dem Faltungsschicht und vor dem Pooling-Layer verwendet.

Hier in der Abbildung 2.11 ist x der Eingabewert und $f(x)$ der Ausgabewert der Funktion. Die ReLU-Funktion ist sehr einfach: Wenn der Eingabewert x positiv ist, wird er unverändert zurückgegeben, ansonsten wird er zu Null gesetzt.

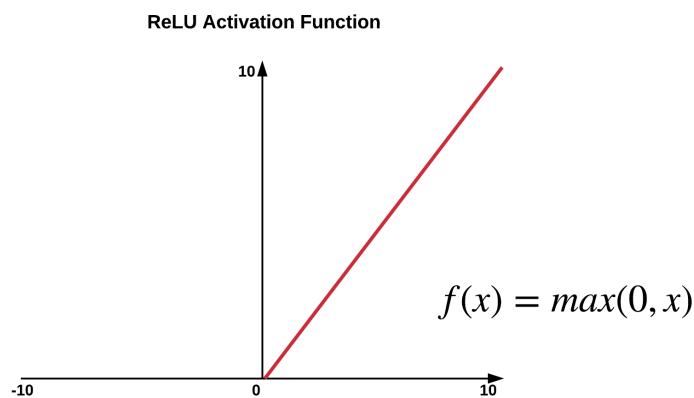


Abbildung 2.11: ReLU (Rectified Linear Unit) [10]

- **Sigmoid-Funktion**: Die Sigmoid-Funktion ist eine mathematische Funktion, die den Wertebereich zwischen 0 und 1 einschränkt. Sie wird oft als Aktivierungsfunktion in den letzten Schichten von binären Klassifikationsaufgaben verwendet.

In Abbildung 2.12 nimmt die Funktion einen Eingangswert x entgegen und gibt eine Ausgabe im Intervall $(0, 1]$ zurück

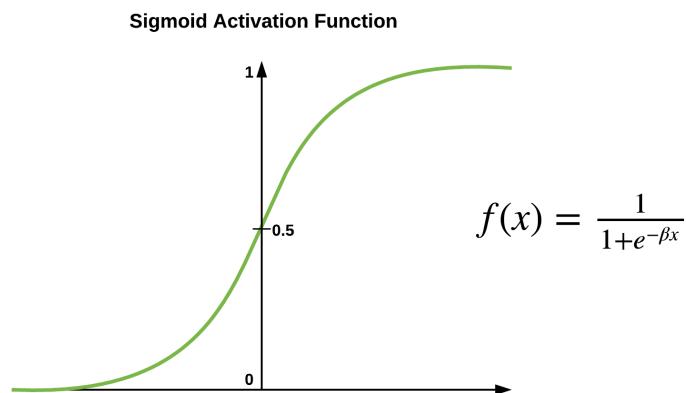


Abbildung 2.12: Sigmoid-Funktion [10]

- **Softmax-Funktion:** Die Softmax-Funktion ist eine Aktivierungsfunktion, die oft in der Ausgabeschicht(Output-Layer) von Convolutional Neural Networks (CNNs), für die Klassifikation von Bildern verwendet wird. Sie wird verwendet, um eine Wahrscheinlichkeitsverteilung über verschiedene Klassen(wie in 2.13 mit 3-Klassen) zu erzeugen.

Angenommen in der Abbildung 2.13, der Wert von Z_{21} , Z_{22} und Z_{23} beträgt 2,33, -1,46 bzw. 0,56. Nun wird die Softmax-Aktivierungsfunktion auf jedes dieser Neuronen angewendet und die folgenden Werte in 2.14 generiert.

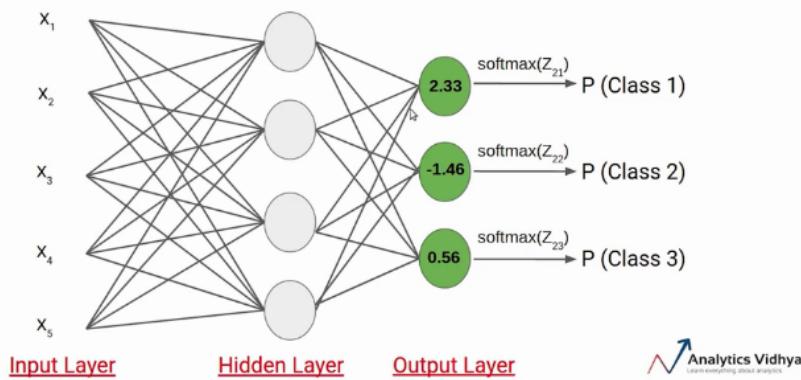


Abbildung 2.13: Softmax-Funktion [11]

Example :

$$2.33 \rightarrow P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

$$-1.46 \rightarrow P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \rightarrow P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

Abbildung 2.14: Softmax-Funktion Mathe-Function [11]

In der Abbildung 2.14 nimmt die Softmax-Funktion einen Vektor von Eingabewerten und führt eine exponentielle Normalisierung durch. Sie berechnet die exponentielle Funktion für jeden Eingabewert, dividiert diesen Wert durch die Summe aller exponentiell transformierten Eingabewerte und liefert schließlich einen Vektor von Wahrscheinlichkeitswerten zurück.

Dies sind die Wahrscheinlichkeitswerte in 2.14, dass ein Datenpunkt zu den jeweiligen Klassen gehört. Zu Beachten, dass die Summe der Wahrscheinlichkeiten in diesem Fall gleich 1 ist.

In diesem Fall ist klar, dass die Eingabe zur Klasse 1 in 2.14 gehört. Wenn sich also die Wahrscheinlichkeit einer dieser Klassen ändert, würde sich auch der Wahrscheinlichkeitswert der ersten Klasse ändern.

3 Datensatz von Hautkrebsbildern

3.1 Entstehung des Hautkrebses

Hautkrebs ist eine Art von Krebs, der in der Haut beginnt und durch das unkontrollierte Wachstum abnormer Zellen in der Epidermis, der äußersten Hautschicht, verursacht wird. Dies geschieht aufgrund von nicht reparierten DNA-Schäden, die Mutationen auslösen. Diese Mutationen führen dazu, dass sich die Hautzellen schnell vermehren und bösartige Tumore bilden.

Der Haut besteht aus mehreren Schichten, darunter die äußere Oberhaut, Bindegewebe und darunterliegende Fettschicht. Die Melanozyten (Pigmentzellen) befinden sich in der Oberhaut und produzieren Melanin (Pigment), welches in der Oberhaut gleichmäßig verteilt und die Aufgabe hat, unsere Körpern vor schädigen UV-Strahlung zu schützen. Melanozyten können gutartige Hautflecken (Hautveränderungen) und bösartige Tumoren bilden.

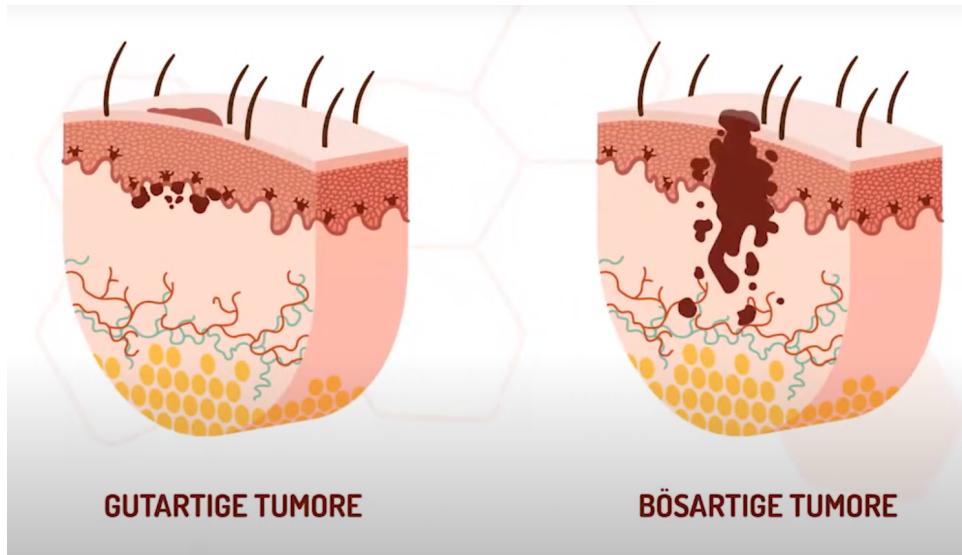


Abbildung 3.1: Gutartige und Bösartige Tumoren [15]

Gutartige Hautflecke (Hautveränderungen) werden als Muttermale oder Leberflecken bezeichnet und sie sind in der Dermatologie keine Hautkrebsarten haben verschiedene Arten:

- Benign keratosis (solar lentigo / seborrheic keratosis / lichen planus-like keratosis)
- Melanocytic nevus
- Dermatofibroma
- Vascular lesion

Bösartige Tumoren wird als Melanom bezeichnet und haben verschiedene Arten:

- Melanoma
- Basal cell carcinoma
- Actinic keratosis
- Squamous cell carcinoma

Die genauen Ursachen für Hautkrebs sind noch nicht vollständig verstanden, aber es gibt bestimmte Risikofaktoren, die das Risiko erhöhen können, an Hautkrebs zu erkranken:

- Der Hauptgrund für Hautkrebs ist langfristige und wiederholte Exposition gegenüber ultravioletter (UV) Strahlung, insbesondere von der Sonne. UV-Strahlung kann Veränderungen im Erbgut der Hautzellen verursachen, was zu unkontrolliertem Zellwachstum führt und letztendlich Krebs verursachen kann.
- Helle Haut: Menschen mit heller Haut haben weniger Melanin, das die Haut vor schädlicher UV-Strahlung schützt. Daher haben sie ein höheres Risiko, Hautkrebs zu entwickeln.
- Sonnenbrandgeschichte: Menschen, die in der Vergangenheit schwere Sonnenbrände erlitten haben, haben ein höheres Risiko, Hautkrebs zu entwickeln.
- Familiäre Vorgeschichte: Eine familiäre Vorgeschichte von Hautkrebs kann das Risiko erhöhen, selbst Hautkrebs zu bekommen.
- Immunsuppression: Menschen mit einem geschwächten Immunsystem, z. B. aufgrund einer Organtransplantation oder HIV-Infektion, haben ein erhöhtes Risiko für Hautkrebs.

3.2 Hautkrebserkennung und Bilderquellen

Mithilfe von Deep Learning-Algorithmen kann eine Künstliche Intelligenz (KI) auf Basis großer und vertrauenswürdiger medizinischer Bild-Datensätzen Hautkrebsläsionen erkennen und klassifizieren. Die KI erlernt dabei die charakteristischen Merkmale von gesunden und kranken Hautzellen, wodurch sie frühzeitig erkennen kann.

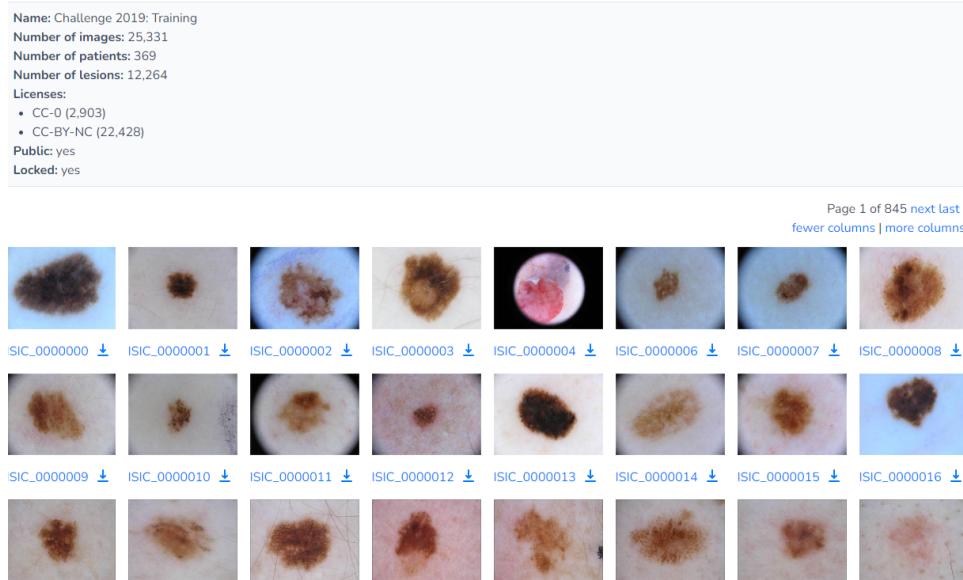


Abbildung 3.2: The International Skin Imaging Collaboration Webseite [12]

In diesem Projektarbeit werden es dann in den nächsten Kapitel 2 CNN-Modellen für die Hautkrebsklassifikation implementiert:

- Das erste CNN-Modell ist eine **binäre Bilderklassifikation**, das eine Vorhersage entweder gutartiger Hautfleck (Hautveränderung) oder bösartige Tumor(Cancer) liefert und das CNN-Modell dann mit Tensorflow und Keras aufgebaut wurde.
- Das zweite Modell ist eine **multiklassen Bildklassifikation** und wurde so programmiert, dass es eine Vorhersage von den 4 verschiedenen Hautkrebsarten oder von den 4 Verschiedenen gutartige Hautflecken in 3.1 in dem ISIC-Datensatz in Kaggle liefert.

In **Kaggel [13]** wurde ISIC - 2019-Datensatz gefunden, wobei die Quelle von diesen Bildern von **The International Skin Imaging Collaboration (ISIC) [12]** in 3.2 stammen. Der Datensatz enthält mehr als 25.000 Hautveränderungsbildern von 369 Patienten, die in 8 Kategorien, wobei 4 davon gutartige Hautveränderungsarten und den 4 restliche Klassen Hautkrebsarten sind, klassifiziert sind.

Im **ISIC-Datensatz** befinden sich **4 gutartige Hautveränderungsarten** von Muttermale oder Leberflecken, darunter Benign keratosis, Melanocytic nevus, Dermatofibroma und Vascular lesion, und **4 Hautkrebsarten**, darunter Melanoma, Basal cell carcinoma, Squamous cell carcinoma und Actinic keratosis. Es fehlt die **zusätzliche Klasse**, die die **Information** enthält, ob ein Bild kein **Hautbild** ist oder keine **Hautläsionen** zeigt, wenn die Benutzer ein falsches Bild hochladen oder ein Bild, die keine **Hautläsionen** enthält, deswegen wird die **Vorhersage-Ergebnis** von der hochgeladenen **Hautbild** nur auf den **8 entsprechenden Klassen** eingeschränkt.

Hier in den folgenden Abbildungen sind Beispielbildern von den 4 gutartigen Hautflecken (Hautveränderungen) und 4 bösartigen Hautkrebsarten, die am meistens bei der Diagnose von Hautkrebs auftreten können und sich in ISIC - 2019-Datensatz befinden:



Abbildung 3.3: Melanocytic nevus

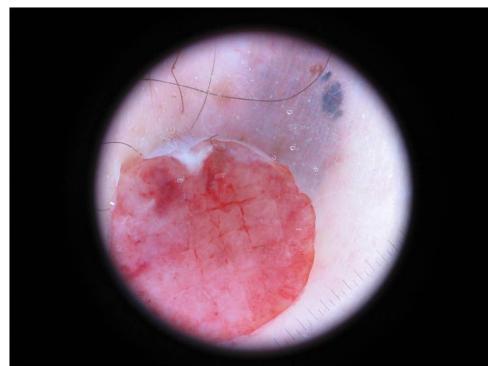


Abbildung 3.4: Melanoma



Abbildung 3.5: Squamous cell carcinoma



Abbildung 3.6: Vascular lesion



Abbildung 3.7: Dermatofibroma



Abbildung 3.8: Benign keratosis

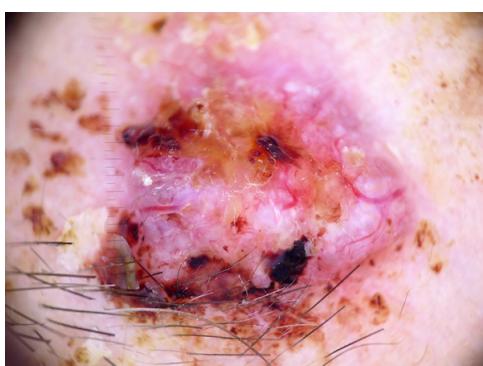


Abbildung 3.9: Basal cell carcinoma



Abbildung 3.10: Actinic keratoses

4 Implementierung von CNNs in Deep Learning-Bilderklassifikation

4.1 Programmierwerkzeuge

Es gibt verschiedene Programmierwerkzeuge, die für die Implementierung und das Training von CNNs in Deep Learning verwendet werden können.

TensorFlow und Keras sind die weit verbreitete Python-Bibliotheken, die Entwicklern umfangreiche Funktionalitäten für die Implementierung von CNNs in Deep Learning-Bilderklassifikation bieten. Dabei übernehmen sie Aufgaben wie das Erstellen von Netzwerkarchitekturen, das Laden und Verarbeiten von Bildern und das Training von CNNs.

4.1.1 TensorFlow

TensorFlow 4.1 ist ein Open-Source-Framework für maschinelles Lernen, das von Google entwickelt wurde. Es ist eine leistungsstarke Plattform, die die Implementierung und Berechnung von neuronalen Netzwerken erleichtert. TensorFlow wurde speziell für die effiziente Ausführung von komplexen Berechnungen auf unterschiedlichen Hardwareplattformen entwickelt.

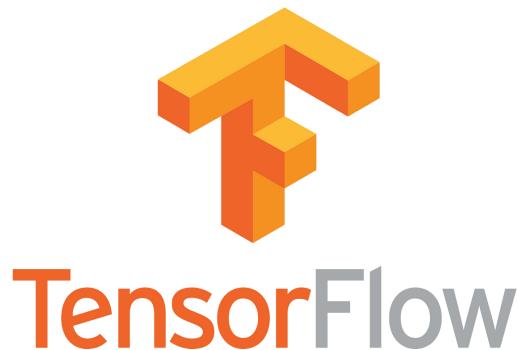


Abbildung 4.1: TensorFlow

TensorFlow bietet umfangreiche Unterstützung für den Aufbau, das Training und die Auswertung von CNNs. Es stellt spezielle Operationen und Funktionen bereit, die für CNNs optimiert sind, wie zum Beispiel Faltungsschichten, Pooling-Operationen und Aktivierungsfunktionen. Darüber hinaus bietet TensorFlow auch Tools und Bibliotheken für die Datenverarbeitung, Visualisierung und Modellbewertung, die bei der Arbeit mit CNNs in der Bilderklassifikation unterstützen.

4.1.2 Keras

Keras 4.2 ist eine beliebte Deep-Learning-Bibliothek, die häufig zur Implementierung von Convolutional Neural Networks (CNNs) in der Bilderklassifikation verwendet wird. In TensorFlow ist Keras direkt integriert, was bedeutet, dass man Keras-Modelle direkt in TensorFlow erstellen kann, um von beiden Bibliotheken zu profitieren.

In Keras können CNNs relativ einfach erstellt und trainiert werden. Es bietet eine benutzerfreundliche API, die es Entwicklern ermöglicht, Schichten des CNNs wie Convolutional Layers, Pooling Layers und Vollvernetzte (neuronale) Schichten intuitiv zu konfigurieren.



Abbildung 4.2: Keras

Keras bietet auch verschiedene vortrainierte Modelle, die auf großen Bilderklassifikationsdatensätzen wie ImageNet trainiert wurden. Diese vortrainierten Modelle, wie z.B. EfficientNet, VGG16 oder ResNet, können als Ausgangspunkt verwendet werden, um auf spezifische Aufgaben zugeschnittene CNNs zu erstellen.

4.2 Binäre Hautkrebs-Bilderklassifikation

Das Ziel von der Erstellung eines CNN-Modell für die binäre Hautkrebs-Bilderklassifikation besteht darin, dass man experimentell die CNN-Schichten aufbauen kann und um die Parametern in den Schichten wie die Anzahl der Filtern-Matrizen, Optimizerwerte und die Anzahl der Hidden-Layers in der vollvernetzten Schichten so passend wie möglich einzustellen.

Zuerst wurde eine Menge von der medizinischen gutartigen Hautflecken (Hautveränderungen) und bösartigen Hautkrebs-Bildern von **The International Skin Imaging Collaboration (ISIC)** heruntergeladen.

Dann wurde ein Datei namens **data** erstellt, in dem 2 Dateiordnern enthält. Diese wurden so genannt, dass jedes Dateiordner der Name der zwei Hautveränderungsarten-Klassen [Benigne Hautfleck , Bösartiger Hautkrebs] repräsentiert.

In jedem Dateiordner wurden dann die entsprechend heruntergeladenen Bildern gespeichert.

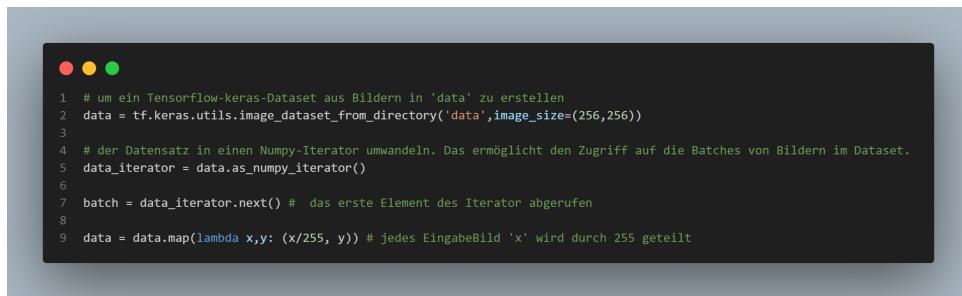
4.2.1 CNN-Modell für die binäre Hautbilderklassifikation

In den folgenden Code wird das CNN-Algorithmus in meinem eigenen PC aufgebaut, wobei das Jupyter-Notebook in Anaconda für die Codeumgebung ausgewählt wurde. **Jupyter-Notebook** ist eine interaktive Entwicklungsumgebung, die es ermöglicht, Code in einem Webbrowser auszuführen. Es unterstützt verschiedene Programmiersprachen, darunter Python, und organisiert den Code in Zellen, die einzeln ausgeführt werden können.

Anaconda ist eine Open-Source-Distribution für die Programmiersprache Python, die speziell für Data Science und Machine Learning entwickelt wurde. Es enthält eine Vielzahl von Bibliotheken, Pakete, Werkzeuge, die für Deep Learning erforderlich sind, wie z.B. TensorFlow und Keras, und einschließlich Jupyter-Notebook. Diese sollten zuerst in der virtuellen Umgebung in Anaconda-App installiert und dann kann man mit der Schreiben des Codes starten.

In diesem Github-repository 4.3 [16] befindet sich das Modell Schritt für Schritt mit den Kommentaren:

- Im Code 4.3 wurde in der Zeile 2 vor dem Beginn mit der Erstellung der CNN-Algorithmus ein KerasAPI-Datensatz aus den Bildern in der Verzeichnis **data**, in der die gelabelte Modell-Klassen sich befinden, generiert, wobei das Bildergöße bei (256x256) Pixel liegt.



```
1 # um ein Tensorflow-keras-Dataset aus Bildern in 'data' zu erstellen
2 data = tf.keras.utils.image_dataset_from_directory('data',image_size=(256,256))
3
4 # der Datensatz in einen Numpy-Iterator umwandeln. Das ermöglicht den Zugriff auf die Batches von Bildern im Dataset.
5 data_iterator = data.as_numpy_iterator()
6
7 batch = data_iterator.next() # das erste Element des Iterator abgerufen
8
9 data = data.map(lambda x,y: (x/255, y)) # jedes EingabeBild 'x' wird durch 255 geteilt
```

Abbildung 4.3: Dataset KerasAPI

Der Erstellung von der **numpy-Iterators für den Datensatz** in der Zeile 5 in 4.3 ermöglicht uns, durch den Datensatz zu iterieren und die Daten in Form von Numpy-Arrays zu erhalten. Diese ist nützlich, um die Bilddaten in einem bestimmten Format für die nächsten Verarbeitungsschichten in CNN zu umwandeln.

Danach wurde in der Zeile 9 in 4.3 die Normalisierung der Bilddaten durchgeführt, wobei das Eingabebilddata **X** durch 255 geteilt wurde, um sicherzustellen, dass die Pixelwerte im Bereich von 0 bis 1 liegen, was das Training von neuronalen Netzen erleichtern kann.

Hier in 4.4 wird von einem Batch die ersten 4 Bildern von Datensatz dargestellt, wobei im oberen Bereich des Bildes die (Labels) ein 0 als gutartiger Hautfleck (Hautveränderung) oder 1 als bösartiger Hautkrebs von der 4 Batch-Bildern angezeigt wird.

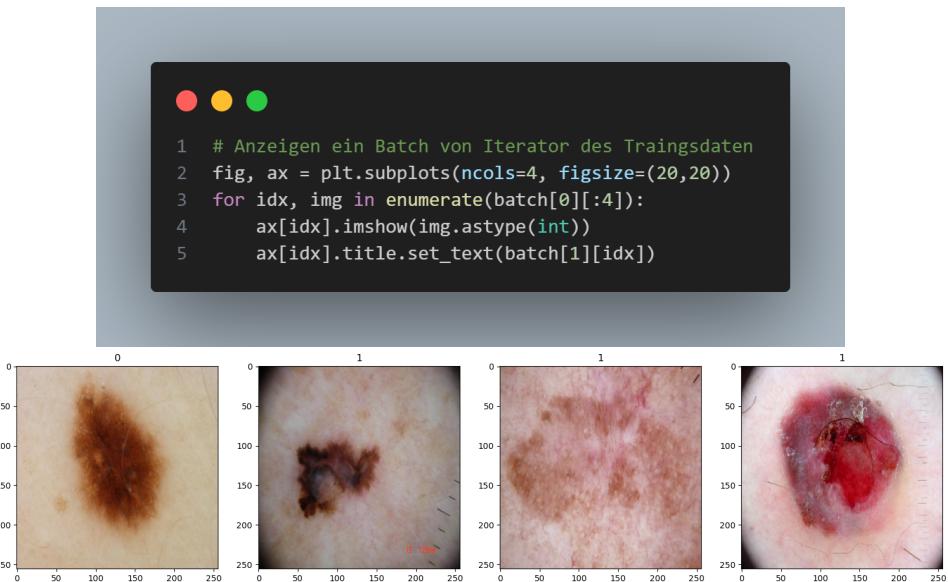
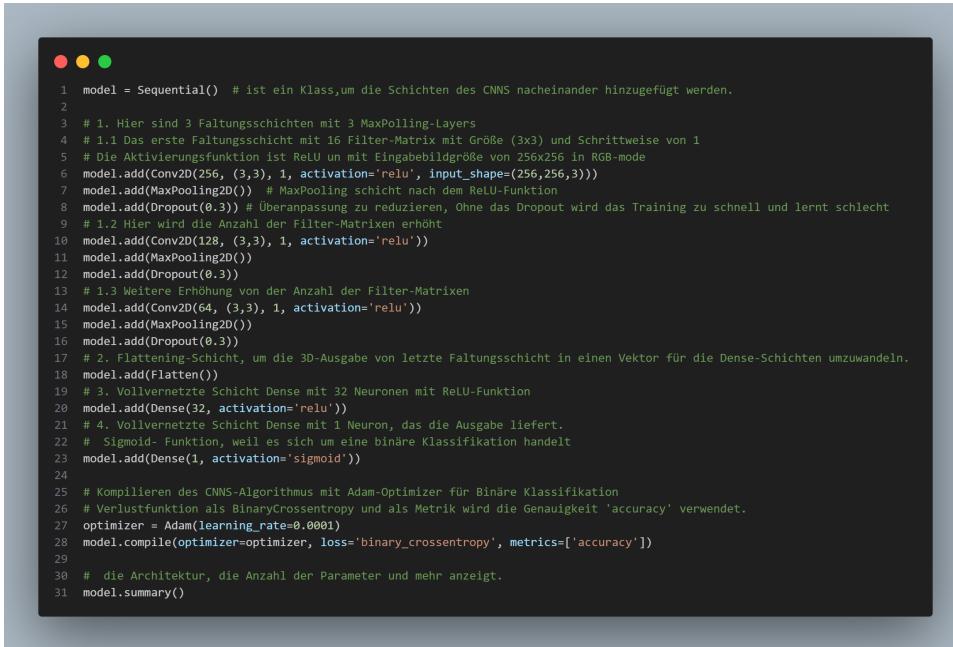


Abbildung 4.4: 4 Batch-Bildern von Datensatz

- In den folgenden Code 4.5 wurde ein CNN-Algorithmus für die binäre Hautkrebsklassifikation aufgebaut.



```

1 model = Sequential() # ist ein Klass,um die Schichten des CNNs nacheinander hinzugefügt werden.
2
3 # 1. Hier sind 3 Faltungsschichten mit 3 MaxPooling-Layers
4 # 1.1 Das erste Faltungsschicht mit 16 Filter-Matrix mit Größe (3x3) und Schrittweise von 1
5 # Die Aktivierungsfunktion ist ReLU um mit Eingabebildgröße von 256x256 in RGB-mode
6 model.add(Conv2D(256, (3,3), 1, activation='relu', input_shape=(256,256,3)))
7 model.add(MaxPooling2D()) # MaxPooling schicht nach dem ReLU-Funktion
8 model.add(Dropout(0.3)) # Überanpassung zu reduzieren, Ohne das Dropout wird das Training zu schnell und lernt schlecht
9 # 1.2 Hier wird die Anzahl der Filter-Matrizen erhöht
10 model.add(Conv2D(128, (3,3), 1, activation='relu'))
11 model.add(MaxPooling2D())
12 model.add(Dropout(0.3))
13 # 1.3 Weitere Erhöhung von der Anzahl der Filter-Matrizen
14 model.add(Conv2D(64, (3,3), 1, activation='relu'))
15 model.add(MaxPooling2D())
16 model.add(Dropout(0.3))
17 # 2. Flattening-Schicht, um die 3D-Ausgabe von letzte Faltungsschicht in einen Vektor für die Dense-Schichten umzuwandeln.
18 model.add(Flatten())
19 # 3. Vollvernetzte Schicht Dense mit 32 Neuronen mit ReLU-Funktion
20 model.add(Dense(32, activation='relu'))
21 # 4. Vollvernetzte Schicht Dense mit 1 Neuron, das die Ausgabe liefert.
22 # Sigmoid- Funktion, weil es sich um eine binäre Klassifikation handelt
23 model.add(Dense(1, activation='sigmoid'))
24
25 # Kompilieren des CNNs-Algorithmus mit Adam-Optimizer für Binäre Klassifikation
26 # Verlustfunktion als BinaryCrossentropy und als Metrik wird die Genauigkeit 'accuracy' verwendet.
27 optimizer = Adam(learning_rate=0.0001)
28 model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
29
30 # die Architektur, die Anzahl der Parameter und mehr anzeigen.
31 model.summary()

```

Abbildung 4.5: CNN-Algorithmus für die binäre Hautbildderklassifikation

Zuerst wurde in der Zeile 1 die Klasse **Sequential()** von Keras abgerufen , um die Schichten nacheinander in der Reihenfolge hinzufügen.

In den nächsten Zeilen in 4.5 wurde 3 Faltungsschichten **Conv2D**-Schichten nacheinander hinzugefügt, um die Faltungsoperationen auf den Eingangsbildern durchzuführen. Diese Faltungsschichten **Convolutional Layer** variieren mit ihre Parametern, wobei die Anzahl der Filtern-Matrizen in der erste Schicht, die als Eingangsschicht für die Hautbildern im Datensatz gilt, 256 liegt, Kernelgröße von (3x3), eine Schrittweite 1 und eine ReLU- Aktivierungsfunktion hat.

Jedes **Convolutional Layer** wird mit einem Max Pooling Layer, die die räumliche Dimension der vorherigen Schicht reduziert, und einem Dropout Layer mit einer Dropout-Rate von 0.3 gefolgt.

Das Ziel von den obigen Faltungsschichten besteht darin, dass man durch diese **Convolutional Layers** aus den Hautkrebsbildern einen größeren Anzahl von den Merkmalen (Feature Map) gewinnen kann.

In der Zeile 18 in 4.5 wurde dann die Flatten-Schicht hinzugefügt, um die Ausgabe der vorherigen Schicht in einen eindimensionalen Vektor umzuwandeln, damit sie von den Dense-Layers verarbeitet werden kann.

Am Ende des CNNs wurden 2 **Dense-Schicht** in der Zeilen 20 und 23 in 4.5 als **Fully-connected Layer** oder **die vollvernetzte (neuronale) Schicht** hinzugefügt.

Bei einer Dense-Schicht ist jeder Ausgang jedes Neurons mit jedem Eingangsneuron verbunden. Diese Verbindungen haben Gewichte, die während des Trainings angepasst werden, um das Modell zu optimieren.

Die Dense-Schichten spielen eine wichtige Rolle bei der Zusammenführung der im CNN gelernten Merkmale, um eine endgültige Vorhersage zu treffen. Sie nehmen die Merkmale, die von den vorherigen Convolutional-Schichten extrahiert wurden, und lernen, wie sie zu einer bestimmten Vorhersage kombiniert werden können.

Bei der ersten Schicht **Dense(32, activation='relu')** wird eine Dense-Schicht mit 32 Neuronen und ReLU-Aktivierungsfunktion hinzugefügt. Diese Schicht dient dazu, die abstrakten Merkmale zu verarbeiten und zu komprimieren.

Dense(1, activation='sigmoid') ist die Ausgabeschicht, die eine einzige Neuron mit einer Sigmoid-Aktivierungsfunktion enthält. Sigmoid- Funktion wurde hier festgelegt, weil es sich um eine binäre Klassifikation [Gutartiger Hautfleck , Bösartiger Hautkrebs] handelt.

Um während des Trainingsprozess die Gewichtungsfaktoren des CNN in jedem Schicht anzupassen, wird in der Zeile 27 in 4.5 ein Optimizer von Tensorflow-Keras abgerufen. Der **Adam-Optimizer** ist eine beliebte Optimierungsalgorithmus für die neuronale Netzwerke. Die Lernrate wurde **learning_rate=0.0001** festgestellt, die angibt, wie groß die Schritte bei der Aktualisierung der Gewichtungsfaktoren sein sollen.

In der Zeile 28 in 4.5 wird das Modell kompiliert. Das bedeutet, dass das Modell mit den obigen spezifizierten Konfigurationen in CNN-Schichten für das Trainingsprozess vorbereitet wird.

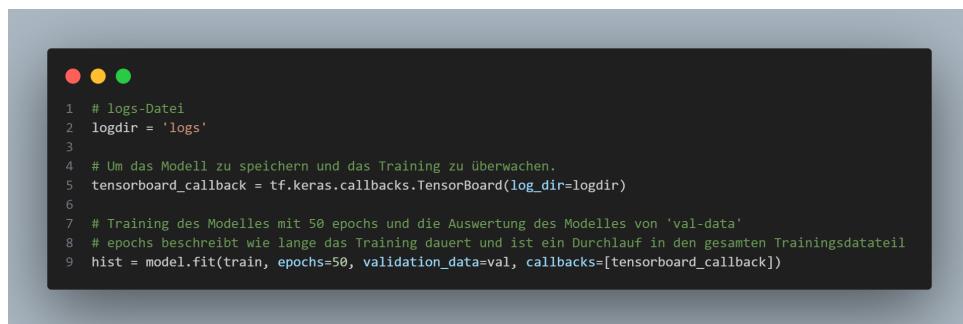
loss='binary_crossentropy' gibt im Verlauf des Trainingsprozesses die Verlustfunktion an, die abnehmen werden soll. Für binäre Klassifikationsaufgaben wird die Funktion '**binary_crossentropy**' verwendet, die den Unterschied zwischen den tatsächlichen und vorhergesagten Werten misst.

Die Funktion **metrics=['accuracy']** gibt die Genauigkeit (accuracy) als Metrik an und wird verwendet, um die Leistung des Modells während des Trainings zu überwachen. Die Genauigkeit gibt an, wie gut das Modell in der Lage ist, die richtigen Vorhersagen zu treffen.

model.summary(): gibt eine Zusammenfassung des Modells aus, einschließlich der Anzahl der Parameter in jeder Schicht und der Gesamtanzahl der Parameter im Modell. Dies ist hilfreich, um einen Überblick über die Architektur und Größe des Modells zu erhalten.

4.2.2 Training des Modells

Der Trainingsprozess im CNN-Modell zielt darauf ab, die Verbindungen zwischen den Neuronen im Netzwerk so anzupassen, dass es in der Lage ist, die komplexe Struktur der Trainingsdaten zu erlernen und genaue Vorhersagen zu machen.



```

● ● ●
1 # logs-Datei
2 logdir = 'logs'
3
4 # Um das Modell zu speichern und das Training zu überwachen.
5 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
6
7 # Training des Modells mit 50 epochs und die Auswertung des Modells von 'val-data'
8 # epochs beschreibt wie lange das Training dauert und ist ein Durchlauf in den gesamten Trainingsdatenteil
9 hist = model.fit(train, epochs=50, validation_data=val, callbacks=[tensorboard_callback])

```

Abbildung 4.6: Training des Modells

Nachdem das Modell aufgebaut wurde, können wir dann mit dem Trainingsprozess beginnen. Dies geschieht mit dem Funktion **model.fit()** in der Zeile 9 in 4.6 mit den folgenden Parametern:

- **'train'** sind der Trainingsdatenanteil, mit den das Modell trainiert werden soll. Dieser Anteil beträgt in diesem Projekt 70%
- **'epochs=50'** beschreibt wie lange das Training dauert und ist ein Durchlauf in den gesamten Trainingsdatenanteil. Hier steht 50 Epochen zur Verfügung bei dem Trainingsprozess.
- **validation_data=val** gibt an, dass die Leistung des Modells auch auf den Validierungsdatenanteil, der in diesem Projekt 20% beträgt, überprüft werden soll.
- **'callbacks=[tensorboard_callback]'**: In der Zeile 5 in 4.6 wurde zuerst ein Tensor-Board für das Modell erstellt und in der Dateiordner **'logs'** gespeichert. Tensor-Board ist ein Visualisierungswerkzeug von TensorFlow, das dabei hilft, das Modell während des Trainingsprozess zu überwachen. In der Variable **'hist'** wird dann das Ergebnis von der Trainingsprozess gespeichert und in jedem Durchlauf eines Epochen die Trainingsinformation wie die Dauer einer Epochen, Verlust und Genauigkeit.

```

Epoch 1/50
24/24 [=====] - 153s 6s/step - loss: 0.7012 - accuracy: 0.5169 - val_loss: 0.6917 - val_accuracy: 0.5039
Epoch 2/50
24/24 [=====] - 150s 6s/step - loss: 0.6794 - accuracy: 0.5495 - val_loss: 0.6848 - val_accuracy: 0.5898
Epoch 3/50
24/24 [=====] - 143s 6s/step - loss: 0.6232 - accuracy: 0.6719 - val_loss: 0.6043 - val_accuracy: 0.7969
Epoch 4/50
24/24 [=====] - 143s 6s/step - loss: 0.5235 - accuracy: 0.7370 - val_loss: 0.5542 - val_accuracy: 0.7617
Epoch 5/50
24/24 [=====] - 144s 6s/step - loss: 0.4104 - accuracy: 0.8359 - val_loss: 0.5092 - val_accuracy: 0.7695
Epoch 6/50
24/24 [=====] - 144s 6s/step - loss: 0.3787 - accuracy: 0.8372 - val_loss: 0.4333 - val_accuracy: 0.8438
Epoch 7/50
24/24 [=====] - 144s 6s/step - loss: 0.3633 - accuracy: 0.8503 - val_loss: 0.4238 - val_accuracy: 0.8477
Epoch 8/50
24/24 [=====] - 143s 6s/step - loss: 0.3502 - accuracy: 0.8581 - val_loss: 0.4237 - val_accuracy: 0.8281
Epoch 9/50
24/24 [=====] - 143s 6s/step - loss: 0.3386 - accuracy: 0.8646 - val_loss: 0.3967 - val_accuracy: 0.8359
Epoch 10/50
24/24 [=====] - 144s 6s/step - loss: 0.3307 - accuracy: 0.8711 - val_loss: 0.3870 - val_accuracy: 0.8906

```

Abbildung 4.7: Ersten 10 Epochen bei Training des Modells

Aus der Abbildung 4.7 kann man sehen, dass das Verlustprozess der Trainingsdaten '**Loss**' abnimmt und die Genauigkeit '**accuracy**' im Verlauf des Trainingsprozess über den Epochen zunimmt, was uns interpretiert, dass das Modell eine positive Entwicklung und ein erfolgreiches Training zeigt.

Im folgenden Diagrammen in 4.8 sind die Verläufe von der Trainingsverluste, Validierungsverluste, Trainingsgenauigkeiten und Validierungsgenauigkeiten über die 50 Epochen grafisch zu sehen.

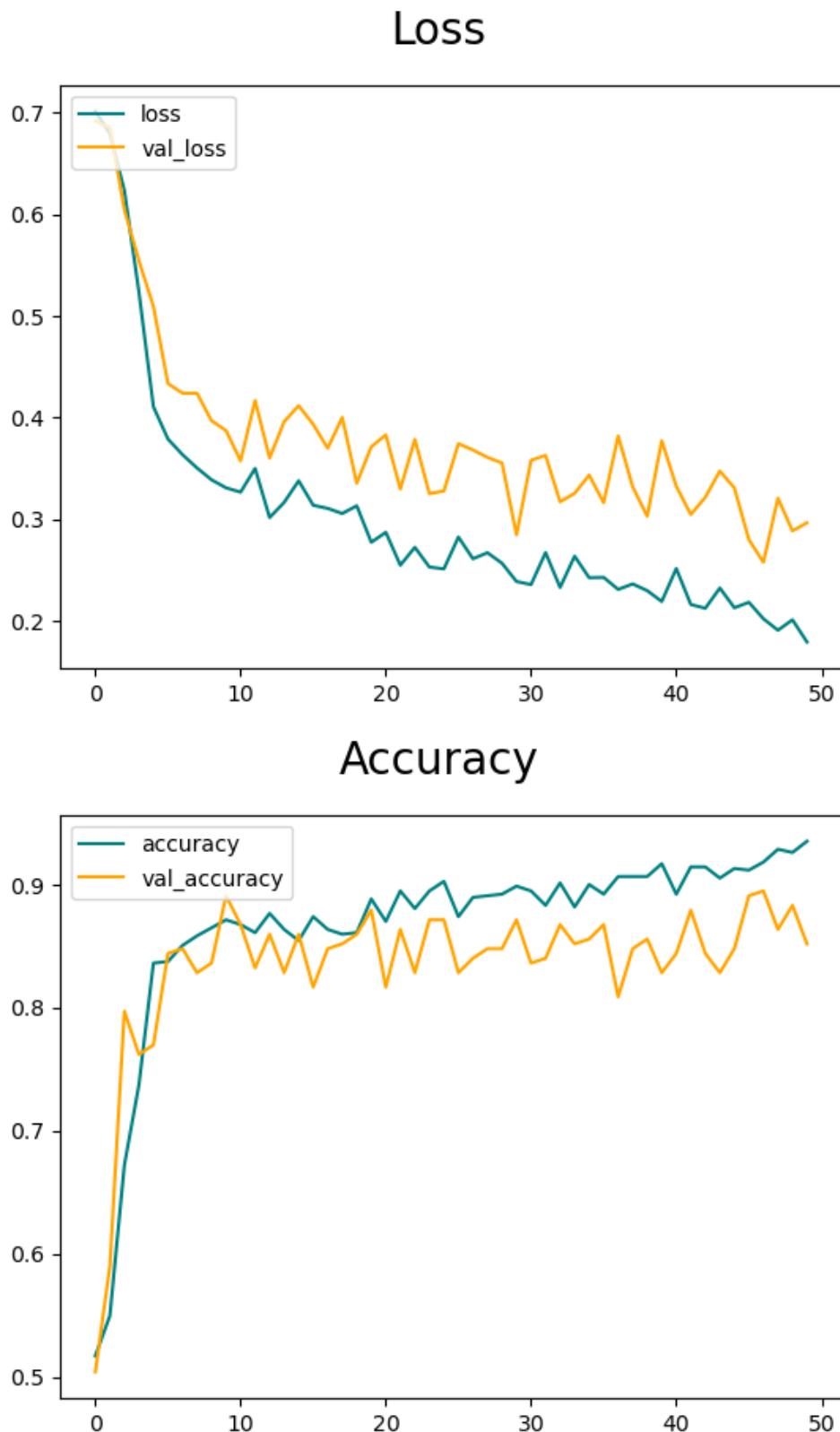
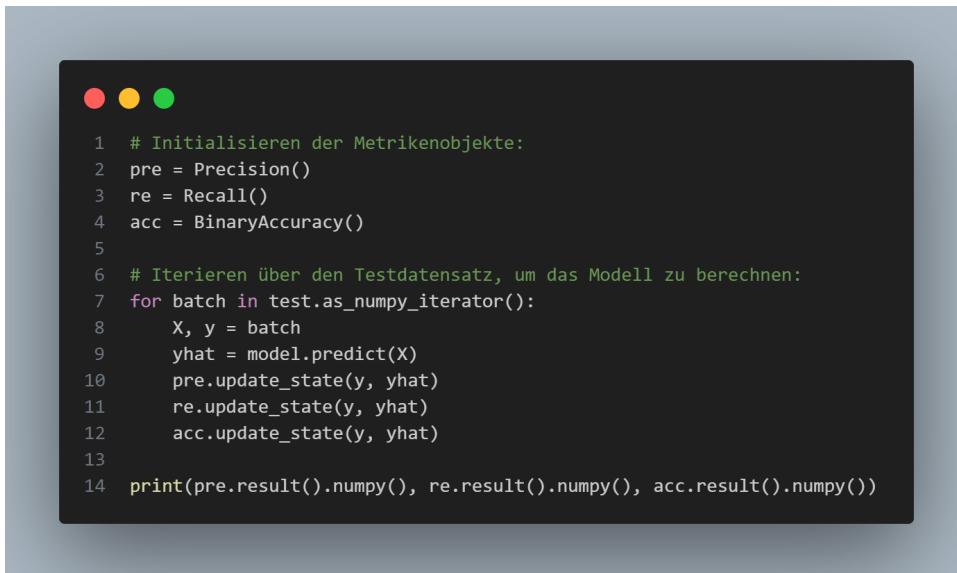


Abbildung 4.8: Loss und Accuracy bei Trainingsprozess
36

4.2.3 Evaluation des CNN-Modells

Nachdem das Modell trainiert wurde, muss dann das trainiertes Modell auf dem Testdatenanteil, welchen 10% von der gesamten Bilderdatenanteil beträgt, evaluiert werden. Das kann man erreichen unter Verwendung von **Precision**, **Recall** und **Binary Accuracy** als Metriken in der Abbildung 4.9.



```

● ● ●

1 # Initialisieren der Metrikenobjekte:
2 pre = Precision()
3 re = Recall()
4 acc = BinaryAccuracy()
5
6 # Iterieren über den Testdatensatz, um das Modell zu berechnen:
7 for batch in test.as_numpy_iterator():
8     X, y = batch
9     yhat = model.predict(X)
10    pre.update_state(y, yhat)
11    re.update_state(y, yhat)
12    acc.update_state(y, yhat)
13
14 print(pre.result().numpy(), re.result().numpy(), acc.result().numpy())

```

Abbildung 4.9: Evaluation des CNN-Modells

Durch der Iteration in Batches über den Testdatensatz in der Zeile 7 in 4.9 werden für jeden Batch die Bildereingaben **X** und die wahren Labels **y** extrahiert. Die Variable **yhat** enthält **model.predict(X)**, wodurch hier die Vorhersagen für die Bildereingabe **X** generiert werden. Die Metriken (**pre**, **re**, **acc**) werden dann mit den wahren Labels und den Vorhersagen aktualisiert.

Am Ende der Evaluationsprozess wird dann das Ergebnis der Evaluation in der Zeile 14 in 4.9 ausgegeben.

Precision: 87.09%, Recall: 87.09%, BinaryAccuracy: 88.05%

4.2.4 Testen des CNN-Modells

Ein wichtige Punkt bei der CNN-Modell in DL, dass das Bilderklassifikation-Modell durch Bilddaten von außerhalb unserem Datensatz getestet werden soll.



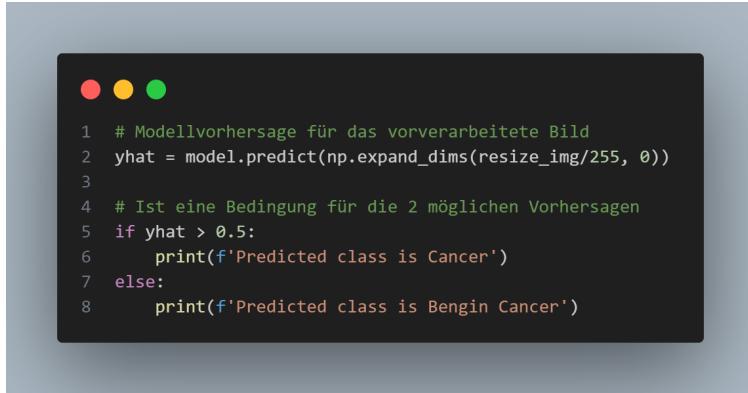
Abbildung 4.10: originales Testbild

Hier in 4.10 wird ein Bild **'test2.jpg'** durch **cv2** Python-Bibliothek eingelesen und in der Originalgröße (764, 1024) angezeigt. Dann wird das Bild in den RGB-Farbmode umgewandelt, da **cv2** die Bilder in BGR-mode anzeigt.

Zunächst wurde das Testbild in 4.11 verarbeitet und ihr Dimension auf eine Größe von (256, 256) reduziert, damit es für die Eingabe des CNN-Modells anpasst. Die reduzierte Version des Bildes wurde dann angezeigt.



Abbildung 4.11: Verarbeitetes Testbild



```

1 # Modellvorhersage für das vorverarbeitete Bild
2 yhat = model.predict(np.expand_dims(resize_img/255, 0))
3
4 # Ist eine Bedingung für die 2 möglichen Vorhersagen
5 if yhat > 0.5:
6     print(f'Predicted class is Cancer')
7 else:
8     print(f'Predicted class is Benign Cancer')

```

Abbildung 4.12: Vorhersage für das Testbild

Das reduzierte Testbild wird dann durch `resize_img/255` in 4.12 normalisiert, indem jedes Pixel durch 255 geteilt wird und der Wertebereich von 0 bis 1 nimmt.

Das CNN-Modell erwartet eine Eingabe in Form von Batch (Viele Bildern), deswegen ist es wichtig hier `np.expand_dims(resize_img/255, 0)` in 4.12 , das normalisierte Testbild um eine zusätzliche Dimension **0** zu erweitern. Das ermöglicht das Testbild als Eingabe für das Modell, in einem einzeln Batch mit einer Größe von **1** zu behandeln.

Um die Vorhersage zu erhalten, wird die Methode `model.predict()` in 4.12 aufgerufen. Diese nimmt die vorverarbeiteten Testbilddaten als Eingabe und gibt die Vorhersage `yhat` zurück.

Als Vorhersagen Für die binäre Hautkrebs-Bilderklassifikation-Modell gibt es letztendlich 2 Möglichkeiten entweder gutartiger Hautfleck (Hautveränderung) oder bösartiger Hautkrebs.

Aus 4.4 sieht man, wenn die Vorhersage ein **0** ist, dann ist das Testbild ein **Gutartiger Hautfleck (Hautveränderung)**, oder **1** ist, dann ist das Testbild ein **Bösartiger Hautkrebs**.

In binäre Bilderklassifikation-Modellen gibt es einen Grenzwert von 50%. Wenn das Ergebnis der Vorhersage von dem Eingabebild unterhalb 50% beträgt, dann ist das Bild gutartiger Hautfleck (Hautveränderung). Dagegen und oberhalb von 50%, ist das Bild bösartiger Hautkrebs.

Als Ergebnis hat das Modell einen Vorhersage-Wert von **[17.85%]**, was das bedeutet, dass das Testbild ein gutartiger Hautfleck (Hautveränderung) ist.

5 Skinlib

5.1 Einführung in Skinlib

Das klinische Ziel von Skinlib besteht darin, Bemühungen zur Reduzierung melanombedingter Todesfälle, indem die Genauigkeit und Effizienz der Melanomfrüherkennung verbessert wird.

Skinlib ist ein Projekt, in dem ein deep learning-Modell für die **muliklassen Hautbilderklassifikation** aufgebaut wurde. Das Modell wurde in **Kaggle**-Kernel mithilfe von Python-Bibliotheken TensorFlow 4.1.1 und Keras 4.1.2 programmiert, wobei der Datensatz von den Bildern von **The International Skin Imaging Collaboration (ISIC)** 3.2 stammen.

ISIC entwickelt die vorgeschlagene Standards für die digitale Bildgebung und bindet die Dermatologie und Informatik ein, um die diagnostische Genauigkeit mithilfe von KI zu verbessern. Während der anfängliche Fokus von ISIC auf Melanomen liegt, sind die von ISIC verfolgten Ziele entscheidend für die Weiterentwicklung der breiteren Landschaft der Hautbildung und künstlichen Intelligenz in der Dermatologie.

Nachdem das CNN-Modell in Kaggle trainiert und evaluiert wurde, wurde dann gespeichert und in h5-Format im PC heruntergeladen.

Dieses Modell wurde danach in einer Webseite mittels Django integriert, um Nutzer ihre Hautbildern hochladen und das Klassifizierungssystem verwenden zu können.

5.2 CNN-Modell in Skinlib

Der Auswahl von dem Kaggle-Kernel(Notebook) im ISIC-Datensatz war eine gute Idee für die Erstellung des CNN-Modells in Skinlib, denn der online Kaggle-Kernel(Notebook) bietet eine ideale Umgebung, um den Code des CNN-Modells zu schreiben. Außerdem kann Pfad von dem Datensatz im Code einfach abgerufen und das GPU-Device in Kaggle-Kernel(Notebook) passt für die Leistung der Bildverarbeitung sowie des Trainingsprozesses von 21490 Hautveränderungsbildern(10GB).

In folgenden Implementierungsplan in 5.1 wurde das CNN-Modell im Kaggle-Kernel(Notebook) erstellt.

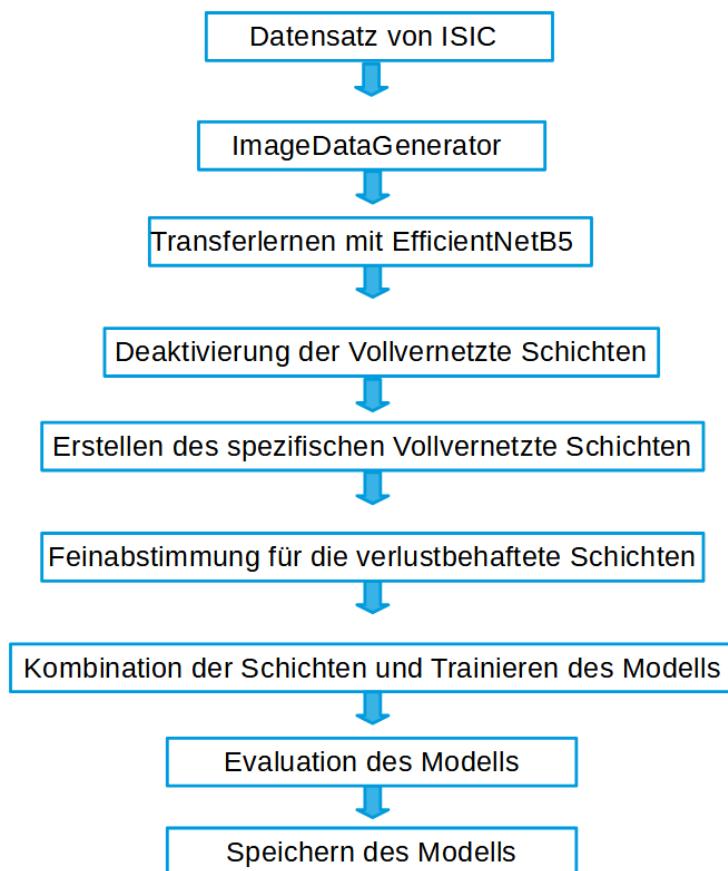
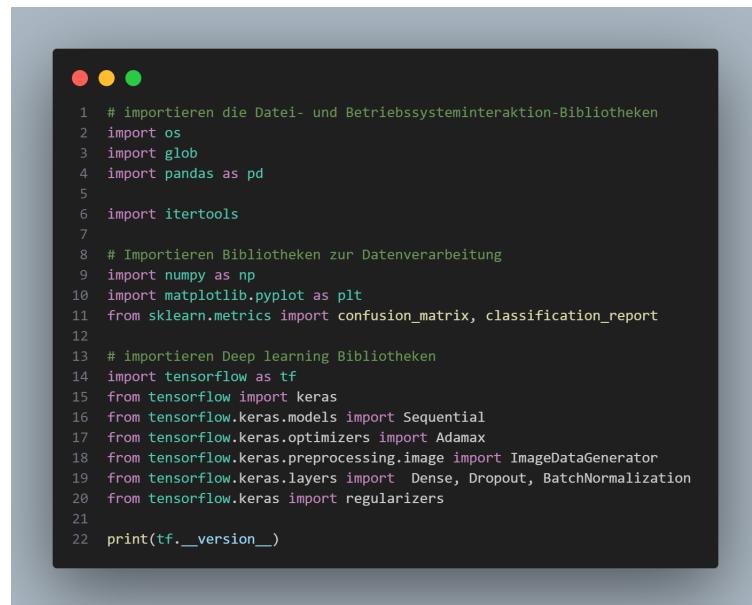


Abbildung 5.1: Implementierungsplan des CNN-Modell in Skinlib

Nach dem Klicken auf den hinzufügten Codebildern, dann findet man das Github-Repository [17], in dem sich das ganze Skinlib-Projekt mit den ausführlichen Kommentaren befindet.

5.2.1 Importieren aller im Code verwendeten Bibliotheken

Im Kaggle-Kernel(Notebook) sind alle benötigten Bibliotheken und Funktionen vorhanden, deswegen braucht man keine Installation von den Bibliotheken. Es reicht dann, die notwendige Bibliotheken oder Funktion unmittelbar zu importieren.



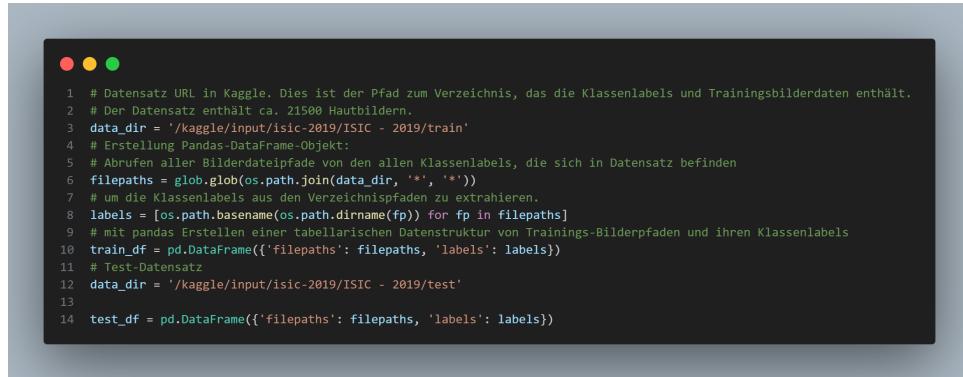
```
1 # importieren die Datei- und Betriebssysteminteraktion-Bibliotheken
2 import os
3 import glob
4 import pandas as pd
5
6 import itertools
7
8 # Importieren Bibliotheken zur Datenverarbeitung
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from sklearn.metrics import confusion_matrix, classification_report
12
13 # importieren Deep learning Bibliotheken
14 import tensorflow as tf
15 from tensorflow import keras
16 from tensorflow.keras.models import Sequential
17 from tensorflow.keras.optimizers import Adamax
18 from tensorflow.keras.preprocessing.image import ImageDataGenerator
19 from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
20 from tensorflow.keras import regularizers
21
22 print(tf.__version__)
```

Abbildung 5.2: Importieren aller im Code verwendeten Bibliotheken

In 5.2 wurde Tensorflow aus der aktuellen Version **2.13.0** importiert.

5.2.2 Abrufen der Training und Test-Bilderdaten von ISIC

Zuerst wurde in `data_dir` in 5.3 der Pfad-URL aus dem Datensatz definiert. Unter diesem Pfad befindet sich die Trainings- und Test-Bilderdaten.



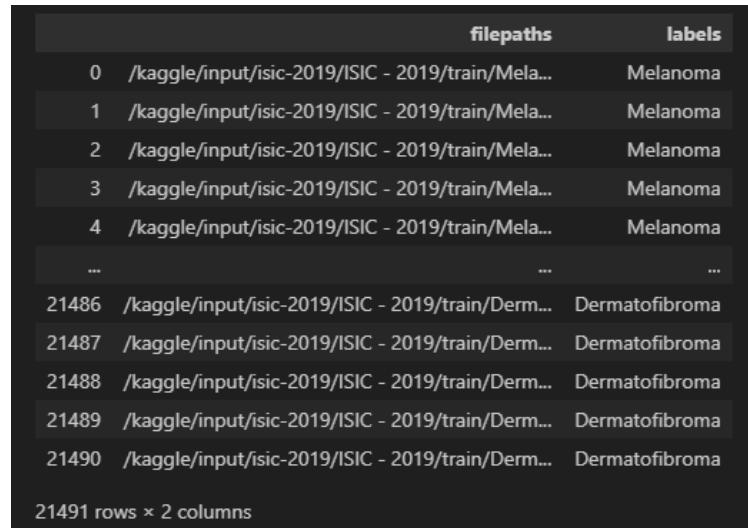
```

1 # Datensatz URL in Kaggle. Dies ist der Pfad zum Verzeichnis, das die Klassenlabels und Trainingsbilderdaten enthält.
2 # Der Datensatz enthält ca. 21500 Hautbildern.
3 data_dir = '/kaggle/input/isic-2019/ISIC - 2019/train'
4 # Erstellung Pandas-DataFrame-Objekt:
5 # Abrufen aller Bilderdateipfade von den alten Klassenlabels, die sich in Datensatz befinden
6 filepaths = glob.glob(os.path.join(data_dir, '*', '*'))
7 # um die Klassenlabels aus den Verzeichnispfaden zu extrahieren.
8 labels = [os.path.basename(os.path.dirname(fp)) for fp in filepaths]
9 # mit pandas Erstellen einer tabellarischen Datenstruktur von Trainings-Bilderpfaden und ihren Klassenlabels
10 train_df = pd.DataFrame({'filepaths': filepaths, 'labels': labels})
11 # Test-Datensatz
12 data_dir = '/kaggle/input/isic-2019/ISIC - 2019/test'
13
14 test_df = pd.DataFrame({'filepaths': filepaths, 'labels': labels})

```

Abbildung 5.3: Abrufen der Training und Test-Bilderdaten von ISIC

Danach wurde ein Pandas-DataFrame-Objekt mithilfe von **Pandas** 5.3 erstellt, um einer tabellarischen Datenstruktur von Trainings- und Test-Bilderpfaden mit den entsprechenden Klassenlabelnamen auszudrucken.



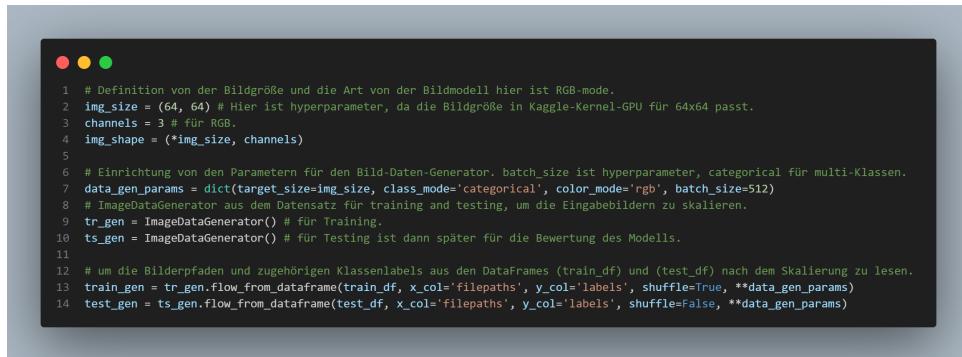
	filepaths	labels
0	/kaggle/input/isic-2019/ISIC - 2019/train/Mela...	Melanoma
1	/kaggle/input/isic-2019/ISIC - 2019/train/Mela...	Melanoma
2	/kaggle/input/isic-2019/ISIC - 2019/train/Mela...	Melanoma
3	/kaggle/input/isic-2019/ISIC - 2019/train/Mela...	Melanoma
4	/kaggle/input/isic-2019/ISIC - 2019/train/Mela...	Melanoma
...
21486	/kaggle/input/isic-2019/ISIC - 2019/train/Derm...	Dermatofibroma
21487	/kaggle/input/isic-2019/ISIC - 2019/train/Derm...	Dermatofibroma
21488	/kaggle/input/isic-2019/ISIC - 2019/train/Derm...	Dermatofibroma
21489	/kaggle/input/isic-2019/ISIC - 2019/train/Derm...	Dermatofibroma
21490	/kaggle/input/isic-2019/ISIC - 2019/train/Derm...	Dermatofibroma

21491 rows × 2 columns

Abbildung 5.4: Pandas-DataFrame-Objek

5.2.3 Erstellung von Bild-data-Generator

Zunächst wurde ein Bilderdaten-Generator erstellt. Dafür wurden als Eingabe die Training und Test-Bildern in den erstellten Pandas-DataFrame-Objekten eingegeben, mit dem Ziel die Bildern auf Größe **img_size=64x64** in 5.5 zu skalieren und diese für die Eingabe des CNN-Modells vorzubereiten. Diese Bildgröße wurde experimentell nach der GPU-Fähigkeiten in Kaggle-Notebook festgelegt.



```

1 # Definition von der Bildgröße und die Art von der Bildmodell hier ist RGB-mode.
2 img_size = (64, 64) # Hier ist hyperparameter, da die Bildgröße in Kaggle-Kernel-GPU für 64x64 passt.
3 channels = 3 # für RGB.
4 img_shape = (*img_size, channels)
5
6 # Einrichtung von den Parametern für den Bild-Daten-Generator. batch_size ist hyperparameter, categorical für multi-Klassen.
7 data_gen_params = dict(target_size=img_size, class_mode='categorical', color_mode='rgb', batch_size=512)
8 # ImageDataGenerator aus dem Datensatz für training and testing, um die Eingabebildern zu skalieren.
9 tr_gen = ImageDataGenerator() # für Training.
10 ts_gen = ImageDataGenerator() # für Testing ist dann später für die Bewertung des Modells.
11
12 # um die Bilderpfade und zugehörigen Klassenlabels aus den DataFrames (train_df) und (test_df) nach dem Skalierung zu lesen.
13 train_gen = tr_gen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', shuffle=True, **data_gen_params)
14 test_gen = ts_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', shuffle=False, **data_gen_params)

```

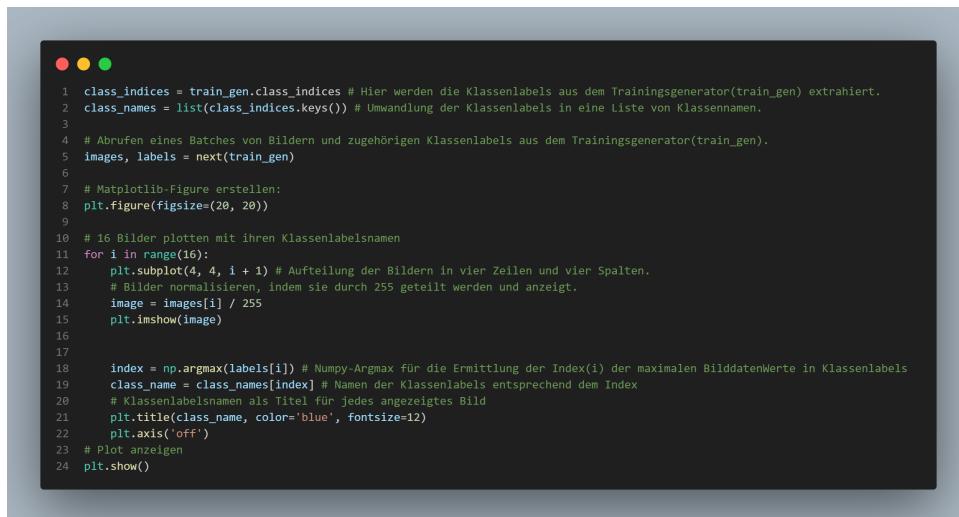
Abbildung 5.5: Erstellung von Bild-data-Generator

Andere Parametern wurde für den Bild-data-Generator wie **color_mode='rgb'**, weil es um farbige Hautbildern handelt, und **class_mode='categorical'**, da es um multiklassen Bildklassifikation geht. Hier wird ein **batch_size=512** verwendet, was bedeutet, dass während des Generierens von den skalierten Bildern 512 Bilder gleichzeitig in Bild-data-Generator eingespeist werden.

5.2.4 Bilderanzeigen von einem Batch in Training-Bilddaten

Es wurde hier 20 skalierten Hautbildern von dem ersten Batch mithilfe von **images, labels = next(train_gen)** in 5.6 angezeigt. Ein Batch enthält die Bildern und die entsprechenden Klassenlabels. Die Anzeige erfolgt mit dem Python-Bibliothek **matplotlib.pyplot** in 5.7.

Dies 20 Bildern wurden nicht zufällig ausgewählt, sondern sind Bildern mit der höchsten Bilddatenwerte **np.argmax(labels[i])** in 5.6 in jeder Klasse der skalierten Training-Datensatz nach dem Normalisierung **image = images[i] / 255.**



```

1  class_indices = train_gen.class_indices # Hier werden die Klassenlabels aus dem Trainingsgenerator(train_gen) extrahiert.
2  class_names = list(class_indices.keys()) # Umwandlung der Klassenlabels in eine Liste von Klassennamen.
3
4  # Abrufen eines Batches von Bildern und zugehörigen Klassenlabels aus dem Trainingsgenerator(train_gen).
5  images, labels = next(train_gen)
6
7  # Matplotlib-Figure erstellen:
8  plt.figure(figsize=(20, 20))
9
10 # 16 Bilder plotten mit ihren Klassenlabelsnamen
11 for i in range(16):
12     plt.subplot(4, 4, i + 1) # Aufteilung der Bildern in vier Zeilen und vier Spalten.
13     # Bilder normalisieren, indem sie durch 255 geteilt werden und angezeigt.
14     image = images[i] / 255
15     plt.imshow(image)
16
17
18     index = np.argmax(labels[i]) # Numpy-Argmax für die Ermittlung der Index(i) der maximalen BilddatenWerte in Klassenlabels
19     class_name = class_names[index] # Namen der Klassenlabels entsprechend dem Index
20     # Klassenlabelsnamen als Titel für jedes angezeigtes Bild
21     plt.title(class_name, color='blue', fontsize=12)
22     plt.axis('off')
23 # Plot anzeigen
24 plt.show()

```

Abbildung 5.6: Bilderanzeigen von einem Batch in Training-Bilddaten



Abbildung 5.7: Plot-Figure Batch in Training-Bilddaten

5.2.5 Transferlernen mittels EfficientNetB5 von Keras

Für unseren großen Datensatz (21490 Hautbildern 10GB) ist es notwendig, das Transferlernen-Verfahren zu nutzen. Dieses Verfahren besteht darin, ein bereits trainiertes Bildklassifizierungsmodell aus Keras-API zu verwenden.

Das Ziel hinter Transferlernen-Verfahren besteht darin, ein CNN-Modell zu verwenden, das eine hohe Genauigkeit bei der Klassifikation von Bildern gewährleistet, während es gleichzeitig sowohl recheneffizient als auch speichereffizient ist. Das bedeutet, um gute Ergebnisse zu erzielen, wird es weniger Ressourcen (wie Trainingsaufwands oder Rechenkapazität) benötigt.

In Keras-API sind verschiedene vorgeübte Convolutional Neural Network (CNN)-Modelle verfügbar, die für das Transferlernen verwendet werden können. Diese CNN-Modelle aus Keras in 5.8 wurden mit dem **ImageNet**-Datensatz trainiert. **ImageNet**-Datensatz ist sehr umfangreich und enthält Millionen von gelabelten Bildern in Tausenden von Kategorien, was es zu einer wertvollen Ressource für das Training von Bilderkennungsmodellen macht.

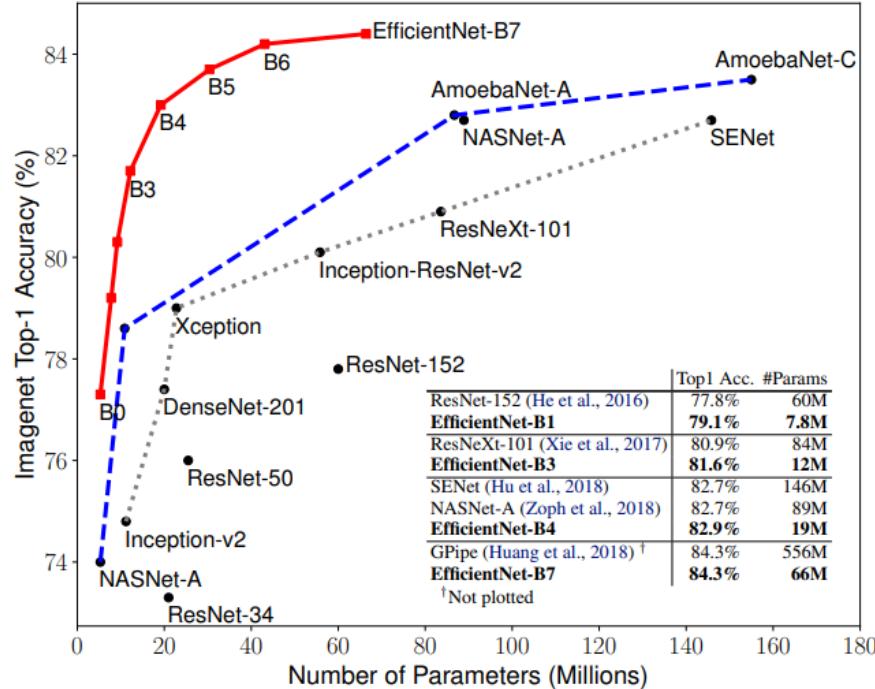


Abbildung 5.8: KerasAPI-Modellen [18]

Zu den KerasAPI-Modellen gehören **EfficientNet B0 bis B7**, die von Google AI Forschern im 2019 entwickelt wurden, um eine optimale Balance zwischen Modellgröße, Genauigkeit und Rechenressourcen zu finden. Aufgrund ihrer Effizienz und Leistungsfähigkeit sind sehr beliebt.

EfficientNet B0 bis B7 variieren in ihrer Architektur, Größe und Komplexität, wobei B0 das kleinste und einfachste Modell ist und B7 das größte und komplexeste. In diesem Projekt wurde das CNN-Modell **EfficientNetB5** in 5.9 aus Keras für das Transferlernen-Verfahren verwendet.

EfficientNetB5 function

```
keras.applications.EfficientNetB5(  
    include_top=True,  
    weights="imagenet",  
    input_tensor=None,  
    input_shape=None,  
    pooling=None,  
    classes=1000,  
    classifier_activation="softmax",  
    **kwargs  
)
```

Abbildung 5.9: EfficientNetB5-Architektur [19]

EfficientNetB5 aus Keras ist ein "pre-trained model", das schon auf großen Datensatz **ImageNet** trainiert wurde. EfficientNet erreicht eine hochmoderne Top-1-Genauigkeit von ca. 90% bei **ImageNet**-Datensatz und ist gleichzeitig kleiner und schneller als das erst entwickelte in 5.10 CNN-Modell **AlexNet** mit Top-1-Genauigkeit von 63.3%, das im 2012 das ImageNet Large Scale Visual Recognition Challenge (ILSVRC) gewann. Sowohl AlexNet als auch EfficientNet sind Convolutional Neural Networks (CNNs), die für die Aufgabe der Bilderkennung entwickelt wurden.

Insgesamt kann man sagen, dass EfficientNet im Vergleich zu AlexNet eine moderne, effizientere und skalierbarere Architektur ist, die speziell darauf abzielt, eine gute Leistung bei minimalen Ressourcen zu erreichen.

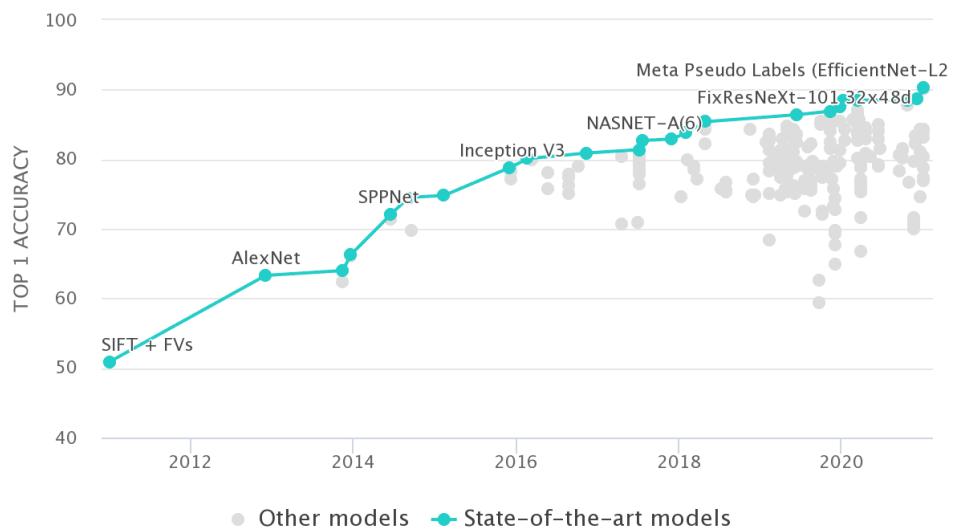


Abbildung 5.10: AlexNet vs EfficientNet [20]

Hier ist eine Schritt-für-Schritt Erklärung, wie Transferlernen in Keras für das CNN-Modell in Skinlib funktioniert:

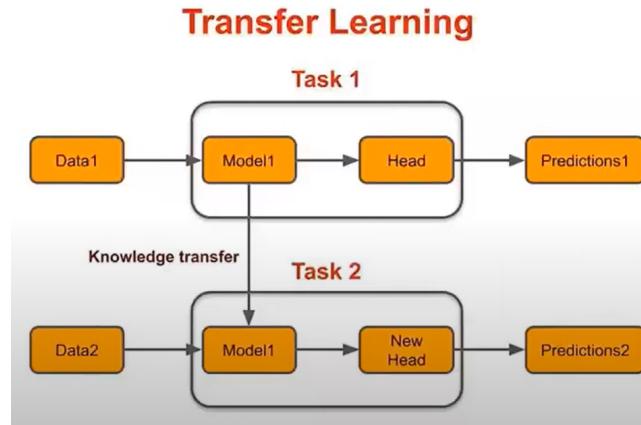
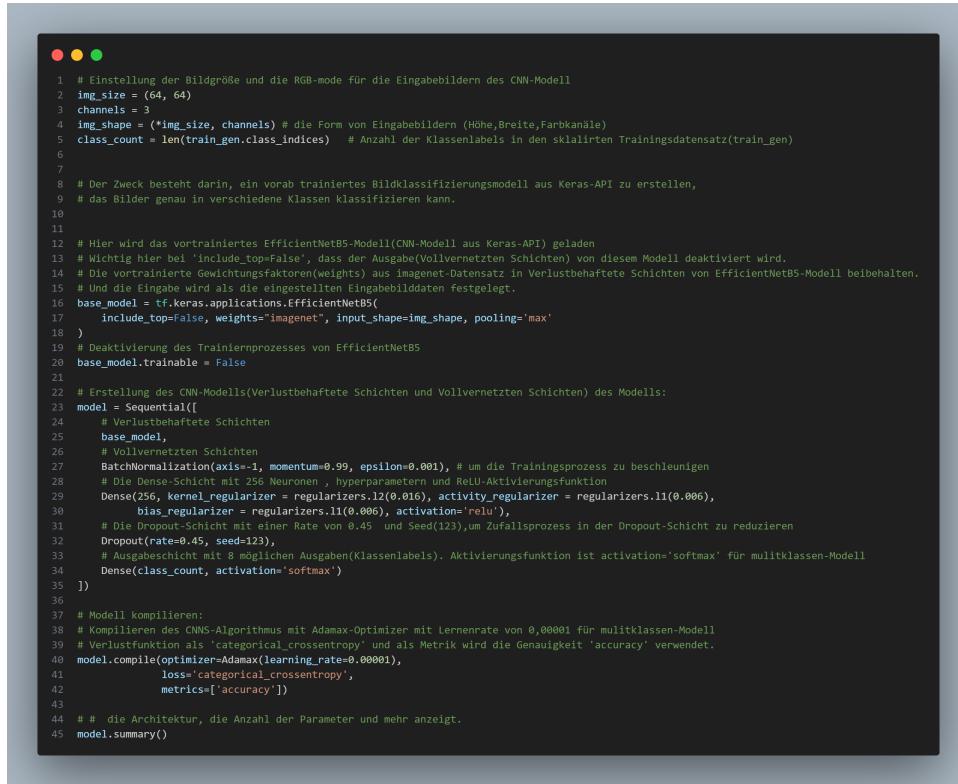


Abbildung 5.11: Transfer learning [21]

- Auswahl eines vortrainierterem CNN-Modell, das auf einem großen Datensatz trainiert wurde. Für Skinlib CNN-Modell wurde **EfficientNetB5** ausgewählt, das bereits gelernt hat, Merkmale aus den Bilddaten in **ImageNet** zu extrahieren, um verschiedene Objekte und Muster zu erkennen.
- Die Verlustbehafteten Schichten des vortrainiertem Modell **Modell_1** in 5.11 werden genommen.
- Die letzten Schichten (Vollvernetzten Schichten) des vortrainiertem Modell **Head in Task 1** in 5.11 werden entfernt, da sie spezifisch für den ursprünglichen Datensatz (**Data_1 = ImageNet**) in **Task 1** in 5.11 sind.
- Anstelle des entfernten Klassifikationskopfs wird ein neuer Klassifikationskopf **New Head in Task 2** in 5.11 hinzugefügt, der auf die spezifische Zielklasse der Hautkrebsbilderklassifikation abgestimmt ist.
- Danach wird die Feinabstimmung (Feintuning) durchgeführt, um aus bestimmten Anzahl der Verlustbehafteten Schichten des vortrainiertem Modell die trainierte Gewichtungsfaktoren zu nehmen.
- Letztendlich wird die Kompilierung des Trainingsprozess auf dem spezifischen Skinlib CNN-Modell durchgeführt.

5.2.6 Erstellung des CNN (convolutional neuronales Netzwerk)-Modell

In den folgenden Schritten im Code in 5.12 wurde das **EfficientNetB5** geladen und für spezifischen Fall von der Hautbilderklassifikation angepasst:



```

1 # Einstellung der Bildgröße und die RGB-mode für die Eingabebildern des CNN-Modell
2 img_size = (64, 64)
3 channels = 3
4 img_shape = (*img_size, channels) # die Form von Eingabebildern (Höhe,Breite,Farbkänele)
5 class_count = len(train_gen.class_indices) # Anzahl der Klassenlabels in den skalierten Trainingsdatensatz(train_gen)
6
7
8 # Der Zweck besteht darin, ein vorab trainiertes Bildklassifizierungsmodell aus Keras-API zu erstellen,
9 # das Bilder genau in verschiedene Klassen klassifizieren kann.
10
11
12 # Hier wird das vorgebildete EfficientNetB5-Modell(CNN-Modell aus Keras-API) geladen
13 # Wichtig hier bei 'include_top=False', dass der Ausgabe(Vollvernetzten Schichten) von diesem Modell deaktiviert wird.
14 # Die vorgebildete Gewichtungsfaktoren(weights) aus imagenet-Datensatz in Verlustbehaftete Schichten von EfficientNetB5-Modell beibehalten.
15 # Und die Eingabe wird als die eingestellten Eingabebilddaten festgelegt.
16 base_model = tf.keras.applications.EfficientNetB5(
17     include_top=False, weights="imagenet", input_shape=img_shape, pooling='max'
18 )
19 # Deaktivierung des Trainierenprozesses von EfficientNetB5
20 base_model.trainable = False
21
22 # Erstellung des CNN-Modells(Verlustbehaftete Schichten und Vollvernetzten Schichten) des Modells:
23 model = Sequential([
24     # Verlustbehaftete Schichten
25     base_model,
26     # Vollvernetzten Schichten
27     BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001), # um die Trainingsprozess zu beschleunigen
28     # Die Dense-Schicht mit 256 Neuronen , hyperparametern und ReLU-Aktivierungsfunktion
29     Dense(256, kernel_regularizer = regularizers.l2(0.016), activity_regularizer = regularizers.l1(0.006),
30           bias_regularizer = regularizers.l1(0.006), activation='relu'),
31     # Die Dropout-Schicht mit einer Rate von 0.45 und Seed(123),um Zufallsprozess in der Dropout-Schicht zu reduzieren
32     Dropout(rate=0.45, seed=123),
33     # Ausgabeschicht mit 8 möglichen Ausgaben(Klassenlabels). Aktivierungsfunktion ist activation='softmax' für multiklassen-Modell
34     Dense(class_count, activation='softmax')
35 ])
36
37 # Modell kompilieren:
38 # Kompilieren des CNNs-Algorithmus mit Adamax-Optimierer mit Lernrate von 0,00001 für multiklassen-Modell
39 # Verlustfunktion als 'categorical_crossentropy' und als Metrik wird die Genauigkeit 'accuracy' verwendet.
40 model.compile(optimizer=Adamax(learning_rate=0.00001),
41                 loss='categorical_crossentropy',
42                 metrics=['accuracy'])
43
44 # # die Architektur, die Anzahl der Parameter und mehr anzeigen.
45 model.summary()

```

Abbildung 5.12: Erstellung des CNN-Modell basierend auf der Keras/EfficientNetB5-Architektur

- In den ersten Zeilen 2 bis 4 in 5.12 wurde die Bildgröße und die RGB-mode für die Eingabebildern des CNN-Modell eingestellt, wobei das **img_size=(64x64)** ist und **channels = 3** für RGB-mode ist, da sich um farbige Hautbildern handelt.
`class_count=len(train_gen.class_indices)` gibt an, der Anzahl der Klassenlabels in den sklalirten Trainingsdatensatz(`train_gen`) und das beträgt 8 Klassen von den Hautveränderungsarten in Datensatz.
- Zunächst wurde in der Zeile 16 in 5.12 das vortrainiertes **EfficientNetB5-Modell** aus Keras-API geladen und in **base_model**-Variable gespeichert, wobei hier mit **include_top=False** der Ausgabe der vollvernetzten Schichten von **EfficientNetB5-Modell** deaktiviert wird.

Dieser Trick (Transferlernen) wurde hier durchgeführt, um zwei Ziele zu erreichen. Erstmal wird damit die vortrainierte **Gewichtungsfaktoren(weights)** in Verlustbehaftete Schichten von **EfficientNetB5-Modell** beibehalten. Diese Schicht wird dann in unserem Modell unter dem definierten Einstellungen in **input_shape=img_shape** verwendet.

Zweitens wurde die Deaktivierung von dem vollvernetzte (neuronale) Schichten des geladenen **EfficientNetB5-Modell** gemacht, um unsere spezifisch vollvernetzte Schichten für die Hautbilderklassifikation aufgebaut zu werden.

Außerdem wurde das Trainingsprozess mit **base_model.trainable = False** deaktiviert, damit das geladenes Modell nicht zwei mal trainiert wird.

- Danach wurde das CNN-Modell **model** in der Zeile 23 in 5.12 entsprechend der verlustbehafteten Schichten aus **base_model** und unserer spezifisch vollvernetzten Schichten aufgebaut.
Vor der vollvernetzten Schichten wurde ein **BatchNormalization** mit ihrem Hyperparametern hinzugefügt, um die Trainingsprozess zu beschleunigen. Diese Hyperparametern kann man je nach der Art der Bildern im Datensatz experimentell einstellen.

Als **Fully-connected Layer** oder **die vollvernetzte (neuronale) Schicht** wurden 2 **Dense-Schichten** in der Zeilen 29 und 34 in 5.12 hinzugefügt.

Bei einer Dense-Schicht ist jeder Ausgang jedes Neurons mit jedem Eingangsneuron verbunden. Diese Verbindungen haben Gewichte, die während des Trainings angepasst werden, um das Modell zu optimieren.

Die Dense-Schichten spielen eine wichtige Rolle bei der Zusammenführung der im CNN gelernten Merkmale, um eine endgültige Vorhersage zu treffen. Sie nehmen die Merkmale, die von den vorherigen Convolutional-Schichten extrahiert wurden, und lernen, wie sie zu einer bestimmten Vorhersage kombiniert werden können.

Die erste Dense-Schicht mit 256 Neuronen und **ReLU**-Aktivierungsfunktion dient als Eingabeschicht von der ausgegeben geflatteten Vektor aus der verlustbehafteten Schicht.

Danach wurde Dropout-Technik durchgeführt, um das Overfitting oder Überanpassung zu reduzieren.

Die zweite Dense-Schicht mit 8 Ausgabe-Neuronen und **Softmax**-Aktivierungsfunktion wird hinzugefügt, um die Ausgabe der Klassifikation zu erhalten. Die Anzahl der Ausgabe-Neuronen entspricht der Anzahl der Klassen, die das Modell vorhersagen soll. Die **Softmax**-Aktivierungsfunktion wird verwendet, um Wahrscheinlichkeiten für jede Klasse der Hautkrebsarten in 3.1 zu erzeugen.

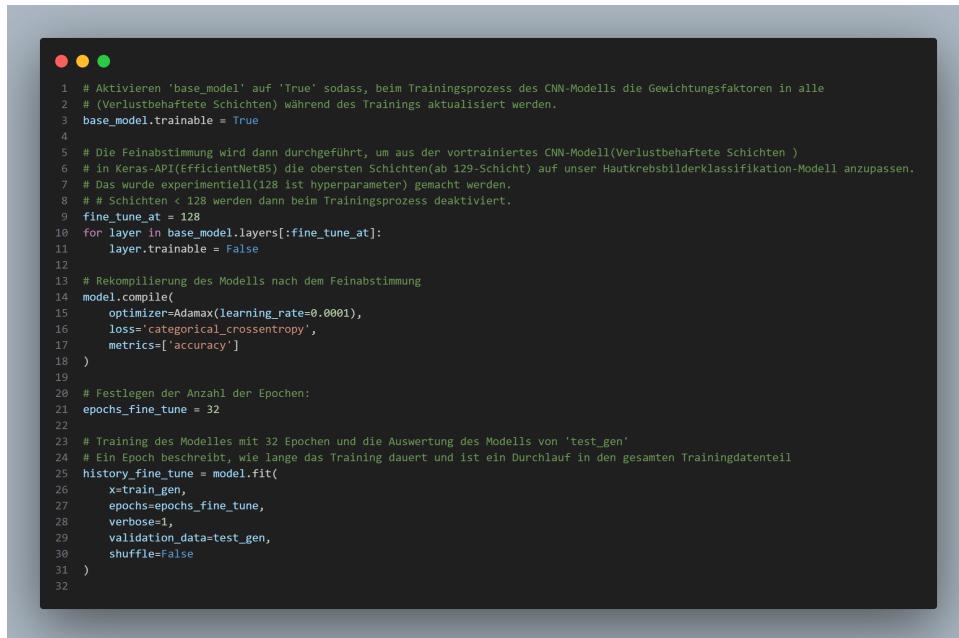
- Zunächst wurde das Modell in der Zeile 40 in 5.12 kompiliert. Es wird der Optimierungsalgorithmus **Adamax** mit einer Lernrate von **0,00001** verwendet. Die Verlustfunktion ist die **categorical_crossentropy**, die für multiklassige Klassifikationsaufgaben geeignet ist. Als Metrik wird die Genauigkeit **accuracy** verwendet.
- **model.summary()** gibt eine Zusammenfassung des Modells in 5.13 aus und der Anzahl der Parameter in jeder Schicht.

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb5\_notop.h5
115263384/115263384 [=====] - 1s 0us/step
Model: "sequential"
-----  
Layer (type)          Output Shape         Param #
-----  
efficientnetb5 (Functional (None, 2048)      28513527
)  
  
batch_normalization (Batch Normalization) (None, 2048)    8192  
  
dense (Dense)          (None, 256)           524544  
  
dropout (Dropout)       (None, 256)           0  
  
dense_1 (Dense)        (None, 8)             2056  
  
-----  
Total params: 29048319 (110.81 MB)
Trainable params: 530696 (2.02 MB)
Non-trainable params: 28517623 (108.79 MB)
```

Abbildung 5.13: Zusammenfassung des Modells-Skinlib

5.2.7 Trainieren des erstellten CNN-Modells

Um das Trainingsprozess von der gesamten CNN-Modell durchzuführen, muss zuerst **base_model** oder die geladen verlustbehafteten Schichten auf **True** wieder aktiviert, sodass während des Trainingsprozesses des CNN-Modells die Gewichtungsfaktoren in alle (Verlustbehaftete Schichten) aktualisiert werden. Das Idee besteht darin, dass das Feinabstimmungsverfahren für den Trainingsdatensatz aus den Hautbildern erzielt wird.



```

1 # Aktivieren 'base_model' auf 'True' sodass, beim Trainingsprozess des CNN-Modells die Gewichtungsfaktoren in alle
2 # (Verlustbehaftete Schichten) während des Trainings aktualisiert werden.
3 base_model.trainable = True
4
5 # Die Feinabstimmung wird dann durchgeführt, um aus der vortrainiertes CNN-Modell(Verlustbehaftete Schichten )
6 # in Keras-API(EfficientNetB5) die obersten Schichten(ab 129-Schicht) auf unser Hautkrebsbilderklassifikation-Modell anzupassen.
7 # Das wurde experimentell(128 ist hyperparameter) gemacht werden.
8 # # Schichten < 128 werden dann beim Trainingsprozess deaktiviert.
9 fine_tune_at = 128
10 for layer in base_model.layers[:fine_tune_at]:
11     layer.trainable = False
12
13 # Rekomplierung des Modells nach dem Feinabstimmung
14 model.compile(
15     optimizer=Adamax(learning_rate=0.0001),
16     loss='categorical_crossentropy',
17     metrics=['accuracy']
18 )
19
20 # Festlegen der Anzahl der Epochen:
21 epochs_fine_tune = 32
22
23 # Training des Modelles mit 32 Epochen und die Auswertung des Modells von 'test_gen'
24 # Ein Epoch beschreibt, wie lange das Training dauert und ist ein Durchlauf in den gesamten Trainingdatenteil
25 history_fine_tune = model.fit(
26     x_train_gen,
27     epochs=epochs_fine_tune,
28     verbose=1,
29     validation_data=test_gen,
30     shuffle=False
31 )
32

```

Abbildung 5.14: Trainieren des erstellten CNN-Modells

Die Feinabstimmung wird dann in der Zeile 9 in 5.14 durchgeführt, um aus der vortrainiertes CNN-Modell(Verlustbehaftete Schichten) in **Keras-API(EfficientNetB5)** die obersten Schichten(ab 129-Schicht) auf unser Hautbilderklassifikation-Modell anzupassen. Das wurde experimentell (128 ist hyperparameter) gemacht. Schichten < 128 werden dann beim Trainingsprozess deaktiviert.

Zunächst wird das Modell nach dem Feinabstimmung mit den gleichen Kompilierungsparametern rekompiliert. Es wurde auch die Anzahl der Epochen **epochs_fine_tune = 32** festgelegt, wobei Ein Epoch beschreibt, wie lange das Training dauert und ist ein Durchlauf in den gesamten Trainingsdatenteil.

Nachdem das Modell aufgebaut wurde, können wir dann mit dem Trainingsprozess beginnen. Dies geschieht mit dem Funktion `history_fine_tune = model.fit()` in der Zeile 25 in 5.14. Hier wurde die Testdatensatz `validation_data=test_gen` für die Auswertung des Modells während des Trainingsprozess mitberücksichtigt und das Trainingsprozess mit 32 in 5.15 Epochen durchgelaufen.

```
Epoch 1/32
2023-12-28 21:07:49.634836: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed: INVALID_ARGUMENT: Size of values
42/42 [=====] - 556s 12s/step - loss: 10.2920 - accuracy: 0.2647 - val_loss: 8.9619 - val_accuracy: 0.5066
Epoch 2/32
42/42 [=====] - 353s 8s/step - loss: 9.4461 - accuracy: 0.4275 - val_loss: 8.5482 - val_accuracy: 0.5488
Epoch 3/32
42/42 [=====] - 359s 9s/step - loss: 8.9261 - accuracy: 0.4950 - val_loss: 8.3464 - val_accuracy: 0.5738
Epoch 4/32
42/42 [=====] - 355s 9s/step - loss: 8.4096 - accuracy: 0.5371 - val_loss: 8.2535 - val_accuracy: 0.5957
Epoch 5/32
42/42 [=====] - 356s 9s/step - loss: 7.9950 - accuracy: 0.5610 - val_loss: 7.8660 - val_accuracy: 0.6135
Epoch 6/32
42/42 [=====] - 376s 9s/step - loss: 7.6564 - accuracy: 0.5782 - val_loss: 7.4838 - val_accuracy: 0.6278
Epoch 7/32
42/42 [=====] - 360s 9s/step - loss: 7.3640 - accuracy: 0.5970 - val_loss: 7.1711 - val_accuracy: 0.6388
Epoch 8/32
42/42 [=====] - 359s 9s/step - loss: 7.0789 - accuracy: 0.6108 - val_loss: 6.8800 - val_accuracy: 0.6499
Epoch 9/32
42/42 [=====] - 357s 9s/step - loss: 6.8252 - accuracy: 0.6151 - val_loss: 6.6141 - val_accuracy: 0.6581
Epoch 10/32
42/42 [=====] - 362s 9s/step - loss: 6.5729 - accuracy: 0.6236 - val_loss: 6.3536 - val_accuracy: 0.6635
Epoch 11/32
42/42 [=====] - 356s 9s/step - loss: 6.3291 - accuracy: 0.6284 - val_loss: 6.1073 - val_accuracy: 0.6690
Epoch 12/32
42/42 [=====] - 358s 9s/step - loss: 6.0899 - accuracy: 0.6320 - val_loss: 5.8683 - val_accuracy: 0.6732
Epoch 13/32
42/42 [=====] - 357s 9s/step - loss: 5.8592 - accuracy: 0.6412 - val_loss: 5.6325 - val_accuracy: 0.6795
...
Epoch 31/32
42/42 [=====] - 360s 9s/step - loss: 2.4618 - accuracy: 0.7675 - val_loss: 2.2232 - val_accuracy: 0.8404
Epoch 32/32
42/42 [=====] - 355s 9s/step - loss: 2.3228 - accuracy: 0.7806 - val_loss: 2.0817 - val_accuracy: 0.8555
```

Abbildung 5.15: Epochen während des Trainierens des erstellten CNN-Modells

5.2.8 Darstellung der CNN-Modellleistung

Danach werden die Trainings- und Validierungsverläufe grafisch dargestellt, um die Genauigkeit und den Verlust über die 32 Epochen hinweg zu verfolgen.

Aus dem Trainingsprozess in 5.15 und der grafische Darstellung in 5.16 kann man sehen, dass der **Loss** abnimmt und die **Accuracy** im Verlauf der Epochen zunimmt, deutet dies darauf hin, dass das Modell die Muster und Merkmale in den Hautbildern aus dem Trainingsdatensatz erfasst und daraus lernen kann, die Hautveränderungsbildern korrekt zu klassifizieren und die tatsächlichen Klassen genau vorherzusagen. Es zeigt auch, dass das Modell effektiv trainiert wird und an Genauigkeit gewinnt.



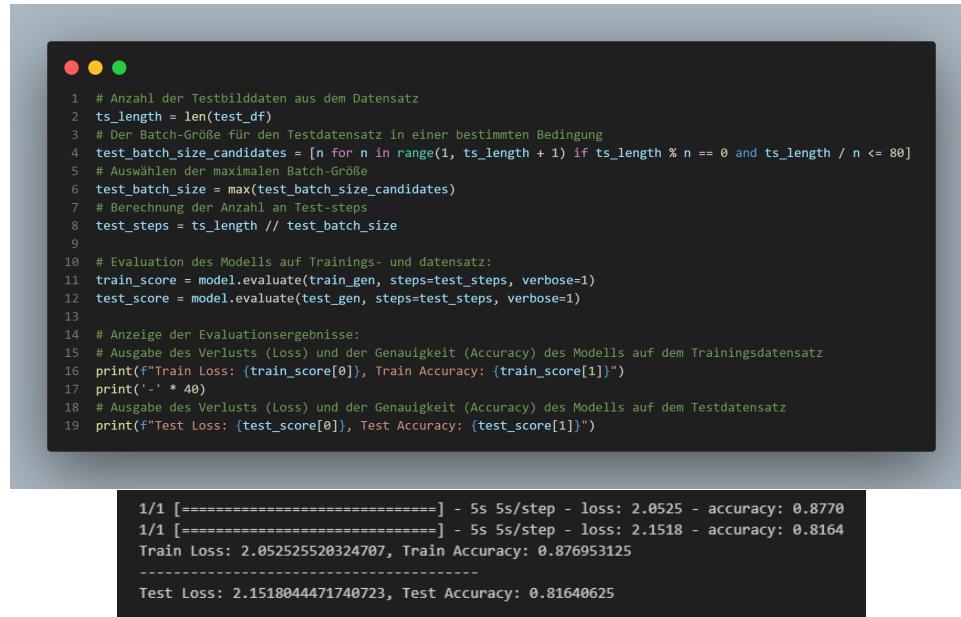
Abbildung 5.16: Darstellung der CNN-Modellleistung

Der **Val-Loss** gibt den Verlust auf den Validierungsdaten aus der Testdatensatz `test_gen` an, die nicht zum Trainieren, sondern zum Überwachen der Leistung des Modells verwendet werden. Ein abnehmender in 5.16 **Val-Loss** zeigt an, dass das Modell auch auf neuen Hautbildern gut generalisiert und nicht überanpasst (overfitting) ist.

Val-Accuracy (Validierungsgenauigkeit): Die Val-Accuracy misst die Genauigkeit des Modells auf den Validierungsdaten aus der Testdatensatz `test_gen`. Eine zunehmende in 5.16 **Val-Accuracy** zeigt an, dass das Modell gut generalisiert und in der Lage ist, auch unbekannte Bilddaten korrekt zu klassifizieren.

5.2.9 Evaluation des Skinlib CNN-Modells

Zunächst wurde die Auswertung des Modells auf Trainings- und Testdaten durchgeführt und danach wurde die Ergebnisse der Auswertung in 5.17 ausgegeben. Die Batch-Größe für die Auswertung wird so gewählt, dass sie die Daten effizient verarbeitet und die maximale Auslastung der Ressourcen berücksichtigt.



```

1 # Anzahl der Testbilddaten aus dem Datensatz
2 ts_length = len(test_df)
3 # Der Batch-Größe für den Testdatensatz in einer bestimmten Bedingung
4 test_batch_size_candidates = [n for n in range(1, ts_length + 1) if ts_length % n == 0 and ts_length / n <= 80]
5 # Auswählen der maximalen Batch-Größe
6 test_batch_size = max(test_batch_size_candidates)
7 # Berechnung der Anzahl an Test-steps
8 test_steps = ts_length // test_batch_size
9
10 # Evaluation des Modells auf Trainings- und datensatz:
11 train_score = model.evaluate(train_gen, steps=test_steps, verbose=1)
12 test_score = model.evaluate(test_gen, steps=test_steps, verbose=1)
13
14 # Anzeige der Evaluationsergebnisse:
15 # Ausgabe des Verlusts (Loss) und der Genauigkeit (Accuracy) des Modells auf dem Trainingsdatensatz
16 print(f"Train Loss: {train_score[0]}, Train Accuracy: {train_score[1]}")
17 print('-' * 40)
18 # Ausgabe des Verlusts (Loss) und der Genauigkeit (Accuracy) des Modells auf dem Testdatensatz
19 print(f"Test Loss: {test_score[0]}, Test Accuracy: {test_score[1]}")

```

1/1 [=====] - 5s 5s/step - loss: 2.0525 - accuracy: 0.8770
1/1 [=====] - 5s 5s/step - loss: 2.1518 - accuracy: 0.8164
Train Loss: 2.0525520324707, Train Accuracy: 0.876953125

Test Loss: 2.1518044471740723, Test Accuracy: 0.81640625

Abbildung 5.17: Evaluation des Skinlib CNN-Modells

Nachdem der Evaluation des CNN-Modells durchgeführt wurde sieht man die folgende Ergebnisse von der Gesamtevaluation:

- **Trainingsverlust (Train Loss): 2.0525**
- **Trainingsgenauigkeit (Train Accuracy): 87.70%**
- **Testverlust (Test Loss): 2.1518**
- **Testgenauigkeit (Test Accuracy): 81.64%**

Die Trainingsgenauigkeit liegt bei **87.70%**, während die Testgenauigkeit bei **81.64%** liegt. Dies deutet darauf hin, dass das Modell während des Trainings gut funktioniert hat.

Danach wurde das **Confusion-Matrix** in 5.18 basierend auf den wahren und vorhergesagten Klassenlabels des Testdatensatzes **test_gen** erstellt, um die Leistung des Modells visuell darzustellen, wobei in der Hauptdiagonale die Anzahl der Hautveränderungsbildern in jeder Klasse des Trainingsdatensatz repräsentiert, die das CNN-Modell anhand des Testdatensatzes richtig betroffen hat.

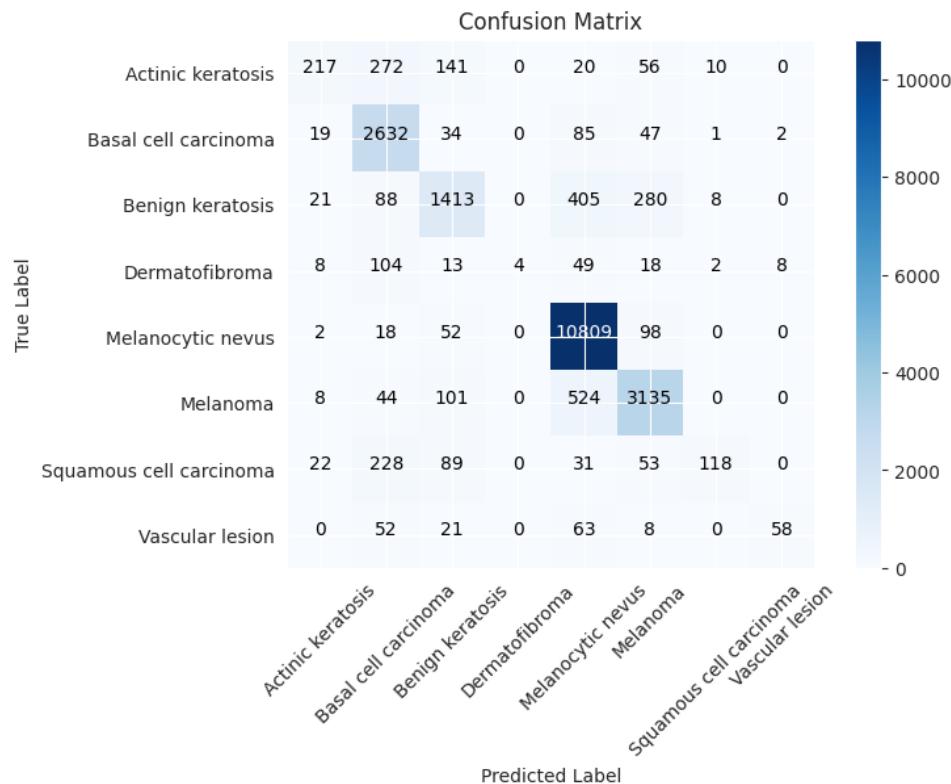


Abbildung 5.18: Confusion-Matrix des CNN-Modells

5.2.10 Speichern und Laden des CNN-Modells

Am Ende des Codes des CNN-Modells wurde das trainierte Modell im h5-Format in 5.19 gespeichert. Das gespeicherte Modell wird dann geladen, um ihr Leistung in Skinlib-Webseite zu verwenden, ohne es erneut trainieren zu müssen.

```

● ● ●
1 model.save('skin_cancer_model_multible_classifier.h5') # in h5-Format wird das CNN-Modell gespeichert
2
3 # Das gespeichertes CNN-Modell wird geladen, ohne es erneut trainieren zu müssen
4 # Das gespeichertes Modell wird dann in einer Webseite integriert, um Benutzern
5 # das Modell zu probieren und die Flecken auf ihren Hautbildern mithilfe des Modells zu testen.
6 loaded_model = keras.models.load_model('skin_cancer_model_multible_classifier.h5')
7

```

Abbildung 5.19: Speichern und Laden des CNN-Modells

In Kaggle gibt es sowie die Option für das Herunterladen des Modells auf dem eigenen PC als auch für das Speichern des CNN-Modell-Codes.

5.3 Skinlib-Webseite

Nachdem das Modell trainiert wurde und die gewünschte Genauigkeit erreicht wurde, wird es dann in Django integrieren, um eine responsive Webseite zu erstellen, die es Benutzern ermöglicht, Bilder von verdächtigen Hautstellen hochzuladen. Das hochgeladene Bild wird dann an das Modell weitergegeben, um eine Diagnose durchzuführen.

Die Webentwicklung besteht aus zwei Hauptbereichen: dem Frontend und dem Backend. Die Frontend- und Backend-Entwicklung sind eng miteinander verbunden und arbeiten zusammen, um eine vollständige Website oder Anwendung zu erstellen.

Als Codeumgebung-Applikation wurde **Visual Studio Code (VS)** ausgewählt, welche ein leichter, aber leistungsstarker Quellcode-Editor ist, der auf dem Desktop ausgeführt wird und für Windows, macOS und Linux verfügbar ist. Es bietet integrierte Unterstützung für (Hypertext Markup Language (HTML) , Cascading Style Sheets (CSS) und JavaScript, und verfügt über ein umfangreiches Ökosystem an Erweiterungen für andere Sprachen und Laufzeiten (wie Python, C++, C#, Java, PHP, Go, .NET).

5.3.1 Skinlib-Frontend

Das **Frontend** ist für die Darstellung der Benutzeroberfläche und Interaktion mit Benutzern verantwortlich. Es umfasst die **HTML** für die Struktur, **CSS** für das Design, **JavaScript** für die Funktionen und Interaktionen, und **Bootstrap**, der die Fähigkeit hat, Webseiten für verschiedene Bildschirmgrößen und Geräte responsive zu gestalten.

In Skinlib-Projekt wurde das Frontend-Templates mittels **HTML**, **CSS**, **JavaScript** und **Bootstrap** programmiert. Daraus wurden 6-HTML Dateien erstellt, um die Benutzeroberfläche aufzubauen.

Diese 6-HTML Dateien wurden später in Django-Projekt in 5.3.2 integriert, um die Anfragen und die Daten von den Benutzern entgegenzunehmen, diese auch zu verarbeiten und dann in den Datenbank(Django-Database) zu speichern.

5.3.2 Skinlib-Backend

Das **Backend** bereitstellt die erforderlichen Daten und Funktionen bereitstellt, um die vom Frontend angeforderten Aufgaben durchzuführen. Das Backend bezieht sich auf den Teil einer Webseite oder Webanwendung, der im Hintergrund abläuft und die Logik, Datenbanken und Berechnungen enthält. Es umfasst verschiedene Programmiersprachen wie PHP, Python, Ruby, Java oder Node.js sowie Datenbanken wie MySQL, MongoDB oder PostgreSQL. Das Backend verarbeitet die Anfragen von Benutzern, führt die erforderlichen Operationen durch und sendet die entsprechenden Daten an das Frontend zurück.

Als Backend-Framework wurde in Skinlib-Projekt **Django**-Framework ausgewählt. Django ist ein leistungsstarkes Webframework, das in Python geschrieben ist und Entwicklern dabei hilft, Webanwendungen effizient aufzubauen. Django verwendet das Model-View-Controller (MVC)-Muster in 5.20, um die Applikationen zu strukturieren.

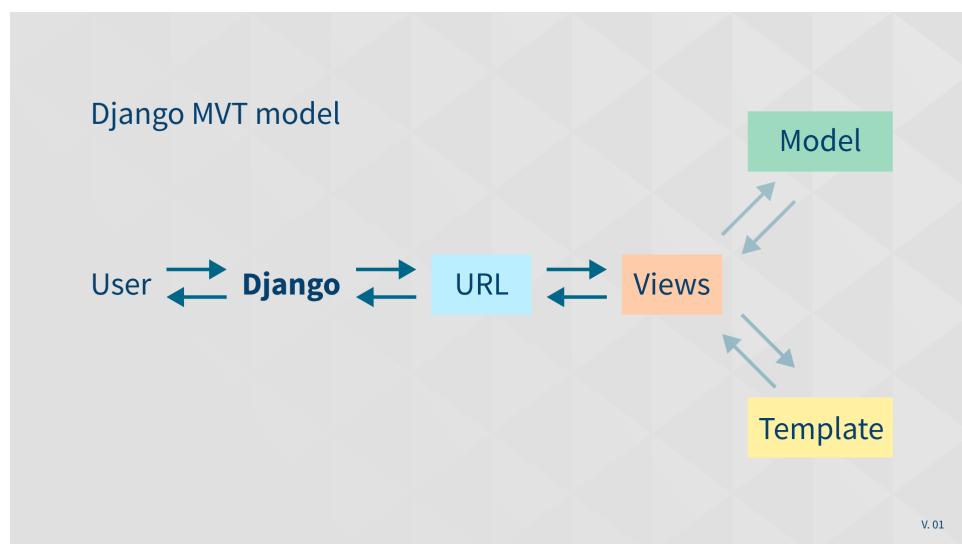


Abbildung 5.20: Django – Model Views Template [14]

Ein Django-Projekt besteht aus mehreren Komponenten:

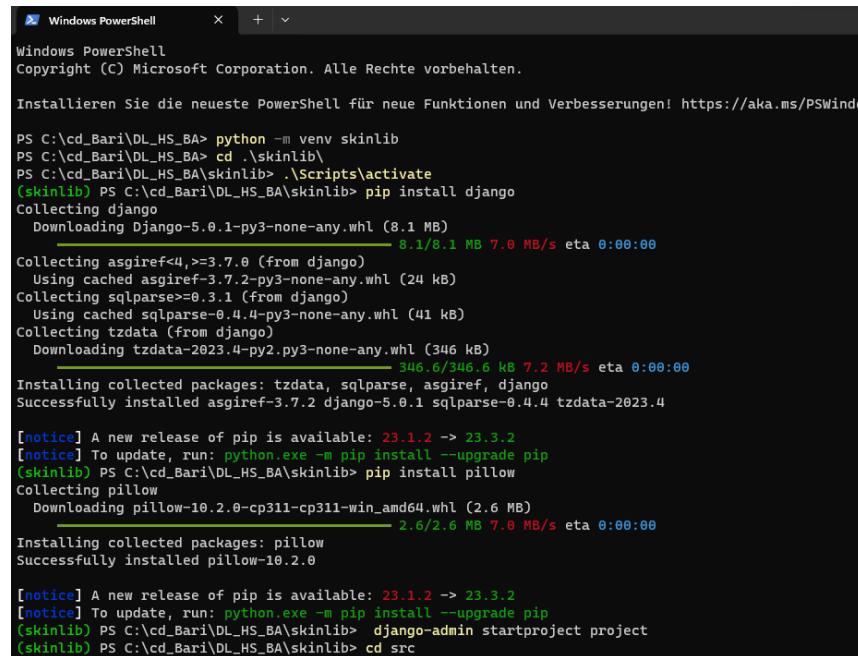
- **Models**: Models definieren die Datenstruktur der Anwendung und werden verwendet, um Daten in einer Datenbank zu speichern. Models repräsentieren typischerweise Tabellen in der Datenbank und verfügen über Methoden zur Datenverarbeitung.

- **Views:** Views nehmen Anfragen von Benutzern entgegen und tragen die Verantwortung für die Verarbeitung der Anfrage. Sie interagieren mit den Datenmodellen, nehmen erforderliche Berechnungen vor und wählen das entsprechende Template für die Darstellung der Antworten aus.
- **Templates:** Templates sind HTML-Dateien, die das Benutzerinterface der Anwendung definieren. Sie enthalten das Design, das Rendering von Daten aus der View ermöglicht und das Endergebnis für den Benutzer generiert.
- **URLs:** URLs definieren die Routing-Muster für die Anwendung und leiten Anfragen an die entsprechende View weiter. Hier wird festgelegt, welche Funktion oder Methode aufgerufen wird, wenn eine bestimmte URL aufgerufen wird.

Darüber hinaus gibt es andere Komponenten wie forms.py für die Verarbeitung von Benutzereingaben, eine Verwaltungsoberfläche für die einfache Verwaltung von Daten und die Einhaltung von Sicherheits- und Authentifizierungsaspekten.

5.3.3 Skinlib Django-Projekt

In den folgenden Schritten wird zunächst erklärt, wie das Skinlib Django-Projekt lokal im PC mittels Windows-PowerShell-Terminal erstellt wurde:



```

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Installieren Sie die neueste PowerShell für neue Funktionen und Verbesserungen! https://aka.ms/PSWindows

PS C:\cd_Bari\DL_HS_BA> python -m venv skinlib
PS C:\cd_Bari\DL_HS_BA> cd .\skinlib
PS C:\cd_Bari\DL_HS_BA\skinlib> .\scripts\activate
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib> pip install django
Collecting django
  Downloading Django-5.0.1-py3-none-any.whl (8.1 MB)
    ━━━━━━━━━━━━━━━━ 8.1/8.1 MB 7.0 MB/s eta 0:00:00
Collecting asgiref<4,>=3.7.0 (from django)
  Using cached asgiref-3.7.2-py3-none-any.whl (24 kB)
Collecting sqlparse>=0.3.1 (from django)
  Using cached sqlparse-0.4.4-py3-none-any.whl (41 kB)
Collecting tzdata (from django)
  Downloading tzdata-2023.4-py2.py3-none-any.whl (346 kB)
    ━━━━━━━━━━━━━━━━ 346.6/346.6 kB 7.2 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.7.2 django-5.0.1 sqlparse-0.4.4 tzdata-2023.4

[notice] A new release of pip is available: 23.1.2 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib> pip install pillow
Collecting pillow
  Downloading pillow-10.2.0-cp311-cp311-win_amd64.whl (2.6 MB)
    ━━━━━━━━━━━━━━━━ 2.6/2.6 MB 7.0 MB/s eta 0:00:00
Installing collected packages: pillow
Successfully installed pillow-10.2.0

[notice] A new release of pip is available: 23.1.2 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib> django-admin startproject project
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib> cd src

```

Abbildung 5.21: Skinlib Django virtuelle Umgebung Terminal 1

1. Um ein Projekt-Django zu erstellen, wurde zuerst in einem Datenträger z.B. in C oder D ein Ordner namens des Projekt ***DL_HS_BA*** erstellt.
2. In diesem Ordner mit der rechten Maustaste wird die Option ***In Terminal Öffnen*** gewählt. Und hier wird die virtuelle Umgebung oder im englischen ***virtual environments*** unter dem Befehl `python -m venv beliebige-Name` in 5.21 erstellt. In Skinlib-Project wurde den beliebigen-Name **Skinlib** genannt. Diese Dateiordner *djmlpraxis* wird in Terminal mit dem Befehl `cd Skinlib` geöffnet.
3. Und dann mit dem Befehl `.\Scripts\activate` leuchtet den Namen des Projekt in 5.21 in grün.
4. In Windows kann ein Fehler hier auftreten. Um der Fehler zu beheben, wird dieses Befehl in Terminal geschrieben. Dieser Befehl wird in der PowerShell verwendet, `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`, um das Ausführungsrichtlinien für Skripte auf einem Windows-System festzulegen.
5. In der virtuellen Umgebung wird dann Django mit dem Befehl `pip install django` in 5.21 installiert.
6. Dann wird Pillow-Bibliothek `pip install pillow` in 5.21 installiert. Pillow ist eine Erweiterung der Python Imaging Library (PIL) und bietet Funktionen zur Bildverarbeitung und -manipulation in Python. Es ermöglicht das Lesen, Schreiben und Bearbeiten verschiedener Bildformate, wie z.B. JPEG, PNG, GIF und BMP.
7. Mit dem Befehl `django-admin startproject project` in 5.21 wird ein django-Projekt erstellt und wird in dem Ordner *skinlib* neue Dateiordnern von Django heruntergeladen.
8. Es wird dann der Name von der Project-Dateiordner zu *src* geändert. So ist bekannt in Entwicklergesellschaft.
9. Es wird diese Dateiordner *src* in Terminal mit dem Befehl `cd src` in 5.21 geöffnet.
10. Das Befehl `python manage.py migrate` wird in 5.22 gegeben, um die Django-Datenbank(Database) zu erstellen.

11. Es muss in der nächsten Schritt ein Adminbenutzer Account in Django mit dem Befehl `python manage.py createsuperuser` in 5.22 erstellt werden. Hier wird dann der Username, Email-addresse und das Passwort eingegeben.

```
[notice] A new release of pip is available: 23.1.2 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib> django-admin startproject project
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib> cd src
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib\src> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib\src> python manage.py createsuperuser
Username (leave blank to use 'user'): HsTest
Email address: zb@gmail.com
Password:
Password (again):
Superuser created successfully.
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib\src> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 29, 2024 - 20:04:48
Django version 5.0.1, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[29/Jan/2024 20:05:27] "GET / HTTP/1.1" 200 10629
(skinlib) PS C:\cd_Bari\DL_HS_BA\skinlib\src> python manage.py startapp skinlib
```

Abbildung 5.22: Skinlib Django virtuelle Umgebung Terminal 2

12. Jetzt lässt sich dann den Server mit dem Befehl `python manage.py runserver` in 5.22 laufen.
13. Wenn der Link, der oben in 5.22 im Browser kopiert wird und in Browser eingefügt wird, öffnet sich dann die Hauptseite vom Django in 5.23.

[django](#) View [release notes](#) for Django 5.0



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

[Django Documentation](#) [Tutorial: A Polling App](#) [Django Community](#)

Abbildung 5.23: Skinlib Django server

14. Um ein Django-App zu erstellen, die die wichtige Python-Dateien wie `models.py`, `views.py` und `admin.py` enthält. Man gibt zunächst das Befehl in das Terminal
`python manage.py startapp skinlib` in 5.22 ein.
15. Mit dem Befehl in das Terminal `code .` in 5.22, wird VS in 5.24 geöffnet. Und hier sieht man die Ergebnisse, die schon im Terminal durchgeführt wurden.

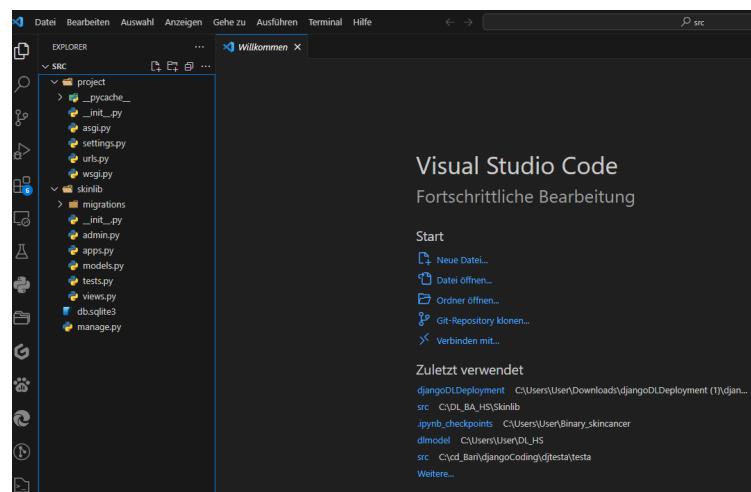


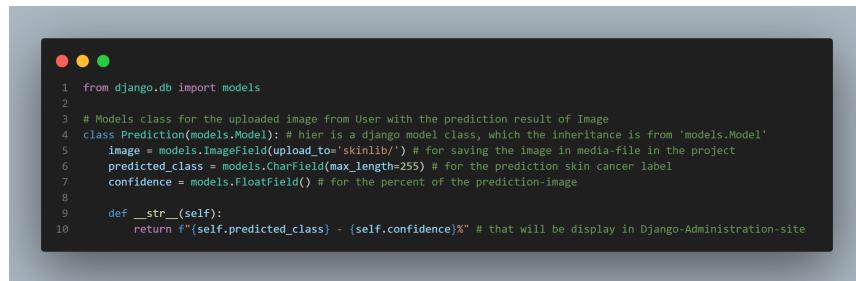
Abbildung 5.24: skinlib Django Visual studio Code

5.3.4 Integration des CNN-Modells in skinlib-Django-Projekt

Um das erstelltes skinlib-CNN-Modell in der oben skinlib-Django-Projekt zu integrieren, muss zuerst das gespeichertes CNN-Modell in h5-Format im Django-Projekt hinzugefügt werden.

Dann die Integration erfolgt in folgenden den 3 Django-Dateien in Projekt:

1. Das Models-Datei wurde nach der Erstellung der Python-Django-App in 5.22 in *src/skinlib/models.py* erstellt. In skinlib-Django-Projekt wurden ein Models-class namens ***Prediction*** in 5.25 zugefügt:



```

● ● ●
1  from django.db import models
2
3  # Models class for the uploaded image from User with the prediction result of Image
4  class Prediction(models.Model): # hier is a django model class, which the inheritance is from 'models.Model'
5      image = models.ImageField(upload_to='skinlib/') # for saving the image in media-file in the project
6      predicted_class = models.CharField(max_length=255) # for the prediction skin cancer label
7      confidence = models.FloatField() # for the percent of the prediction-image
8
9      def __str__(self):
10         return f"{self.predicted_class} - {self.confidence}%" # that will be display in Django-Administration-site

```

Abbildung 5.25: skinlib Django models.py Prediction-class

Hier wurde eine Django-model-Klasse geschrieben, die die Informationen für das von dem Benutzer hochgeladenes Hautbild enthält. Diese ***Prediction-Klasse*** repräsentiert und bildet die Komponente von der Vorhersage (***predicted_class***) des skinlib-CNN-Modells und den Prozentsatz (***confidence***) der Vorhersage, die zu dem hochgeladenes Bild (***image***) gemacht wurde.

2. Da das Django-model direkt mit dem Django-views in Verbindung, siehe Abbildung 5.20, ist, wurde dann eine Funktion in *src/skinlib/views.py* in 5.26 gebildet, die die Verantwortung für die Bildverarbeitung des hochgeladenes Bildes und die Generierung der Vorhersage trägt:
 - Es wurden zuerst die Konstanten wie der Pfad zum CNN-Modell und die Klassennamen entsprechend die Hautveränderungsarten im Datensatz definiert.
 - Danach wurde das CNN-Modell von ihrem Pfad mittels **CUSTOM_MODEL = load_model(MODEL_PATH)** geladen.
 - Es wurde eine View-Funktion **predict** definiert, die die Vorhersagen durchführt.

```

1 # Definieren des CNN_Modellpfads. skinlib-CNN-Modells wird in der src/cnnmodell gespeichert.
2 BASE_DIR = Path(settings.BASE_DIR)
3 MODEL_PATH = str(BASE_DIR / 'cnnmodell' / 'skin_cancer_model_multible_classifier.h5')
4 # Definieren der Hautkrebsklassennamen
5 CLASS_NAMES = [
6     'Actinic keratosis', 'Basal cell carcinoma', 'Benign keratosis', 'Dermatofibroma',
7     'Melanocytic nevus', 'Melanoma', 'Squamous cell carcinoma', 'Vascular lesion'
8 ]
9
10 # Laden des CNN-Modells
11 CUSTOM_MODEL = load_model(MODEL_PATH)
12 # predict-Funktion wird aufgerufen, wenn ein Benutzer eine Vorhersage für sein hochgeladenes Hautbild machen möchte.
13
14 def predict(request):
15     if request.method == 'POST' and 'imagefile' in request.FILES:
16         try:
17             # Hochgeladenes Bild wird erhalten und gespeichert
18             imagefile = request.FILES['imagefile']
19             fs = FileSystemStorage()
20             filename = fs.save(os.path.join(settings.MEDIA_ROOT, 'images', imagefile.name), imagefile)
21             image_path = os.path.join(settings.MEDIA_ROOT, 'images', imagefile.name)
22             # Bild wird geladen und vorverarbeitet, um es für das Modell vorzubereiten
23             target_size = (64, 64)
24             image = load_img(image_path, target_size=target_size)
25             image = img_to_array(image)
26             image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
27             image = preprocess_input(image)
28             # Vorhersage mit dem geladenen Modell wird durchgeführt
29             predictions = CUSTOM_MODEL.predict(image)
30             class_probabilities = predictions[0]
31             class_index = np.argmax(class_probabilities)
32             predicted_class_name = CLASS_NAMES[class_index]
33             # Die Vorhersage wird in der Datenbank gespeichert
34             prediction_obj = Prediction(
35                 image=imagefile,
36                 predicted_class=predicted_class_name,
37                 confidence=class_probabilities[class_index] * 100
38             )
39             prediction_obj.save()
40             # Das Ergebnis wird dem Benutzer angezeigt
41             return render(request, 'skinlib/predict.html', {'prediction_obj': prediction_obj})
42         except Exception as e:
43             # Fehlerbehandlung, falls ein Fehler auftritt
44             return render(request, 'skinlib/predict.html', {'error': f'Error processing image: {str(e)}'})
45     return render(request, 'skinlib/predict.html', {'error': 'No file part'})
```

Abbildung 5.26: skinlib Django views.py predict-Funktion

- Diese Funktion wird nur dann in der Zeile in 5.26 aufgerufen, wenn ein POST-Request mit einem Bild hochgeladen wird
- Das hochgeladene Bild wird dann in *src/media/images* gespeichert und vorverarbeitet, wobei das Bildgröße auf **64x64** skaliert wurde, damit wird es mit den Anforderungen des CNN-Modells übereinstimmt.
- Das Bild wird mit TensorFlow **image = load_img(image_path, target_size=target_size)** geladen und in ein Numpy-Array **image = img_to_array(image)** konvertiert. Damit wird das Bild in die richtige Form **Numpay-Array** für das Modell gebracht und vorverarbeitet.

- **(1, image.shape[0], image.shape[1], image.shape[2]):** Diese Zeile im Code ändert die Form des Bildes, um es in das Modell einzuspeisen. Das Modell erwartet Eingaben in Form von (Batches). Das Bild wird also in eine Batch-Form umgewandelt, indem eine zusätzliche Dimension am Anfang hinzugefügt wird. Hier wird `image.shape` verwendet, um die Dimensionen des Bildes zu erhalten, und dann wird `reshape` verwendet, um die Form entsprechend anzupassen.
Das Bild wird zu einem Batch mit einer Größe von 1 umgeformt, wobei die anderen Dimensionen die Höhe, Breite und die Anzahl der Farbkanäle des Bildes sind.
- Das vorverarbeitete Bild wird an das geladene CNN-Modell mit `predictions = CUSTOM_MODEL.predict(image)` übergeben und werden dann mit `class_probabilities = predictions[0]` die Vorhersagen für die Klassenwahrscheinlichkeiten erhalten.
- Die Klasse mit der höchsten Wahrscheinlichkeit mittels `np.argmax` in der Zeile 31 in 5.26 wird aus den Vorhersagen ausgewählt und der entsprechende Klassenname wird ermittelt.
- Zunächst wurde ein Python-Objekt namens `prediction_obj` in der Zeile 34 in 5.26 hinzugefügt, in dem das erstellten Django-model-Klasse `Prediction` in 5.25 aufgerufen wurde. Hier wurden dann die Prediction-Informationen, die in den vorherigen Schritten durchgeführt wurden, im erstellten Django-model-Klasse `Prediction` eingespeist, wobei dieses Objekt mit `prediction_obj.save()` in der Datenbank in Django gespeichert wird. Diese Python-Objekt `prediction_obj` wurde dann später in der `predict.html` benutzt.
- Schließlich wird die Vorhersage an das `predict.html`-Template übergeben, um das Ergebnis anzuzeigen.
Wenn ein Fehler auftritt (z. B. ein Problem bei der Verarbeitung des Bildes), wird eine Fehlermeldung an das Template übergeben.

3. In der Django-App **src/skinlib** wurde ein Ordner namens **templates** erstellt, in der noch ein Ordner namens der Django-App **skinlib** erzeugt wurde. In diesem Ordner wurde die HTML-Dateien zugefügt.

In Skinlib-Webseite wurde 6 HTML-Dateien geschrieben, welche die Webseite repräsentieren und die Eingang zu dem skinlib-CNN-Modell enthält.

- a) **index.html**: Diese Html-Datei repräsentiert die Hauptseite von Skinlib, in der die Benutzer den Möglichkeit zum Probieren des CNN-Modells und mehr Informationen über die Funktion des Modells haben.
- b) **predict.html**: Diese Html-Datei repräsentiert die Seite von dem Skinlib-CNN-Modells, in der Benutzer ihre Hautbildern hochladen können und das CNN-Modell probieren, um die Vorhersage von dem hochgeladenes Bild zu erhalten.

In diesem Html-Datei wurde dann ein Html-Form, womit die Benutzer auf ihren aufgenommenen Hautbildern zugreifen können, aufgebaut. Als Ergebnis von dem Html-Form-Code sind in 5.27 Bildfeld zu Auswahl des Hautbildes und die Taste zum Erhalten der Vorhersage vom hochgeladenes zu sehen.

Hautbild hochladen: Datei auswählen Keine Datei ausgewählt

Vorhersage erhalten

Abbildung 5.27: skinlib Django predict.html Image-Form

In diesem Html-Form wurde das Python-Objekt **prediction_obj** aufgerufen, um ihre Komponenten wie **image**, **predicted_class** und **confidence** mit den Html-Form-Komponenten, die Daten von den Benutzern erhalten, integriert.

Im **ISIC**-Datensatz sind vier verschiedene Arten (Keine Hautkrebsarten) von gutartigen Hautveränderungen wie Muttermale oder Leberflecken und vier Arten von Hautkrebs enthalten. Es fehlt jedoch eine zusätzliche Kategorie, die Informationen darüber enthält, ob ein Bild kein Hautbild ist oder keine Hautläsionen aufweist. Dies ist wichtig, um sicherzustellen, dass Vorfahrtsgesetze nur für die acht entsprechenden Klassen eingeschränkt werden, wenn Benutzer ein falsches Bild hochladen oder ein Bild ohne Hautläsionen hochladen.

- c) **dashboard.html**: Diese Html-Datei repräsentiert die Datenstatistik, die die Anzahl von den Skinlib-Besuchern anzeigt
- d) **answer.html**: Diese Html-Datei gibt den Benutzern die Bestätigung, dass sie ihre Daten vollständig hochgeladen haben.

6 Ergebnisse

In den vorherigen Schritten wurde dann eine Webseite aufgebaut, die Skinlib genannt wurde.

In diesem Webseite wurde das Frontend mit Html, Bootstrap, CSS und Javascript aufgebaut. Das Backend wurde mithilfe Python-Django Framework erstellt. Das deep learning CNN-Modell wurde mit Tensorflow und Keras für die Hautbilder-klassifikation gebildet, welches anschließend mittels Tensorflow-Bibliotheken in h5-Format **skin_cancer_model_multible_classifier.h5** geladen wurde. Das geladenes CNN-Modell wurde dann im Django-Projekt integriert, um die Benutzern ihre Hautbildern hochladen zu können und die entsprechende Vorhersage in einer Webseite zu erhalten.

Um die Dateien und die Quellcodes des Skinlib-Projektes zu sehen, wurde ein lokalen Git-Repository für das gesamten Skinlib-Projekt in erstellt.

Git ist ein verteiltes Versionskontrollsysteem (VCS), das die Verfolgung von Änderungen in Dateien und Projekten ermöglicht. Mit Git können Entwickler unterschiedliche Versionen ihres Codes verwalten, Branches erstellen, Änderungen verfolgen und zusammenführen, um Kollaboration und parallele Entwicklung zu ermöglichen.

Danach wurde online ein Github-Repository namens **Deep_learning_Skincancerlib** erstellt, in dem das lokale Git-Repository mittels Git-Befehle online in Github ge-hostet wurde.

Github ist eine webbasierte Plattform, die auf Git basiert. Es bietet zusätzliche Funktionen und Dienste für die Zusammenarbeit an Softwareprojekten. Github ermöglicht es Entwicklern, ihre Git-Repositories online zu hosten.

In meinem Github-account hier nach dem Klicken auf unten zugefügtem Link finden Sie das Projekt-Repository und alle Details mit den erklärenden Kommentaren, die dabei helfen könnten.

Deep_learning_Skincancerlib

7 Programmcode

Das CNN-Modell-Notebook sowie der gesamten Skinlib-Projektcode ist in Github freigegeben. Unter dem Github-Repository **Deep_learning_Skincancerlib** findet man alle Commits und Dateien 7.1, die schon während der Erstellung von Skinlib-Webseite mittels Git und Github hochgeladen wurden.

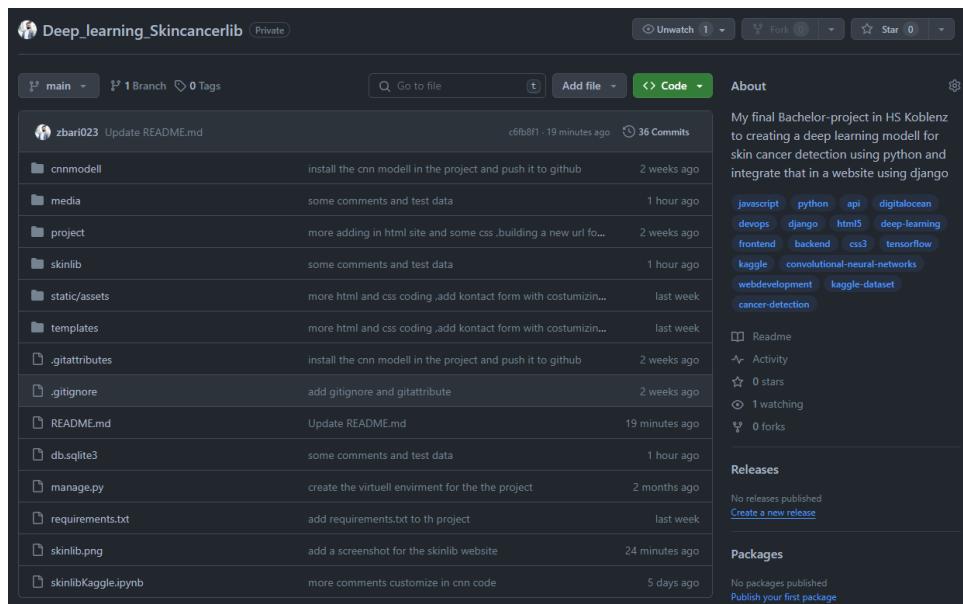


Abbildung 7.1: skinlib Github-repo

Literatur

- [1] Machine-Learning , *Michaela Tiedemann*, URL:<https://www.alexanderthamm.com/de/blog/machine-learning-fuer-den-stationaeren-handel/>, 18.April.2019
- [2] HEICON, *Dipl.-Ing. (FH) Martin Heininger*, URL:<https://heicon-ulm.de/en/testing-ai-systems-possible-or-impossible/>, 14.MAY.2021
- [3] Neurons and the Brain, *Yusu Pan*, URL:<https://www.yython.com/post/tutorials/coursera-machine-learning-week-4/>, 15.Aug.2016
- [4] ResearchGate GmbH, *Chen Wang*, URL:https://www.researchgate.net/publication/326470663_Current_Strategies_and_Applications_for_Precision_Drug_Design#pf8, 01.Jul.2018
- [5] Analytics Yogi, Ajitesh Kumar, URL:<https://vitalflux.com/real-world-applications-of-convolutional-neural-networks/>, 06.Nov.2021
- [6] Data Science, *Bienvenue chez*, URL:<https://datascientest.com/de/convolutional-neural-network-2>, 17.Mai.2023
- [7] The Dropout Layer, *Baeldung*, URL:<https://www.baeldung.com/cs/ml-relu-dropout-layers>, 14.April.2023
- [8] Github-repo von Konvolusi, *Konvolusi*, URL:<https://github.com/irfnrdh/Konvolusi>, 2019
- [9] Maschinelles Lernen, *Prof. Dr. Jens Bongartz*, URL:https://www.youtube.com/watch?v=1ZTdw6wcJjQ&list=PLR0kZwSqSXx2Bf6GPuGkDri_zGyzN2cRa&index=4, 01.Mar.2021
- [10] Deep Learning, *Ekaba Bisong*, URL:https://ekababisong.org/ieee-ompi-workshop/deep_learning/

- [11] Analytics Vidhya , *Shipra Saxena*, URL:<https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/>, 26.Oct.2023
- [12] The International Skin Imaging Collaboration , *Challenge 2019*, URL:<https://api.isic-archive.com/collections/65/>, 2019
- [13] Skin Cancer Dataset from ISIC - 2019 , *BHANU PRASANNA*, URL:<https://www.kaggle.com/datasets/bhanuprasanna/isic-2019?rvi=1>, 2022
- [14] Django Tutorial , *Doprax*, URL:<https://www.doprax.com/tutorial/django-tutorial-for-beginners-part-1/>, 2018
- [15] Melanom , *Novartis*, URL:<https://www.youtube.com/watch?v=dIAqy1scfII&t=60s>, 08.Jan.2021
- [16] GitHub , *Ziad Bari* , *Binary_Classification_Skinlib*, URL:https://github.com/zbari023/Binary_Classification_Skinlib, 2023
- [17] GitHub , *Ziad Bari* , *Deep_learning_Skincancerlib*, URL:https://github.com/zbari023/Deep_learning_Skincancerlib, 2023
- [18] Google Research , *Brain Team*, *Mingxing Tan*, URL:<https://arxiv.org/pdf/1905.11946.pdf>, 2019
- [19] Keras , *EfficientNetB5*, *Mingxing Tan*, URL:<https://keras.io/api/applications/efficientnet/#efficientnetb5-function>, 28.May.2019
- [20] Papers With Code , *Image Classification on ImageNet* , *Mingxing Tan*, URL:<https://blog.csdn.net/gongdiwudu/article/details/132117579>, 08.Aug.2023
- [21] Medium , *Transfer learning* , *Pratik Bhavsar*, URL:<https://medium.com/modern-nlp/3-ways-to-make-new-language-models-f3642e3a4816>, 27.Mar.2020

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

Zutreffendes bitte ankreuzen:

Mit der hochschulinternen Veröffentlichung der Arbeit bin ich:

einverstanden [] n i c h t einverstanden

Ich bin damit einverstanden, dass

- der Titel meiner Arbeit mit meinem Name und denen der Betreuer in Bibliothekskatalog (OPAC) veröffentlicht wird und
- die Arbeit als PDF hochschulintern eingesehen werden kann.

Dafür erhält die Hochschule ein einfaches, nicht übertragbares Nutzungsrecht ausschließlich für den Zweck der Veröffentlichung in der Bibliothek. Das Recht der Veröffentlichung oder Verwertung durch den Verfasser oder die Verfasserin auf andere Weise, z.B. über einen Verlag, bleibt davon unberührt. Die Hochschule ist nicht verpflichtet, die Arbeit zu veröffentlichen. Das Einverständnis kann jederzeit schriftlich (per Brief!) widerrufen werden. Die Bibliothek wird die Arbeit dann unverzüglich aus dem OPAC oder der Auslage entfernen. Die Veröffentlichung hängt außerdem von der später erteilten Zustimmung des Erstbetreuers ab.

Remagen, 19.02.2024

Ort, Datum



Unterschrift