

Aus der Klinik und Poliklinik für Dermatologie und Venerologie
der Universität zu Köln
Direktorin: Universitätsprofessorin Dr. med. E. von Stebut-Borschitz

**Tumordetektion und Vorhersage eines
Progresses anhand histologischer Schnitte
kutaner Plattenepithelkarzinome mithilfe eines
neuronalen Netzes**

Dissertation zur Erlangung der Doktorwürde
der Medizinischen Fakultät
der Universität zu Köln

vorgelegt von
Corinna Bürger
aus Dortmund

promoviert am 22. August 2023

Gedruckt mit Genehmigung der Medizinischen Fakultät der Universität zu Köln
2023

Dekan: Universitätsprofessor Dr. med. G. R. Fink
1. Gutachterin: Privatdozentin Dr. med. D. Helbig
2. Gutachterin: Universitätsprofessor Dr. rer. nat. K. Božek

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Dissertationsschrift ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskriptes habe ich keine Unterstützungsleistungen erhalten.

Weitere Personen waren an der Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich nicht die Hilfe einer Promotionsberaterin/eines Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertationsschrift stehen.

Die Dissertationsschrift wurde von mir bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Die Fragestellung wurde von Frau PD Dr. Doris Helbig, Herrn Dr. Johannes Brägelmann und mir gemeinsam formuliert.

Der dieser Arbeit zugrunde liegende Datensatz (Klinische Daten und histologische Präparate) wurde in der Klinik und Poliklinik für Dermatologie und Venerologie der Uniklinik Köln erstellt und von Frau Dr. Helbig zur Verfügung gestellt.

Die für diese Arbeit verwendeten histologischen Präparate wurden von Herrn Dr. Brägelmann eingescannt.

Die Tumor-Annotationen der gescannten Slides erfolgten durch mich und wurden von Frau Dr. Helbig überprüft.

Die Präprozessierung der Bilder (Zerteilung der Slides in Tiles, Herausfiltern des Hintergrundes, Auswahl relevanter Bildbereiche, Erstellung der Trainings- und Testdaten) sowie Training und Evaluation der neuronalen Netze zur Tumordetektion und Progressprädiktion erfolgten durch mich. Alle dafür verwendeten Python-Skripte wurden von mir selbst geschrieben und ausgeführt. Das beinhaltet die Erstellung jeglicher Abbildungen.


Die Skripte 3, 4, 5, 9, 10, 11 (s. Punkt 7.3) wurden auf dem HPC-Cluster des regionalen Rechenzentrums der Universität zu Köln (RRZK) ausgeführt.

Die statistische Auswertung der klinischen Parameter der Kohorte inklusive Erstellung eines Modells zur Prädiktion eines Progresses anhand klinischer Risikofaktoren erfolgte durch mich. Alle dafür notwendigen Python-Skripte wurden von mir selbst geschrieben und ausgeführt. Das beinhaltet die Erstellung jeglicher Abbildungen.

Erklärung zur guten wissenschaftlichen Praxis:

Ich erkläre hiermit, dass ich die Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten (Amtliche Mitteilung der Universität zu Köln AM 132/2020) der Universität zu Köln gelesen habe und verpflichte mich hiermit, die dort genannten Vorgaben bei allen wissenschaftlichen Tätigkeiten zu beachten und umzusetzen.

Köln, den 15.05.2023

Unterschrift: 

Danksagung

An dieser Stelle möchte ich allen beteiligten Personen danken, die mich während der gesamten Durchführung meiner Promotion unterstützt haben.

Besonders danken möchte ich dabei meiner Doktormutter PD Dr. Doris Helbig sowie meinem Betreuer Dr. Johannes Brägelmann für die enge Zusammenarbeit, den regen fachlichen Austausch und die ausgezeichnete Betreuung.

Außerdem gilt mein Dank meinen Eltern, die mich von Beginn des Studiums bis hin zur Promotion emotional und finanziell unterstützt haben. Einen großen Dank möchte ich zudem meinem Partner Daniel Schütte aussprechen, der mir während der Durchführung meiner Promotion stets motivierend zur Seite stand.

Des Weiteren möchte ich meinen Dank der Klinik für Dermatologie sowie dem pathologischen Institut der Universitätsklinik Köln aussprechen, die durch Bereitstellung der Daten sowie notwendigen Ressourcen diese Arbeit ermöglicht haben. Nicht zuletzt möchte ich mich bei dem regionalen Rechenzentrum Köln (RRZK) für die Ermöglichung der Nutzung des HPC-Clusters sowie den technischen Support bedanken.

Inhaltsverzeichnis

ABKÜRZUNGSVERZEICHNIS.....	8
1. ZUSAMMENFASSUNG.....	9
2. EINLEITUNG.....	11
2.1 Kutanes Plattenepithelkarzinom.....	11
2.1.1. Epidemiologie des kutanen Plattenepithelkarzinoms.....	11
2.1.2. Therapie des kutanen Plattenepithelkarzinoms.....	11
2.1.3. Risikostratifizierung für Lokalrezidive und Metastasen.....	12
2.1.4. Nachsorge.....	13
2.2. Neuronale Netze.....	13
2.2.1. Convolutional Neural Network.....	14
2.2.2. Trainingsprozess und Trainingsparameter.....	16
2.2.3. Transferlernen.....	17
2.2.4. Neuronale Netze in der Histopathologie.....	18
2.2.5. Präprozessierung.....	18
2.3. Fragestellungen und Ziel der Arbeit.....	19
3. MATERIAL UND METHODEN.....	21
3.1. Material.....	21
3.1.1. Kohorte.....	21
3.1.2. QuPath.....	21
3.1.3. Python und verwendete Bibliotheken.....	21
3.1.4. Rechencluster.....	22
3.2. Methoden.....	22
3.2.1. Detektion des Tumorbereichs.....	23
(1) Präprozessierung.....	23
(2) Training.....	24
(3) Testen.....	25
(4) Klassifikation ganzer Slides.....	25
3.2.2. Vorhersage von Progressen.....	26
(1) Präprozessierung.....	26
(2) Training.....	27

(3) Testen.....	27
(4) Klassifikation ganzer Slides.....	28
3.2.3. Vorhersage von Progressen anhand klinischer Merkmale.....	28
4. ERGEBNISSE.....	29
4.1. Detektion des Tumorbereichs.....	29
4.1.1. Klassifikation der Tiles durch CNN_{Tumor}	31
4.1.2. Klassifikation ganzer Slides durch CNN_{Tumor}	33
4.2. Vorhersage von Progressen.....	35
4.2.1. Deskriptive Statistik der Trainings- und Testkohorten.....	35
4.2.2. Klassifikation der Tiles durch $CNN_{Progress}$	37
4.2.3. Klassifikation ganzer Slides durch $CNN_{Progress}$	41
4.2.4. Klinische Merkmale als Prädiktoren für einen Progress.....	42
5. DISKUSSION.....	45
5.1. Sichtbare histomorphologische Merkmale ermöglichen eine zuverlässige Erkennung von Gewebe durch ein neuronales Netz.....	45
5.2. Vorhersage von Progressen anhand von klinischen Risikofaktoren.....	47
5.3. Anwendbarkeit von neuronalen Netzen zur automatisierten Analyse von Slides im klinischen Alltag.....	48
6. LITERATURVERZEICHNIS.....	50
7. ANHANG.....	54
7.1. Abbildungsverzeichnis.....	54
7.2. Tabellenverzeichnis.....	56
7.3. Skripte.....	57
7.3.1. Skript 1: Zerteilen der annotierten Slides und Exportieren der Tiles aus QuPath.....	57
7.3.2. Skript 2: Sortieren der Annotationen.....	57
7.3.3. Skript 3: Erstellen der Trainingsdaten für CNN_{Tumor}	58
7.3.4. Skript 4: Trainieren von CNN_{Tumor}	59
7.3.5. Skript 5: Evaluation von CNN_{Tumor}	61
7.3.6. Skript 6: Zerteilen der nicht-annotierten Slides in Tiles.....	63
7.3.7. Skript 7: Herausfiltern des Hintergrunds.....	63

7.3.8. Skript 8: Markieren der Tumor-Tiles.....	63
7.3.9. Skript 9: Erstellen der Trainingsdaten für CNN _{Progress}	64
7.3.10. Skript 10: Trainieren von CNN _{Progress}	66
7.3.11. Skript 11: Evaluation von CNN _{Progress}	67
7.3.12. Skript 12: Markieren der als „Progress“ eingestuften Tiles.....	70
7.3.13. Skript 13: Testen der Risikofaktoren auf Signifikanz und Logistische Regression.....	71
8. VORABVERÖFFENTLICHUNGEN VON ERGEBNISSEN.....	73

Abkürzungsverzeichnis

5-JÜR	5-Jahres-Überlebensrate
AUC	Area Under the Curve
CL	Convolutional Layer
CNN	Convolutional Neural Network
cSCC	Cutaneous Squamous Cell Carcinoma (<i>deutsch: kutanes Plattenepithelkarzinom</i>)
DL	Dropout Layer
FCL	Fully-Connected Layer
GB	Gigabyte
LR	Lernrate
Max	Maximum
Min	Minimum
NPW	Negativer prädiktiver Wert
OR	Odds Ratio
PL	Pooling Layer
PPW	Positiver prädiktiver Wert
ROC	Receiver Operating Characteristic
RR	Relatives Risiko
RRZK	Regionales Rechenzentrum der Universität zu Köln
WSI	Whole-Slide-Image

1. Zusammenfassung

Kutane Plattenepithelkarzinome (englisch: *Cutaneous squamous cell carcinoma*, „cSCC“) gehören in Deutschland zu den häufigsten Karzinomen. Während die meisten davon erfreulicherweise eine gute Prognose aufweisen, sinkt diese drastisch, sobald ein Progress, entweder in Form eines Lokalrezidivs oder einer Fernmetastase, auftritt. Bisher erfolgt die Risikostratifizierung bezüglich des Auftretens eines Progresses anhand klinischer Risikofaktoren. Diese Methode der Risikobewertung ist jedoch häufig noch nicht ausreichend, um einen Progress zuverlässig vorherzusehen.

Das Ziel der vorliegenden Arbeit war die Entwicklung eines neuronalen Netzes, welches anhand histologischer Schnitte von cSCC eine Vorhersage trifft, ob der betroffene Patient einen Progress entwickeln wird oder nicht. Der Begriff „Progress“ beinhaltet hierbei sowohl Lokalrezidive als auch Fernmetastasen. Zu diesem Zweck wurde mittels Transferlernens ein Convolutional Neural Network (CNN_{Progress}) an insgesamt 28250 Bildausschnitten, sogenannten „Tiles“, von jeweils 42 histologischen Schnitten exzidiert cSCC-Primarien mit und ohne dokumentiertem Auftreten eines Progresses trainiert. Zuvor erfolgte die Entfernung von Tiles ohne Gewebe sowie jener Tiles, auf denen ausschließlich physiologisches Gewebe oder nicht ausreichend Tumor abgebildet war. Die Tumorerkennung auf den Tiles erfolgte dabei automatisiert mithilfe eines weiteren neuronalen Netzes (CNN_{Tumor}), welches anhand von 12527 selbst-annotierten Tiles trainiert wurde. Das CNN_{Tumor} erreichte in einem Testset auf der Ebene der Tiles ($n = 15429$) eine hohe Sensitivität (92 %) und Spezifität (89 %). Die Erstellung einer *Receiver Operating Characteristic* (ROC)-Kurve ergab eine *Area Under the Curve* (AUC) von 0.96. Bezogen auf die Klassifikation ganzer histologischer Schnitte, sogenannter „Slides“, und in Anbetracht des Zustandes, dass sich falsch-positiv klassifizierte Tiles nicht gänzlich vermeiden lassen, ließ sich errechnen, dass bei unserem trainierten CNN_{Tumor} ab einem Anteil von 21,7 % klassifizierter „Tumor“-Tiles mit dem tatsächlichen Vorliegen eines Tumors auf dem Slide zu rechnen ist. Mit diesem Schwellenwert wurden alle 11 getesteten „Tumor“-Slides als richtig-positiv und 9/10 Normalhaut-Slides (90 %) als richtig-negativ klassifiziert. Das CNN_{Progress} erreichte auf der Ebene der Tiles einen negativen prädiktiven Wert (NPW) von 70 % bei eher geringer Sensitivität (61 %) und Spezifität (63 %). Bezogen auf die Klassifikation ganzer Slides wurde errechnet, dass ab einem Anteil von 55,4 % als „progredient“ klassifizierter Tiles bei dem jeweiligen Patienten von der Entwicklung eines Progresses auszugehen ist. Damit wurden 10/12 Slides (83 %) als richtig-negativ klassifiziert.

Zudem untersuchten wir, inwiefern sowohl klinische Merkmale als auch Merkmale des Tumors mit einem erhöhten Risiko für das Auftreten eines Progresses korrelieren. Dabei konnte eine signifikante Erhöhung des Risikos bei Vorliegen einer Perineuralscheideninvasion beobachtet werden. Zudem wiesen die cSCC, bei denen im Verlauf ein Progress auftrat, eine signifikant größere Tumordicke auf im Vergleich zu den cSCC, die sich nicht progredient zeigten.

Insgesamt gelang durch das CNN_{Tumor} eine Tumordetektion mit hoher Sensitivität und Spezifität, während bei der Entwicklung des CNN_{Progress} noch Optimierungsbedarf besteht. Wir schlussfolgerten, dass die Ursache hierfür am ehesten in den im Vergleich zur Tumordetektion weniger ausgeprägten histomorphologischen Unterschieden zwischen Tiles von Patienten, die einen Progress entwickelten und denjenigen, bei denen kein Progress auftrat, begründet liegt. Obwohl neuronale Netze prinzipiell in der Lage sind, Bildmerkmale zu extrahieren, die dem menschlichen Auge verborgen bleiben, konnten solche Merkmale durch unser CNN_{Progress} weniger herausgelesen werden. Die für Dermatologen prominenten Unterschiede zwischen Normalhaut- und Tumorgewebe sind von unserem CNN_{Tumor} zuverlässig gelernt worden, sodass methodische Fehler zumindest unwahrscheinlich erscheinen. Eine Möglichkeit zur Verbesserung der Vorhersagekraft wäre, die histologischen Bilder im Ganzen oder zumindest in größeren Bildausschnitten zu verarbeiten, um so prognoserelevante Informationen, wie beispielsweise die Tumordicke oder die Eindringtiefe, erhalten zu können. Die Verwendung einer größeren Kohorte mit einer höheren Anzahl an Progressen könnte ebenso zu einer höheren Sensitivität und Spezifität führen. Auch ein multifaktorieller Ansatz könnte hilfreich sein, in dem klinische Risikofaktoren ebenso wie das histologische Bild von Patienten gemeinsam von einem neuronalen Netz verarbeitet werden.

Zusammenfassend konnten wir am Beispiel des cSCC zeigen, dass Methoden der künstlichen Intelligenz für die Interpretation histologischer Schnitte von großem Wert sein können.

2. Einleitung

2.1 Kutanes Plattenepithelkarzinom

2.1.1. Epidemiologie des kutanen Plattenepithelkarzinoms

2018 wurden in Deutschland 94200 bzw. 105230 neue Fälle von nicht-melanotischem Hautkrebs bei Frauen bzw. Männern diagnostiziert. Fast ein Viertel davon waren cSCC, womit diese bei beiden Geschlechtern fast so häufig auftraten wie Mamma- bzw. Prostatakarzinome¹. Die Inzidenz ist dabei über die letzten Jahrzehnte steigend, was vor allem an der immer älter werdenden Bevölkerung, aber auch an vermehrter Erkennung durch das Hautkrebsscreenings, liegt¹.

Während die allgemeine 5-Jahres-Überlebensrate (5-JÜR) mit 97-100 % sehr hoch ist¹, reduziert sich diese deutlich, sobald ein Progress eintritt. Bei Auftreten von Lokalrezidiven beträgt die 5-JÜR 44 % und nur noch 24 % bei Vorliegen von Fernmetastasen².

2.1.2. Therapie des kutanen Plattenepithelkarzinoms

Der Goldstandard in der Therapie des primär nicht-metastasierten cSCC ist die Exzision des Tumors mit anschließender histologischer Begutachtung des Exzidats. Dabei spielt insbesondere die Frage nach tumorfreien Resektionsrändern im Sinne einer R0-Resektion eine Rolle, da nur so von einer vollständigen Exzision ausgegangen werden kann³. Der Nutzen einer Sentinel-Lymphknoten-Biopsie ist nicht ausreichend bewiesen und wird nicht standardmäßig empfohlen³. Regionäre Lymphknoten sollen nur bei klinisch manifester Lymphknotenmetastase entfernt werden³.

Eine postoperative Bestrahlung ist nicht standardmäßig erforderlich, sondern wird nur bei erhöhtem Risiko für einen Progress durchgeführt. Während sie bei R1-Resektion, einem Resektionsrand < 2 mm ohne Möglichkeit der Nachresektion, Perineuralscheideninvasion sowie ausgedehntem bzw. parotidealem Lymphknotenbefall zu einem Überlebensvorteil zu führen scheint, ist die Datenlage für ihren Nutzen bei Vorliegen anderer Risikofaktoren, welche im nächsten Abschnitt diskutiert werden, nicht ganz eindeutig³.

Lokalrezidive werden, wie auch der Primärtumor, wenn möglich reseziert. Bei erhöhtem Risiko für einen erneuten Progress, wird eine adjuvante Radiotherapie empfohlen³.

Für die Therapie von fernmetastasierten cSCC war die Evidenz bei fehlenden kontrollierten Studien lange Zeit gering und die Auswahl einer geeigneten systemischen Therapie eine Einzelfallentscheidung, die in einem Tumorboard diskutiert wurde⁴. 2019 erfolgte dann die Zulassung des PD-1-Inhibitors Cemiplimab⁵ und kurz darauf von Pembrolizumab⁶ zur

Therapie des metastasierten cSCC, welche sich mittlerweile in der Therapie des fortgeschrittenen cSCC als Standard etabliert haben⁷.

2.1.3. Risikostratifizierung für Lokalrezidive und Metastasen

Die Prognose des cSCC ist stark von dem Auftreten eines Progresses in Form von lokalen bzw. Fernmetastasen abhängig². Aus diesem Grund ist es wichtig, Patienten mit einem erhöhten Risiko für einen Progress frühzeitig zu erkennen und somit engmaschiger überwachen bzw. ausgeweiteter therapieren zu können, beispielsweise mit adjuvanter Bestrahlung. Neben den oben genannten Risikofaktoren (R1-Resektion, Resektionsrand < 2 mm ohne Möglichkeit der Nachresektion, Perineuralscheideninvasion, ausgedehnter Lymphknotenbefall³) wurden ein Tumordurchmesser > 20 mm, eine Tumordicke > 6 mm, die Infiltration bis in das Fettgewebe sowie von Lymph- und Blutgefäßen, ein niedriger Differenzierungsgrad, desmoplastisches Wachstum und Immunsuppression in diversen Studien als Risikofaktoren identifiziert⁸⁻¹⁰. Die Lokalisation des Tumors scheint ebenfalls Einfluss auf das Risiko für das Auftreten eines Progresses zu haben. So ist das Auftreten von cSCC am Ohr sowie am Unterlippenrot mit einem höheren Risiko für die Entwicklung eines Progresses verbunden^{9,11}. Tabelle 1 gibt einen Überblick über die beschriebenen Risikofaktoren. Auch wenn somit einige Risikofaktoren bereits identifiziert wurden, gelingt es dennoch noch nicht anhand dieser eine suffiziente Aussage bezüglich der Wahrscheinlichkeit eines Progresses zu tätigen. Zudem kommen die (größtenteils retrospektiven) Studien zu teilweise unterschiedlichen Ergebnissen. Weitere zuverlässige Prädiktoren sind notwendig zur Entscheidung, welche Patienten von einer ausgeweiteten Therapie sowie einer engmaschigeren Nachsorge profitieren.

	Risikofaktoren	Quelle(n)
Eigenschaften des Tumors	Tumordurchmesser > 20 mm Tumordicke > 6 mm Niedriger Differenzierungsgrad Desmoplastisches Wachstum Tumorlokalisierung Perineuralscheideninvasion Lymphatische Invasion Vaskuläre Invasion Infiltration des Fettgewebes Ausgedehnter Lymphknotenbefall	Brantsch et al. ⁸ , Thompson et al. ⁹ , Nuño-González et al. ¹⁰ Mourouzis et al. ¹¹
Eigenschaften des Patienten	Immunsuppression	Brantsch et al. ⁸ , Thompson et al. ⁹ , Nuño-González et al. ¹⁰
Sonstige	R1-Resektion Resektionsrand < 2 mm	Berking et al. ³

Tabelle 1: In der Literatur beschriebene Risikofaktoren für das Auftreten eines Progresses bei cSCC

2.1.4. Nachsorge

Die Intervalle der Nachsorgeuntersuchungen bei R0-resezierten cSCC sind angepasst an das Progressionsrisiko in Abhängigkeit des Vorliegens der o.g. Risikofaktoren⁴. Zudem muss berücksichtigt werden, dass die meisten Progressionen innerhalb der ersten zwei Jahre auftreten. So treten 58 % bzw. 69 % der Lokalrezidive bzw. Metastasen nach einem Jahr, 75 % bzw. 84 % innerhalb von zwei Jahren und 83 % bzw. 91 % innerhalb von drei Jahren auf¹². Bei hohem Risiko beträgt das Nachuntersuchungsintervall somit in den ersten beiden Jahren drei Monate, im 3. - 5. Jahr sechs Monate und im 6. - 10. Jahr zwölf Monate^{3,4}. Bei niedrigem Risiko wird eine Nachsorge in den ersten beiden Jahren alle sechs Monate und ab dem 3. Jahr alle zwölf Monate durchgeführt³. Die Einteilung in die zwei o.g. Risikogruppen korreliert nicht zwingend mit dem Tumorstadium. Das Tumorstadium ergibt sich klassischerweise aus der TNM-Klassifikation. Neben dem N- und M-Stadium ergibt sich das T-Stadium aus dem Tumordurchmesser, der Eindringtiefe, einer Perineuralscheideninvasion sowie Invasion des Achsenskeletts. Weitere o.g. Risikofaktoren, insbesondere eine Immunsuppression, der Differenzierungsgrad oder Lokalisation des Tumors spielen in der TNM-Klassifikation und somit auch für das Tumorstadium keine Rolle³. Eine einheitliche Einteilung in die zwei Risikogruppen liegt nicht vor. Die *EDF* (European Dermatology Forum), *EADO* (European Association of Dermato-Oncology) und *EORTC* (European Organization for Research and Treatment of Cancer) schlagen die Zuordnung eines Patienten in die Hochrisiko-Gruppe beim Erfüllen der folgenden Kriterien vor: Eindringtiefe > 6 mm oder Eindringtiefe von 6 mm mit Vorliegen zwei weiterer Risikofaktoren (Durchmesser > 20 mm, Perineuralscheideninvasion, Invasion über das Fettgewebe hinaus, moderat bis schlecht differenzierter Tumor, Rezidivtumor, Lokalisation an Ohr oder Lippe, Immunsuppression)⁴.

2.2. Neuronale Netze

Bevor im Folgenden auf neuronale Netze eingegangen werden soll, müssen im Vorfeld einige Begrifflichkeiten geklärt werden. Von künstlicher Intelligenz spricht man allgemein bei der Entwicklung von Computersystemen, welche menschliche Intelligenz nachahmen und anhand dessen Probleme lösen oder Entscheidungen treffen¹³.

Maschinelles Lernen ist eine Unterform der künstlichen Intelligenz, bei der Computersysteme auf der Basis von gesammelten und erlernten Daten Entscheidungen treffen. Je mehr Daten dafür zur Verfügung stehen, desto häufiger fallen die Entscheidungen des Systems korrekt aus¹⁴. Ein einfaches Beispiel ist hier die Vorhersage des Körpergewichts von Personen anhand der Kenntnis ihrer Körpergröße.

Deep Learning, welches neuronale Netze beinhaltet, ist eine Unterform des maschinellen Lernens. Hierbei wird die Struktur menschlicher Neuronen imitiert, was die Lösung von komplexen und logischen Problemen ermöglicht, die weit über reine Korrelation weniger Parameter hinausgeht¹⁵. Ein Beispiel dafür ist die Spracherkennung oder die hier genutzte Erkennung von Strukturen auf Bildern.

2.2.1. Convolutional Neural Network

Je nach Fragestellung sind jeweils verschiedene Typen von neuronalen Netzen geeignet. Für die hier benötigte Bilderkennung, genauer gesagt die Erkennung von Gewebearchitektur auf histologischen Slides, eignen sich sogenannte Convolutional Neural Networks (CNN) am besten¹⁶. Diese bestehen aus mehreren Schichten simulierter Neurone, die in der Bilderkennung nacheinander durchlaufen werden. Die ersten Ebenen der Schichten erkennen dabei eher Grundformen wie Ecken, Kanten oder Rundungen. In den hinteren Ebenen werden diese zu komplexeren Strukturen zusammengesetzt bis hin zur Erkennung ganzer Objekte wie beispielsweise Tiere, Gesichter oder histologisches Gewebe¹⁷. Es gibt drei Typen von Schichten: Convolutional Layer (CL), Pooling Layer (PL) und Fully-Connected Layer (FCL). Die Architektur eines solchen CNN ist exemplarisch anhand des von uns trainierten VGG-Netzes in den Abbildungen 1 und 2 veranschaulicht. Einem CL kann entweder ein weiterer CL oder ein PL folgen, was beliebig häufig wiederholt werden kann¹⁸. Je mehr Schichten ein CNN hat, desto komplexer wird es und desto mehr Rechenleistung wird benötigt. Der CL ist das Kernstück von CNN und ermöglicht die Verarbeitung des dreidimensionalen Inputs des Bildes (Höhe x Breite x Farbwerte des Pixels). Dabei wird mithilfe eines sogenannten Kernels immer nur jeweils ein Bereich des Bildes auf einmal betrachtet und mit den Gewichten des Kernels, welche durch das Training angepasst werden, ein Punktprodukt gebildet. Das Ergebnis des Produkts geht dann in eine zweidimensionale Output-Matrix ein, bevor der Kernel den nächsten Abschnitt des Bildes betrachtet und das nächste Punktprodukt für die Output-Matrix entsteht¹⁸. Die Funktion des PL ist die Vereinfachung des Inputs zugunsten einer leichteren Verarbeitung. Dabei wird analog zu dem CL jeweils ein Bereich der zweidimensionalen Input-Matrix betrachtet und daraus entweder der höchste Wert (Max Pooling) oder Durchschnittswert (Average Pooling) in die Output-Matrix überführt. Zwar gehen hierbei Informationen verloren, dennoch wird so eine höhere Effizienz erreicht^{18,19}. Die eindimensionalen FCL bilden den Abschluss, anhand derer schließlich auch die endgültige Klassifikation stattfindet¹⁸. Diese erfolgt, indem für die möglichen Klassen jeweils eine Wahrscheinlichkeit ausgegeben wird, welche sich zu 1,0 ergänzen, sodass die Klasse mit der höchsten Wahrscheinlichkeit dem Input, also dem Bild, zugeteilt wird. Im Falle einer Multi-Label-Klassifikation können anhand einer anderen Aktivierungsfunktion der FCL auch mehrere Labels zutreffen²⁰. Hierbei kann die Summe der

Wahrscheinlichkeiten mehr als 1,0 betragen. Beispielsweise können auf einem Röntgenbild des Thorax sowohl eine Pneumonie als auch ein Emphysem und ein Pleuraerguss zu sehen sein. Im Falle der Tumordetektion existiert jedoch nur eine wahre Zuordnung, weshalb sich hier die Wahrscheinlichkeiten des Outputs des letzten FCL zu 1,0 ergänzen.

Zwischen den FCL können fakultativ sogenannte Dropout-Layers (DL) gesetzt werden, in denen ein gegebener Anteil an Trainingsdaten zufällig ausgeschlossen wird und folglich Variabilität in den Trainingsprozess einbringt. Dies vermindert eine Überanpassung des neuronalen Netzes an die Trainingsdaten und verbessert somit die Übertragbarkeit auf externe Daten²¹.

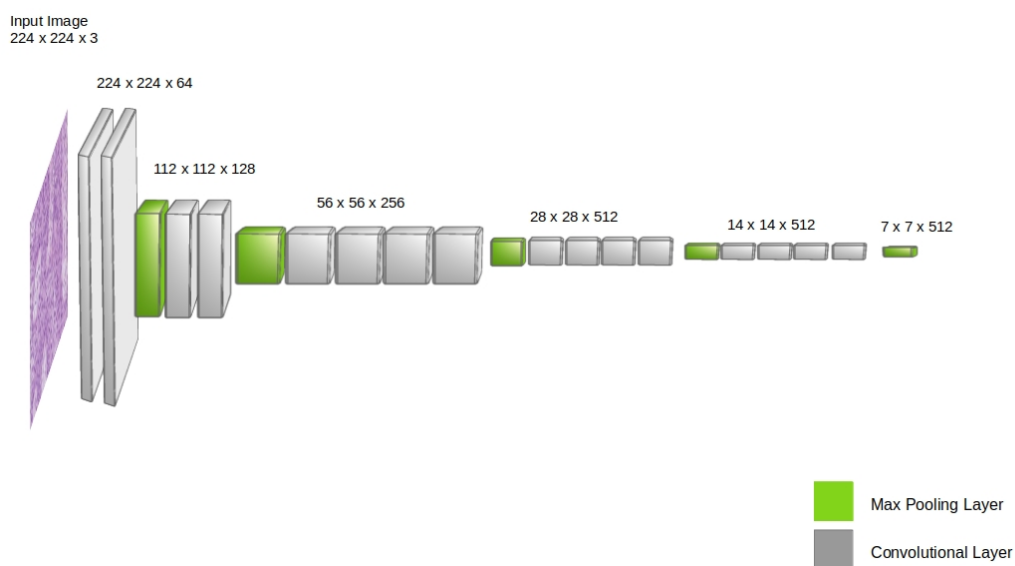


Abbildung 1: Architektur des VGG19-Netzes ohne FCL. Adaptiert von Hasan et al.⁶⁰

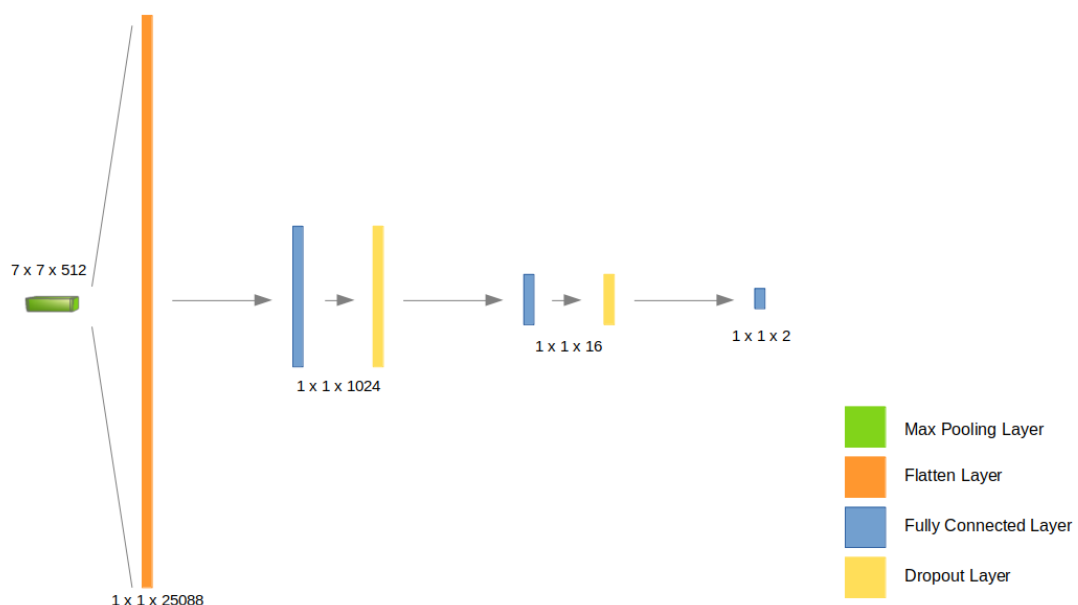


Abbildung 2: Architektur der hinzugefügten, trainierten Schichten von CNN_{Tumor} und $CNN_{Progress}$

2.2.2. Trainingsprozess und Trainingsparameter

Während des Trainings eines neuronalen Netzes werden die im vorherigen Punkt beschriebenen Schichten nacheinander von den Trainingsdaten durchlaufen und die Gewichte des Kernels und der Neuronen durch sogenannte „Backpropagation“²² berechnet und angepasst. Hierbei werden die Gewichte anhand der Abweichung des Outputs von dem wahren Wert rückwärts, also beginnend bei den hinteren Schichten bis hin zu den vorderen Schichten, adaptiert. Die Abweichung wird auch „Loss“ genannt. Der Datensatz durchläuft das Netz jedoch nicht nur einmal, sondern viele Male. Einen Durchlauf des kompletten Datensatzes nennt man Epoche. Die Anzahl der zu durchlaufenden Epochen wird vor dem Training festgelegt²³. Mehr Epochen führen zu einer genaueren Prädiktion, zu viele Epochen jedoch letztlich zu einer Überanpassung an den Trainingsdatensatz.

In der Regel durchlaufen nicht alle Daten gleichzeitig die Schichten des neuronalen Netzes. Dies geschieht vielmehr in sogenannten „Batches“²³. Gängige „Batch“-Größen sind z.B. 32, 64 oder 128. Aber auch der gleichzeitige Durchlauf von nur einem Datenpunkt oder des kompletten Datensatzes ist theoretisch möglich. Die Gewichte des Kernels werden dabei nach jedem fertigen „Batch“ angepasst. Der Vorteil von kleineren „Batches“ ist eine schnellere Anpassung der Gewichte, da weniger Daten berechnet werden müssen. Zudem wird weniger Rechenleistung benötigt. Der Nachteil ist jedoch ein in der Regel ungenauerer und sprunghafterer Trainingsprozess als es mit einem größeren „Batch“ und somit auch mehr verfügbaren Daten zur Anpassung der Gewichte möglich wäre²⁴.

Wie oben bereits beschrieben, wird die Abweichung („Loss“) des Outputs des neuronalen Netzes von dem wahren Wert anhand von „Backpropagation“ berechnet. Das Ziel ist es, einen möglichst geringen „Loss“ zu erreichen. Das Ausmaß, in dem die Gewichte des Kernels und der Neuronen angepasst werden, wird zusätzlich zu dem „Loss“ auch von der Lernrate bestimmt. Die Lernrate ist ein Faktor zwischen 0 und 1, wird vor dem Training fest definiert und mit dem berechneten „Loss“ multipliziert²³. Eine Lernrate von 0,1 bedeutet folglich, dass die Gewichte um 10 % des „Loss“ angepasst werden. Eine größere Lernrate erlaubt schnellere Veränderungen der Gewichte, auch schon bei niedriger Epochenzahl, ist jedoch auch ungenauer und kann sehr sprunghaft sein. Eine kleinere Lernrate kann ein genaueres Training ermöglichen, erfordert jedoch auch längere Trainingszeiten und neigt eher dazu an einem lokalen Minimum des „Loss“ stecken zu bleiben und dadurch das globale Minimum des „Loss“ niemals zu erreichen²⁵.

Ebenfalls vor dem Training angegeben wird der Anteil des Trainingssets, der als Validierungsset genutzt werden soll²³. Das Validierungsset durchläuft am Ende jeder Epoche die Schichten und trägt selbst nicht zum Training bei, sondern dient dem Monitoring des Trainingserfolges während des Trainings. Da die Validierungsdaten keinen Einfluss auf die Gewichte des Netzes haben und sie dem Netz in jeder Epoche erneut wie unbekannt sind,

spricht eine hohe Genauigkeit in den Validierungsdaten für eine gute Übertragbarkeit auf externe Daten¹⁸. Im Gegensatz dazu deutet eine zu hohe Genauigkeit in den Trainingsdaten häufig lediglich auf eine Überanpassung hin. In der Regel wird ein Validierungsanteil von um die 20 % gewählt. Neben dem Validierungsset gibt es auch noch ein sogenanntes Testset. Dieses ist nicht in den Trainingsprozess involviert, sondern dient der Evaluation eines fertig trainierten Netzes (s. Abbildung 3).

Letztlich gibt es keine allgemeingültigen Trainingsparameter, die für alle Datensätze und Fragestellungen gut funktionieren. Mit Berücksichtigung der Vor- und Nachteile der einzelnen Ausprägungen der Parameter (Epochenzahl, „Batch“-Größe, Lernrate) müssen für jeden Trainingsdatensatz die idealen individuellen Trainingsparameter gefunden werden. Dabei beeinflussen sich diese in Bezug auf das Ergebnis auch gegenseitig, sodass dies eine anspruchsvolle und nicht triviale Aufgabe darstellt.

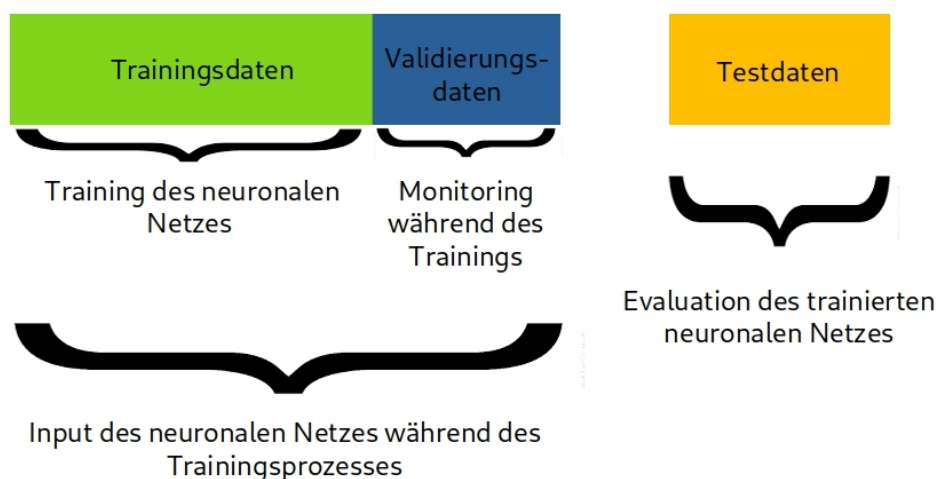


Abbildung 3: Funktion der Trainings-, Validierungs- und Testdaten

2.2.3. Transferlernen

Eine Methode zum Training von neuronalen Netzen, die auch in meiner Arbeit zur Anwendung kam, ist das Transferlernen²⁶. Hierbei wird sich zunutze gemacht, dass jedes Bild, ob von Tieren, Landschaften oder histologischen Geweben, aus ähnlichen Grundformen wie zum Beispiel Ecken, Kanten oder Rundungen besteht. Daher ist es nicht notwendig, beim Trainieren eines neuronalen Netzes zur Bilderkennung jedes Mal von vorne anzufangen. Stattdessen kann man ein bereits vortrainiertes CNN nutzen und für eine

bestimmte Fragestellung weitertrainieren²⁷. Dabei werden klassischerweise nur die CL und PL verwendet. Die abschließenden FCL werden dabei von dem vorherigen Teil des vortrainierten CNN getrennt und durch naive FCL ersetzt. Das weitere Training findet schließlich nur an diesen naiven Schichten statt. Die vorherigen, bereits trainierten, Schichten werden von den Trainingsdaten durchlaufen, ohne dass deren Gewichte angepasst werden. Somit findet eine enorme Zeit- und Rechensparnis statt. Beispiele für bereits trainierte CNN sind VGGNet²⁸, ResNet²⁹ oder GoogLeNet³⁰. Alle drei Netze wurden im Rahmen der „ImageNet Large Scale Visual Recognition Challenge (ILSVRC)“ anhand von 1,2 Millionen einzelnen Bildern mit insgesamt 1000 verschiedenen Klassen trainiert³¹. Die Trainingsdaten sind ein Teil von *ImageNet*, einer öffentlich zugänglichen Datenbank, die aktuell aus über 14 Millionen Bildern besteht³².

2.2.4. Neuronale Netze in der Histopathologie

In den letzten Jahren wurden zunehmend Ansätze entwickelt, neuronale Netze zur diagnostischen Beurteilung von histologischen Schnitten zu verwenden. Dies beinhaltet unter anderem die Detektion von Tumorgewebe^{27,33}, das Rückschließen auf Mutationen^{34,35} sowie auch das Tätigen von prognostischen Aussagen bezüglich des Überlebens oder Auftretens eines Progresses³⁶. Der Vorteil dieser Methoden ist, dass bereits vor Betrachtung der histologischen Schnitte durch einen Pathologen oder Dermatopathologen Aussagen bezüglich klinischer Fragestellungen getroffen werden können. Dabei sollen neuronale Netze die Arbeit der Ärzte nicht ersetzen, jedoch könnte eine gewisse automatisierte Vorverarbeitung getroffen und schließlich ergänzend zur Beurteilung hinzugezogen werden, was eine Zeitersparnis ermöglicht und teilweise repetitive Aufgaben abnehmen kann.

2.2.5. Präprozessierung

Bevor man eingescannte histologische Schnitte, sogenannte Whole-Slide-Images (WSI), zum Training eines neuronalen Netzes verwenden kann, gibt es einige Punkte zu beachten. Zum einen sind diese Bilddateien aufgrund der hohen Auflösung mit einem benötigten Speicher von mehreren Gigabytes (GB) pro WSI sehr groß. Diese direkt ohne Präprozessierung zu verwenden würde eine immense Rechenleistung beanspruchen, die aktuell kaum verfügbar ist. Eine häufig genutzte Lösung ist daher das Zerteilen des WSI in viele kleine Stücke, sogenannte Tiles, die dann einzeln durch das neuronale Netz verarbeitet werden können. Pro WSI entstehen somit je nach Größe mehrere Tausend Tiles.

Der zweite Punkt ist, dass nicht alle Tiles relevante Informationen enthalten. Hintergrund, also Bereiche ohne Gewebe, wird beispielsweise nicht zur Entscheidung bezüglich klinischer Fragestellungen beitragen. Ansätze diesen herauszufiltern gibt es verschiedene. Während

manche Autoren mit Schwellenwerten bezüglich der Pixelintensität³⁶ oder des RGB-Farbraums³⁴ arbeiten, verwenden andere dafür bereits bestehende Algorithmen, mithilfe derer eine automatisierte Segmentierung des Gewebes erfolgen kann³⁷. Doch nicht nur der Hintergrund wird in der Regel vorher entfernt, auch ist gegebenenfalls nicht jeder Bereich innerhalb des Gewebes von Interesse. Tumorfrees Binde- oder Fettgewebe trägt womöglich nicht ausschlaggebend zu den Tumor betreffenden Fragestellungen bei, weshalb die meisten Studien die tumorfreen Tiles von dem Training ausschließen^{38,39}. Andere hingegen beziehen das gesamte Gewebe ein, was einen Schritt weniger in der Präprozessierung kostet, jedoch am Ende aufgrund der deutlich höheren Anzahl an, inklusive vermutlich weniger relevanten, Tiles mehr Rechenleistung beim Trainieren des neuronalen Netzes benötigt³⁵.

Der dritte Punkt beinhaltet die Farbnormalisierung. Häufig gibt es Unterschiede bezüglich des Farbstichs sowie der Farbintensität des Schnitts, insbesondere zwischen verschiedenen Instituten jedoch auch innerhalb einer Klinik. Um zu verhindern, dass das neuronale Netz unterschiedlich kräftige Färbungen oder divergierende Farbstiche in seine Beurteilung miteinbezieht, gibt es verschiedene Methoden, die Farben auf den Schnitten zu normalisieren. Hier zu nennen sind unter anderem die Macenko-Methode⁴⁰, die Reinhard-Methode⁴¹, sowie die Methode nach Vahadane et al.⁴².

Ein letzter möglicher Teil der Präprozessierung ist die Datenaugmentation. Hierbei wird die Anzahl an relevanten Tiles erhöht, indem einige davon zufällig gedreht, gespiegelt oder um ein paar Pixel verschoben werden. Dies ist insbesondere dann nützlich, wenn nur ein kleiner Trainingsdatensatz zur Verfügung steht, um diesen somit vervielfachen zu können⁴³.

2.3. Fragestellungen und Ziel der Arbeit

Kutane Plattenepithelkarzinome gehören zu den häufigsten Karzinomen in Deutschland. Ihre eigentlich gute Prognose verschlechtert sich drastisch, sobald lokale Rezidive oder Fernmetastasen auftreten. Jene Patienten bereits bei Diagnosestellung zu identifizieren, gelingt aktuell leider noch nicht suffizient. Bei anderen Tumorentitäten wie beispielsweise dem malignen Melanom³⁶ konnte bereits gezeigt werden, dass ein CNN in der Lage ist anhand von histologischen Schnitten der Exzidate einen Progress vorherzusagen. Somit ergibt sich die Fragestellung, ob dies auch bei cSCC möglich ist.

Das Ziel meiner Arbeit war die Entwicklung eines neuronalen Netzes (CNN_{Progress}), welches anhand von histologischen Schnitten von Primären von cSCC vorhersagen kann, ob der

jeweilige Patient einen Progress entwickeln wird oder nicht, um somit Hochrisikopatienten besser zu identifizieren und letztlich gezielter eine intensivere Therapie und/oder Nachsorge durchführen zu können. Ein zweites neuronales Netz (CNN_{Tumor}) sollte zudem entwickelt werden, um automatisiert den Tumorbereich auf den Slides einzugrenzen, da nur dieser für die Vorhersage eines Progresses durch CNN_{Progress} berücksichtigt werden sollte. Da es bei dieser Tumorentität bisher noch kaum Ansätze mit Deep Learning gibt, ist diese Arbeit eine der ersten, die neuronale Netze auf cSCC anwendet.

Ein weiteres Ziel war die Identifizierung von signifikanten klinischen Risikofaktoren bezüglich des Auftretens eines Progresses sowie die Erstellung eines Regressionsmodells, das basierend auf den gegebenen Risikofaktoren eines jeweiligen Patienten das Risiko für die Entwicklung eines Progresses ausgibt.

3. Material und Methoden

3.1. Material

3.1.1. Kohorte

Die meiner Arbeit zur Verfügung stehende Kohorte K_{ALL} bestand aus 1241 Patienten der Klinik für Dermatologie und Venerologie der Universitätsklinik Köln, bei denen zwischen dem 13.01.2009 und 02.05.2019 ein primäres Plattenepithelkarzinom der Haut diagnostiziert und exzidiert wurde. Retrospektiv waren Angaben zu dem Auftreten eines Progresses, dem Outcome, Eigenschaften des Tumors sowie klinischen Daten der Patienten aus dem medizinischen Dokumentationssystem *ORBIS* anonymisiert erfasst.

Für das Training des $CNN_{Progress}$ wurde daraus die Kohorte K_{Train} ($n = 84$) gebildet, welche aus jeweils 42 Schnitten von Primarien mit und ohne Auftreten eines Progresses im Verlauf bestand. Die Testkohorte K_{Test} ($n = 24$) bestand aus jeweils 12 Schnitten von Primarien mit und ohne Auftreten eines Progresses.

3.1.2. QuPath

Für das manuelle Setzen der Annotationen an den eingescannten WSI zur Erstellung des Trainings- und Testsets für das CNN_{Tumor} wurde *QuPath 2.3*⁴⁴ verwendet. Anschließend erfolgte in diesem Programm die Zerteilung der WSI des Trainings- und Testsets für die Tumorerkennung inklusive Export der annotierten Tiles.

3.1.3. Python und verwendete Bibliotheken

Als Programmiersprache aller Skripte, außerhalb der Anwendung von *QuPath*⁴⁴, wurde auf dem Rechencluster ausschließlich *Python 3.5*⁴⁵ und für die lokal ausgeführten Skripte *Python 3.9*⁴⁶ verwendet. Dabei wurden die folgenden Bibliotheken verwendet:

Zur statistischen Auswertung der klinischen Daten wurde *Python Data Analysis Library (pandas)*⁴⁷ angewandt. Die Präprozessierung der gescannten Slides erfolgte mittels *largeimage*⁴⁸ sowie *Python Image Library (PIL)*⁴⁹. Zum Trainieren und Testen der beiden CNN wurde die *Tensorflow API Keras*⁵⁰ verwendet. Zur Senkung der Rechenzeit wurde *multiprocessing*⁵¹ eingesetzt. Die grafische Darstellung der Ergebnisse erfolgte mithilfe von *Matplotlib*⁵² sowie *Scikit-learn*⁵³. In allen Skripten erfolgte die Anwendung von *NumPy*⁵⁴ zur Verarbeitung und Handhabung der Daten in Arrays.

3.1.4. Rechencluster

Aufgrund der benötigten hohen Rechenleistung erfolgte die Ausführung der Skripte 3, 4, 5, 9, 10, 11 (s. Punkt 7.3) auf dem HPC-Cluster der Universität zu Köln⁵⁵. Hierdurch konnte der notwendige Arbeitsspeicher von bis zu > 100 GB zur Verfügung gestellt werden, welcher hauptsächlich durch die Verarbeitung der Daten in den CL der CNN sowie durch die Verwendung von *multiprocessing*⁵¹ zustande kam. Die Ausführung der Skripte 1, 2, 6, 7, 8, 12, 13 (s. Punkt 7.3) konnte auf einem lokalen Rechner erfolgen.

3.2. Methoden

Zunächst soll anhand dieses Abschnitts sowie mithilfe von Abbildung 4 ein Überblick über den Ablauf der Trainingsprozesse gegeben werden, bevor die einzelnen Schritte genauer erläutert werden.

Für das Training des CNN_{Progress} zur Vorhersage von Progressen wurde die in Punkt 3.1.1 beschriebene Kohorte K_{Train} ($n = 84$) verwendet. Zunächst erfolgte die Zerteilung der gescannten WSI der cSCC-Primarien in kleinere Tiles, um eine Verarbeitung in dem CNN_{Progress} zu ermöglichen. Anschließend wurden Tiles mit zu viel Hintergrund automatisiert herausgefiltert. Im nächsten Schritt wurden zusätzlich jene Tiles entfernt, die keinen oder einen nur geringen Tumoranteil hatten, mit der Annahme, dass hierin wenig Informationen bezüglich des Auftretens eines Progresses stecken. Um dies ebenfalls möglichst automatisiert durchführen zu können, wurde im Vorfeld ein weiteres CNN_{Tumor} zur Tumordetektion trainiert. An den übrig gebliebenen Tumor-Tiles der Kohorte K_{Train} erfolgte schließlich das Training bezüglich des Auftretens eines Progresses anhand von Transferlernens mithilfe eines vortrainierten VGG19-Netzes. Abschließend konnte das trainierte CNN_{Progress} an der Testkohorte K_{Test} ($n = 24$) evaluiert werden. Im Folgenden sollen die genannten Schritte genauer erläutert werden.

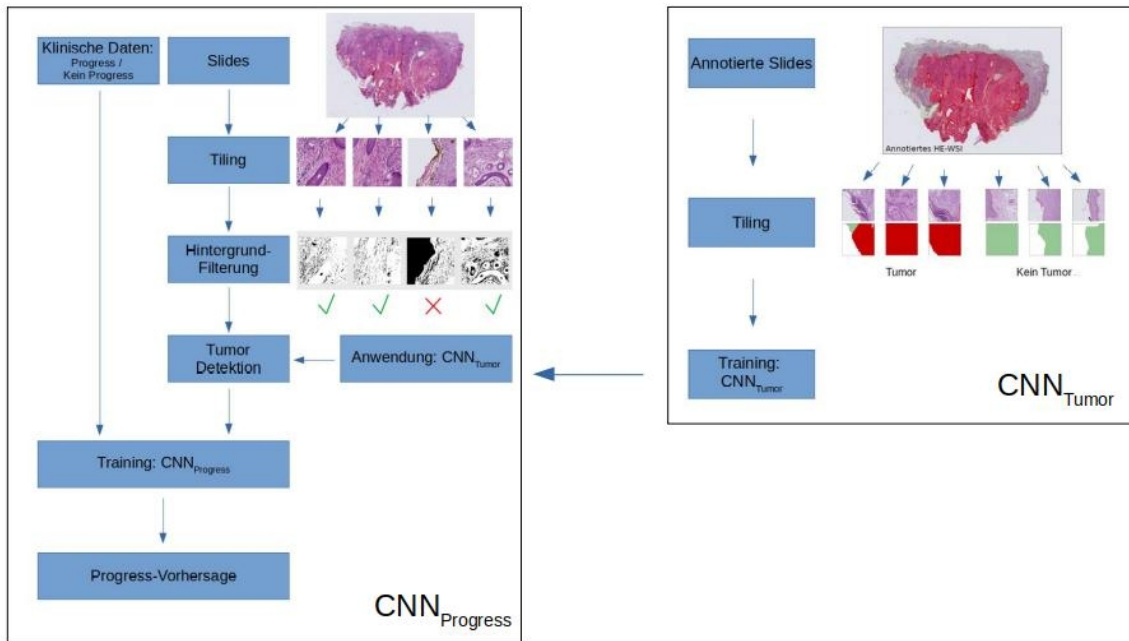


Abbildung 4: Übersicht über die Trainingsprozesse von $CNN_{Progress}$ und CNN_{Tumor}

3.2.1. Detektion des Tumorbereichs

Für die automatisierte Tumordetektion auf den Tiles der Kohorte K_{Train} wurde ein separates CNN_{Tumor} trainiert. Dafür wurden aus der Kohorte K_{ALL} 12 gescannte WSI zufällig ausgewählt.

(1) Präprozessierung

In dem Programm *QuPath*⁴⁴ erfolgte die manuelle Annotation der zufällig ausgewählten WSI in „Tumor“, „tumorfrees Gewebe“ und „Hintergrund“ (s. Abbildung 5). Anschließend wurde das gesamte WSI in 224x224 Pixel große Tiles auf 5-facher Vergrößerungsstufe zerteilt. Dafür wurde das in der *QuPath*-Dokumentation verfügbare Beispielskript⁵⁶ entsprechend angepasst (s. Skript 1, Punkt 7.3.1). Die angegebene Größe der Tiles wurde gewählt, um mit der Input-Dimension des anschließend zur Verwendung kommenden VGG19-CNN²⁸ übereinzustimmen. Mithilfe der exportierten Annotationen sowie der Python-Bibliothek *Python Image Library (PIL)*⁴⁹ wurden zunächst Tiles mit mehr als 40 % Hintergrund-Anteil herausgefiltert. Übrig gebliebene Vordergrund-Tiles mit einem Tumor-Anteil von über 30 % wurden als „Tumor“ und Tiles mit einem Tumoranteil von 0 % als „tumorfrees Gewebe“ klassifiziert. Tiles mit einem Tumor-Anteil bis zu 30 % wurden vom Training ausgeschlossen (s. Abbildung 4b und Skript 2, Punkt 7.3.2). Insgesamt blieben somit 12527 Tiles übrig, die zum Trainieren des CNN_{Tumor} verwendet werden konnten.

Auf Datenaugmentation wurde aufgrund der bereits hohen Anzahl an Tiles zugunsten einer effizienteren Verarbeitung verzichtet. Eine Farbnormalisierung mittels Macenko-Methode⁴⁰ sowie nach Vahadane⁴² wurde mithilfe der Python-Bibliothek *staintools*⁵⁷ probatorisch angewendet, fand aufgrund einer fehlenden Verbesserung des Ergebnisses jedoch keine weitere Anwendung.

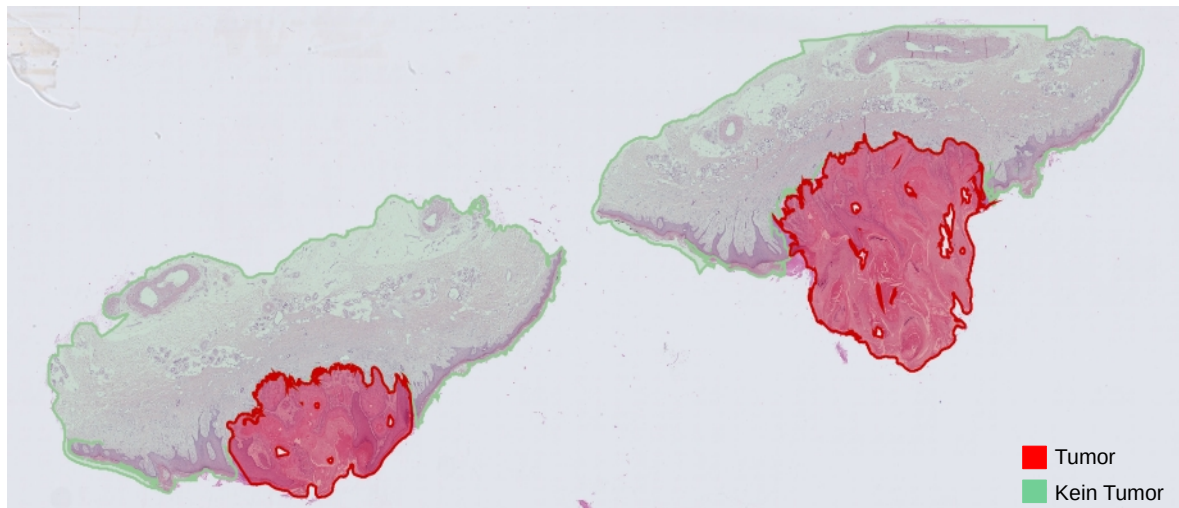


Abbildung 5: Manuelle Annotation des Tumorbereichs auf den Slides

(2) Training

Das Training des CNN_{Tumor} anhand der 12527 Tiles erfolgte mittels Transferlernens. Hierfür wurde das bereits vortrainierte VGG19-CNN²⁸ verwendet. Zunächst wurden die CL und PL von den Tiles durchlaufen, ohne dass die bereits trainierten Gewichte des Kerns dabei verändert wurden. Da die Verarbeitung der Tiles innerhalb dieser Schichten des CNN äußerst zeitaufwendig ist, wurden hier mittels der Python-Bibliothek *multiprocessing*⁵¹ mehrere Prozesse gleichzeitig genutzt und die Daten stapelweise verarbeitet (s. Skript 3, Punkt 7.3.3). Somit gelang eine Reduktion der Programmlaufzeit auf 4 Stunden und 21 Minuten bei einer CPU-Rechenzeit von insgesamt 2 Tagen, 4 Stunden und 22 Minuten. Anschließend erfolgte das Training der naiven FCL anhand der Ausgabe der vorherigen Schichten (s. Skript 4, Punkt 7.3.4). Diese bestanden aus jeweils zwei Abfolgen von FCL und DL. Die bereits in Punkt 2.2.1 dargestellten Abbildungen 1 und 2 geben eine Übersicht über die Architektur des verwendeten VGG19-CNN sowie der von uns hinzugefügten FCL. Für das Training der FCL wurden verschiedene Parameter für die „Batch“-Größe, die Lernrate

sowie die Anzahl der zu durchlaufenden Trainingsepochen getestet. Der Validierungsanteil, anhand dessen der Trainingsprozess kontrolliert wurde, betrug 20 %.

(3) Testen

Anschließend erfolgte die Testung des trainierten $\text{CNN}_{\text{Tumor}}$. Dies erfolgte an 11 zufällig ausgewählten WSI der Kohorte K_{ALL} , die analog zu dem Trainingsset zunächst annotiert, in 224×224 Pixel große Tiles auf 5-facher Vergrößerungsstufe zerteilt (s. Skript 1, Punkt 7.3.1) und anschließend nach den gleichen o.g. Kriterien in „Tumor“ und „tumorfrees Gewebe“ sortiert wurden (s. Skript 2, Punkt 7.3.2). Insgesamt blieben 15429 Tiles zum Testen des $\text{CNN}_{\text{Tumor}}$ übrig. Die Auswertung sowie Darstellung der Ergebnisse erfolgte anhand der Python-Bibliotheken *Matplotlib*⁵² sowie *Scikit-learn*⁵³. Bei nur zwei verfügbaren Klassen und nur einer zulässigen wahren Klassifikation wurde zunächst die Klasse dem jeweiligen Tile zugeteilt, dessen ausgegebene Wahrscheinlichkeit $> 0,5$ betrug. Anschließend wurde anhand der Erstellung einer ROC-Kurve sowie der Berechnung des Youden-Index ein idealer Schwellenwert der Tumorstwahrscheinlichkeit errechnet, ab dem ein Tile als solches klassifiziert werden sollte (s. Skript 5, Punkt 7.3.5).

(4) Klassifikation ganzer Slides

Auch wenn das hier trainierte $\text{CNN}_{\text{Tumor}}$ vor allem zur automatisierten Klassifikation einzelner Tiles in „Tumor“ und „tumorfrees Gewebe“ zur weiteren Vorhersage des Auftretens eines Progresses an den „Tumor“-Tiles genutzt werden sollte, wurde dennoch getestet, inwiefern es auch ganze Slides klassifizieren kann. Dafür wurde der jeweilige Tumoranteil der 11 WSI des Testsets durch das $\text{CNN}_{\text{Tumor}}$ vorhergesagt. Zudem wurde es auf 10 gescannte tumorfrees WSI angewendet und auch hier ein Tumoranteil durch das $\text{CNN}_{\text{Tumor}}$ bestimmt. Dafür wurden diese tumorfrees WSI zunächst anhand der Python-Bibliothek *largeimage*⁴⁸ in 224×224 Pixel große Tiles auf 5-facher Vergrößerungsstufe zerteilt (s. Skript 6, Punkt 7.3.6). Da sich hier kein Tumorgewebe befand, war eine vorherige Annotation nicht notwendig. Der Hintergrund wurde mithilfe der Python-Bibliothek *Python Image Library (PIL)*⁴⁹ herausgefiltert. Die Bestimmung des Hintergrundes erfolgte mittels temporärer Graustufung und anschließender Anwendung eines Schwellenwerts. Tiles mit einem Vordergrundanteil von $\leq 40\%$ wurden schließlich ausgeschlossen, sodass die Testung an den Vordergrund-Tiles stattfinden konnte (s. Abbildung 6 und Skript 7, Punkt 7.3.7).

Unter Berücksichtigung, dass sich falsch-positive Tiles kaum vollständig vermeiden lassen, wurde anhand der vorhergesagten Tumoranteile aller WSI mithilfe der Erstellung einer ROC-Kurve inklusive Berechnung des Youden-Index ein optimaler Schwellenwert für den

vorhergesagten Tumoranteil eines WSI bestimmt, ab dem das WSI als „Tumor-enthaltend“ eingestuft werden sollte.

Abschließend wurde der als „Tumor“ eingestufte Bereich auf den WSI markiert. Dafür wurden die Dateien der eingescannten WSI mithilfe der Python-Bibliothek *OpenSlide*⁵⁸ zunächst in ein JPEG-Format mit 20-fach reduzierter Auflösung umgewandelt. Danach konnten die als „Tumor“ klassifizierten Tiles mithilfe der Bibliothek *Python Image Library (PIL)*⁴⁹ eingezeichnet werden (s. Skript 8, Punkt 7.3.8).

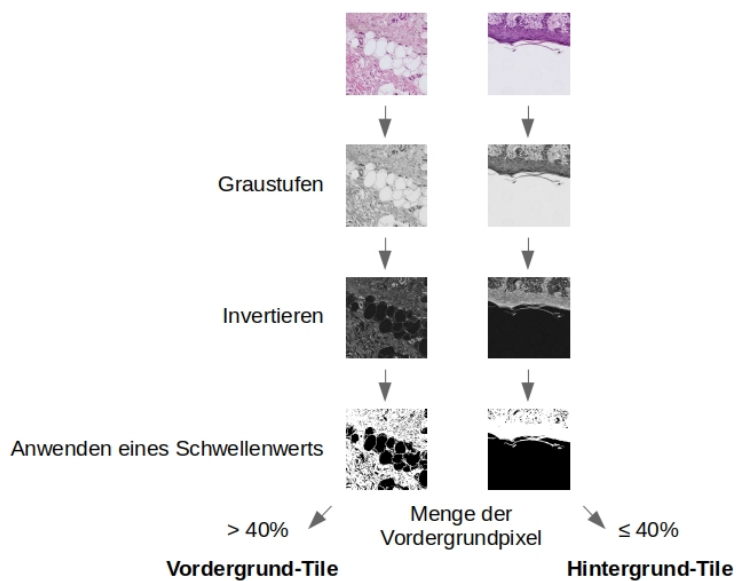


Abbildung 6: Automatisches Aussortieren des Hintergrunds

3.2.2. Vorhersage von Progressen

(1) Präprozessierung

Für das Training des $\text{CNN}_{\text{Progress}}$ wurde die Kohorte K_{Train} ($n = 84$), bestehend aus jeweils 42 WSI von exzidierten cSCC-Primären von Patienten mit und ohne Entwicklung eines Progresses, verwendet. „Progress“ beinhaltet sowohl lokale Rezidive als auch Fernmetastasen. Im Rahmen der Präprozessierung der Daten erfolgte zunächst die Generierung von 224×224 Pixel großen Tiles in 5-facher Vergrößerung mithilfe der Python-Bibliothek *largeimage*⁴⁸ (s. Skript 6, Punkt 7.3.6). Anschließend wurden Tiles mit einem

Vordergrundanteil von $\leq 40\%$ herausgefiltert. Dies geschah analog zu der im vorherigen Punkt beschriebenen Präprozessierung der Normalhaut-Tiles mithilfe der Bibliothek *Python Image Library (PIL)*⁴⁹ (s. Abbildung 6 und Skript 7, Punkt 7.3.7). Im nächsten Schritt folgte die automatische Tumorklassifikation durch das $\text{CNN}_{\text{Tumor}}$ (s. Skript 9, Punkt 7.3.9). Da dieser Schritt aufgrund der Prozessierung in den CL und PL des $\text{CNN}_{\text{Tumor}}$ erneut sehr rechenaufwendig war, wurden hier ebenfalls mittels der Python-Bibliothek *multiprocessing*⁵¹ mehrere Prozesse gleichzeitig verwendet, um die CPU-Rechenzeit von 15 Tagen, 4 Stunden und 20 Minuten auf eine Laufzeit von 1 Tag, 12 Stunden und 26 Minuten zu reduzieren.

(2) Training

Anhand der 28250 übrig gebliebenen Tumor-Tiles konnte nun das Training des $\text{CNN}_{\text{Progress}}$ erfolgen (s. Skript 10, Punkt 7.3.10). Da sowohl für das $\text{CNN}_{\text{Tumor}}$ als auch das $\text{CNN}_{\text{Progress}}$ das gleiche VGG19-Netz zum Transferlernen verwendet wurde, konnten die bereits in den CL und PL des $\text{CNN}_{\text{Tumor}}$ verarbeiteten Daten von K_{Train} für das Training der naiven FCL des $\text{CNN}_{\text{Progress}}$ weiterverwendet werden. Die Architektur des gesamten $\text{CNN}_{\text{Progress}}$ entsprach dabei der des $\text{CNN}_{\text{Tumor}}$ (s. Abbildungen 1 und 2). Für das Training der FCL wurden verschiedene Parameter für die „Batch“-Größe, die Lernrate sowie die Anzahl der zu durchlaufenden Trainingsepochen getestet.

(3) Testen

Das Testen des $\text{CNN}_{\text{Progress}}$ erfolgte an der Kohorte K_{Test} ($n = 24$), bestehend aus jeweils 12 WSI von exzidierten cSCC-Primarien von Patienten mit und ohne Entwicklung eines Progresses. Diese wurden analog zu den Trainingsdaten präprozessiert, sodass letztlich 10054 Tumor-Tiles als Testdaten übrig blieben. Für die Auswertung und Darstellung der Ergebnisse wurden hier ebenfalls die Python Bibliotheken *Matplotlib*⁵² sowie *Scikit-learn*⁵³ verwendet. Bei auch hier nur zwei verfügbaren Klassen („Entwicklung eines Progresses im Verlauf“ oder im Folgenden auch häufig verkürzt als das Label „Progress“ bezeichnet und „Kein Auftreten eines Progresses im Verlauf“ oder im Folgenden auch verkürzt als das Label „Kein Progress“ bezeichnet) und nur einer zulässigen wahren Klassifikation wurde zunächst die Klasse dem jeweiligen Tile zugeteilt, dessen Wahrscheinlichkeit $> 0,5$ betrug. Anschließend wurde anhand der Erstellung einer ROC-Kurve sowie der Berechnung des Youden-Index ein idealer Schwellenwert der Wahrscheinlichkeit für das Auftreten eines Progresses errechnet, ab dem ein Tile als „Progress“ klassifiziert werden sollte.

(4) Klassifikation ganzer Slides

Letztlich wurde geprüft, wie groß der Anteil an klassifizierten „Progress“-Tiles innerhalb eines WSI sein muss, um von einem hohen Risiko für das Auftreten eines Progresses für den Patienten zu sprechen. Dies erfolgte analog zur Tumordetektion mittels Erstellung einer ROC-Kurve inklusive Berechnung des Youden-Index (s. Skript 11, Punkt 7.3.11).

Abschließend erfolgte die Markierung der „Progress“-Tiles auf den WSI (s. Skript 12, Punkt 7.3.12).

3.2.3. Vorhersage von Progressen anhand klinischer Merkmale

Neben der Vorhersage von Progressen durch ein CNN wurden die in den Kohorten K_{Train} und K_{Test} ($n = 108$) erfassten klinischen Merkmale als Risikofaktoren bezüglich des Auftretens eines Progresses untersucht. Dabei wurden sowohl klinische Merkmale der Patienten (Alter, Geschlecht, Immunsuppression, Vorliegen einer Psoriasis oder eines Diabetes mellitus) als auch Merkmale des Tumors (Tumordicke, Tumordurchmesser, Lokalisation, Sicherheitsabstand nach Exzision, Perineuralscheideninvasion, lymphatische Invasion, vaskuläre Invasion, Invasion über die Subkutis hinaus sowie histologische Merkmale wie Ulzeration, Desmoplasie, infundibulozystisches Wachstum) untersucht. Dies erfolgte mithilfe der Python-Bibliotheken *pandas*⁴⁷, *Matplotlib*⁵² sowie *Scikit-learn*⁵³ (s. Skript 13, Punkt 7.3.13). Bei den metrischen Variablen Alter, Tumordicke, Tumordurchmesser und Sicherheitsabstand nach Exzision erfolgte bei Nicht-Normalverteilung mittels Mann-Whitney-U-Test die Testung, ob sich die jeweiligen Mediane der Patienten, die einen Progress entwickelten, von denen ohne Auftreten eines Progresses signifikant unterscheiden. Die Signifikanztestung bezüglich eines höheren Risikos für einen Progress bei Vorliegen der restlichen nominalen Variablen erfolgte mittels Chi-Quadrat-Test. Das Signifikanzniveau wurde für $\alpha = 0,05$ gewählt.

Zudem wurde ein Regressionsmodell entwickelt, welches basierend auf den erhobenen klinischen Parametern eine Vorhersage bezüglich der Wahrscheinlichkeit für das Auftreten eines Progresses vornehmen sollte (s. Skript 13, Punkt 7.3.13). Hierfür erfolgte die Aufteilung der zusammengeführten Kohorten K_{Train} und K_{Test} erneut zufällig in eine Trainings- ($n = 86$) und Testkohorte ($n = 22$). Diese zufällig beim Durchlaufen des Skripts erstellten Trainings- und Testkohorten für das Regressionsmodell sind unterschiedlich und unabhängig von der Zuordnung in K_{Train} und K_{Test} . Die Ergebnisse wurden in Form einer Konfusionsmatrix sowie einer ROC-Kurve dargestellt. Zunächst erfolgte dies mit einem gewählten Schwellenwert für die Progresswahrscheinlichkeit von 0,5 und anschließend mit einem mittels Youden-Index berechneten idealen Schwellenwert.

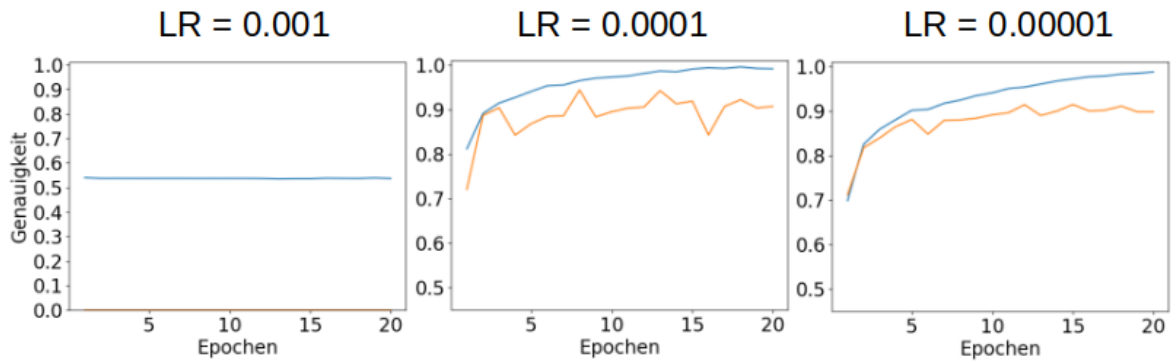
4. Ergebnisse

In meiner Arbeit wurden jeweils zwei neuronale Netze ($\text{CNN}_{\text{Tumor}}$, $\text{CNN}_{\text{Progress}}$) erstellt, trainiert und evaluiert. Dieser Prozess, inklusive der Präprozessierung der Bilddaten (Zerteilung der WSI in Tiles, Herausfiltern des Hintergrundes, Auswahl relevanter Bildbereiche, Erstellung der Trainings- und Testdaten) und Parallelisierung der Rechenprozesse wurde im vorherigen Punkt „3. Material und Methoden“ ausführlich dargelegt. Er ist neben den unten beschriebenen Aspekten bereits als Ergebnis meiner Arbeit zu verstehen, wurde jedoch aus didaktischen Gründen unter Punkt 3 beschrieben. In diesem Abschnitt werden neben der statistischen Auswertung der Kohorte nun vornehmlich die Trainings- und Testergebnisse der jeweiligen CNN dargelegt.

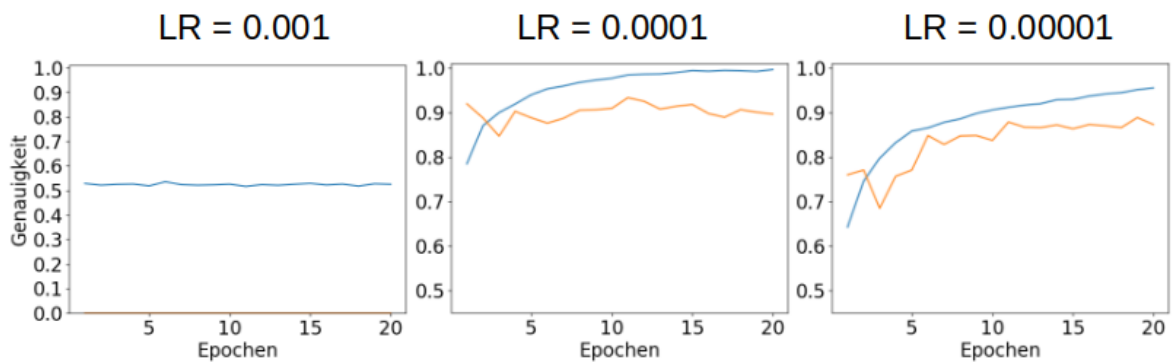
4.1. Detektion des Tumorbereichs

Für eine automatisierte Eingrenzung des Tumorbereichs zur nachfolgenden Vorhersage der Progresswahrscheinlichkeit anhand der „Tumor“-Tiles wurde mittels Transferlernens das $\text{CNN}_{\text{Tumor}}$ trainiert. Dabei wurden nur die Gewichte der FCL, nicht aber die der bereits vor-trainierten CL und PL des VGG19-Netzes, angepasst. Hierfür wurden verschiedene Trainingsparameter für die „Batch“-Größe, die Lernrate sowie die Anzahl der zu durchlaufenden Trainingsepochen getestet. Die Trainingsergebnisse werden in Abbildung 7 dargestellt. Hierbei erkennt man die jeweilige Trainings- und Validierungsgenauigkeit des Netzes in Abhängigkeit der gewählten Trainingsparameter. Während unabhängig von der „Batch“-Größe bei einer Lernrate von 10^{-3} die Validierungsgenauigkeit nicht anstieg, konnten bei einer Lernrate von 10^{-4} bzw. 10^{-5} nahezu stetig ansteigende Trainingsergebnisse mit Validierungsgenauigkeiten um 0,9 erreicht werden, bis letztlich die Überanpassung zu groß wurde. Schließlich konnten die besten und unten beschriebenen Testergebnisse mit einer „Batch“-Größe von 128, dem Adam-Optimizer mit einer Lernrate von 10^{-5} sowie einer Epochenzahl von 20 erzielt werden. Die Anwendung einer Farbnormalisierung verbesserte das Ergebnis nicht, sodass abschließend keine verwendet wurde.

a) Batchsize = 32



b) Batchsize = 64



c) Batchsize = 128

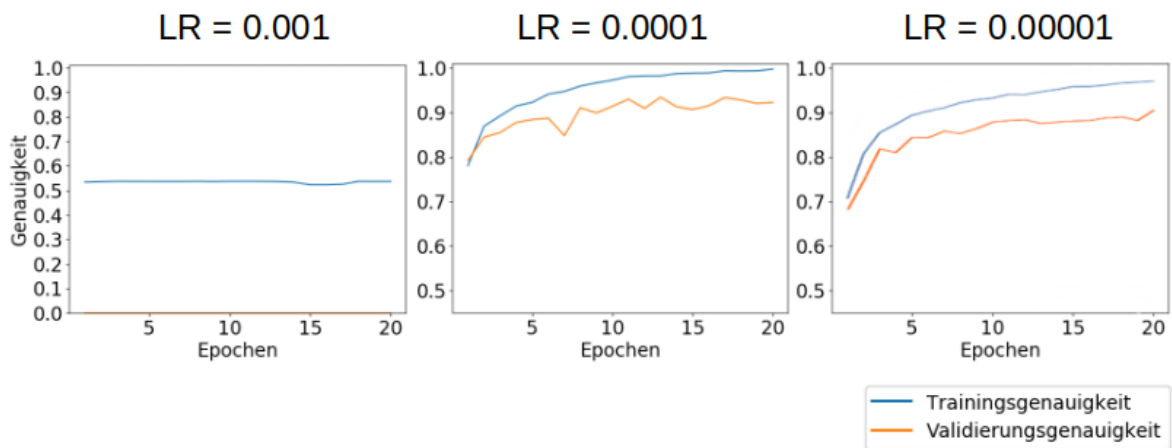


Abbildung 7: Trainingsergebnisse von CNN_{Tumor} bei verschiedenen „Batch“-Größen und Lernraten (LR)

4.1.1. Klassifikation der Tiles durch CNN_{Tumor}

Anschließend wurde das CNN_{Tumor} anhand eines separaten Testsets, bestehend aus 11 Schnitten von cSCC-Exzidaten mit insgesamt jeweils 5424 Tumor-abbildenden und 10005 tumorfreien Tiles, evaluiert. Unter der Annahme, dass ein Tile ab einer ausgegebenen Tumorwahrscheinlichkeit P_1 von $> 0,5$ als „Tumor“ klassifiziert werden sollte, wurden 4838 Tiles als richtig-positiv, 9124 Tiles als richtig-negativ, 881 Tiles als falsch-positiv und 586 Tiles als falsch-negativ klassifiziert. Dies ergab eine Sensitivität von 89 %, eine Spezifität von 91 %, einen positiven prädiktiven Wert (PPW) von 85 % und einen negativen prädiktiven Wert (NPW) von 94 % (s. Abbildung 8a und 8b).

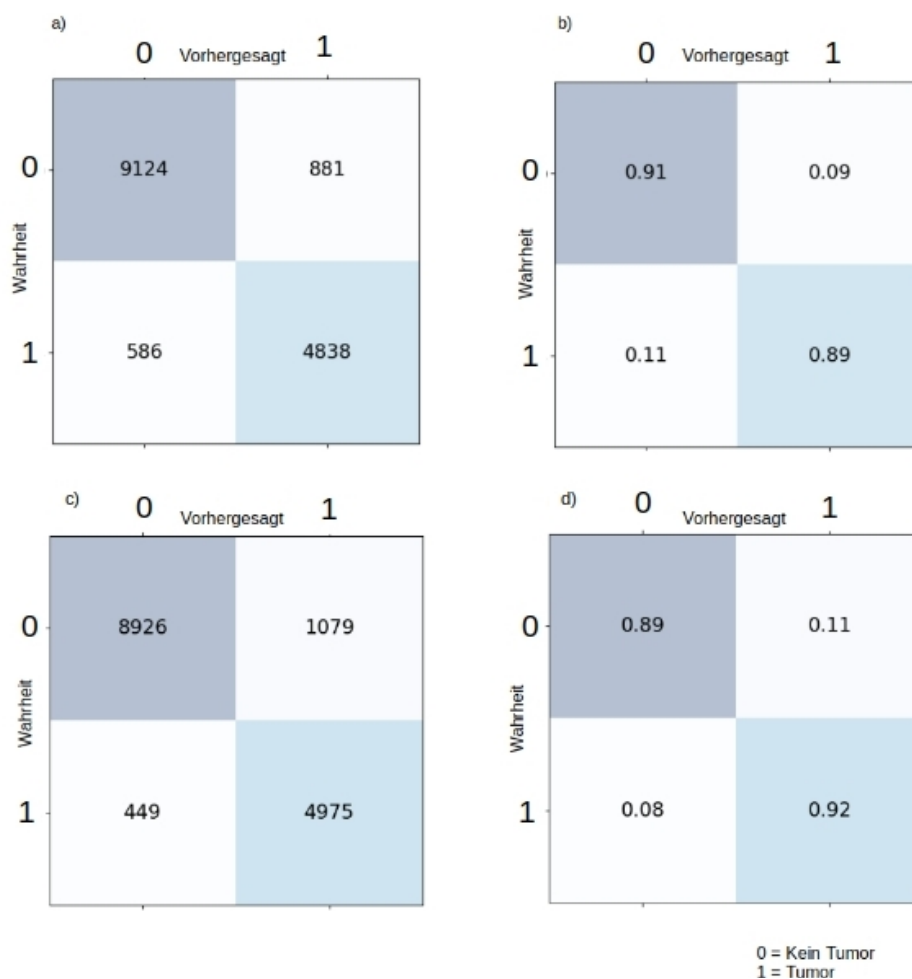


Abbildung 8: Klassifikation der Tiles durch das CNN_{Tumor} bei $P_1 = 0,5$ absolut (a) und prozentual (b). Klassifikation der Tiles durch das CNN_{Tumor} bei $P_1 = 0,278$ absolut (c) und prozentual (d).

Am eindeutigsten traf das Netz dabei die richtig-negativen Entscheidungen mit einem medianen P_1 -Wert von 0,000 (Min: 0,000, Max: 0,499). Auch die richtig-positiven Entscheidungen wurden bei einem medianen P_1 -Wert von 0,992 (Min: 0,502, Max: 1,0) mit einer hohen Sicherheit getroffen. Dagegen bestand bei den falsch-positiv (P_1 -Median: 0,865, Min: 0,502, Max: 1,0) sowie falsch-negativ (P_1 -Median: 0,061, Min: 0,000, Max: 0,499) klassifizierten Tiles eine höhere Unsicherheit vonseiten des CNN_{Tumor} (s. Abbildung 9). Daher erfolgte anschließend die Bestimmung eines idealen Schwellenwerts für P_1 , ab dem ein Tile als „Tumor“ eingestuft werden sollte, mittels Erstellung einer ROC-Kurve (s. Abbildung 10) sowie Berechnung des Youden-Index. Dieser betrug 0,278, womit 4975 Tiles als richtig-positiv, 8926 Tiles als richtig-negativ, 1079 Tiles als falsch-positiv und 449 Tiles als falsch-negativ klassifiziert wurden. Dies ergab eine Sensitivität von 92 %, eine Spezifität von 89 %, einen PPW von 82 % und einen NPW von 95 % (s. Abbildung 8c und 8d). Die Erstellung einer ROC-Kurve ergab eine AUC von 0,96.

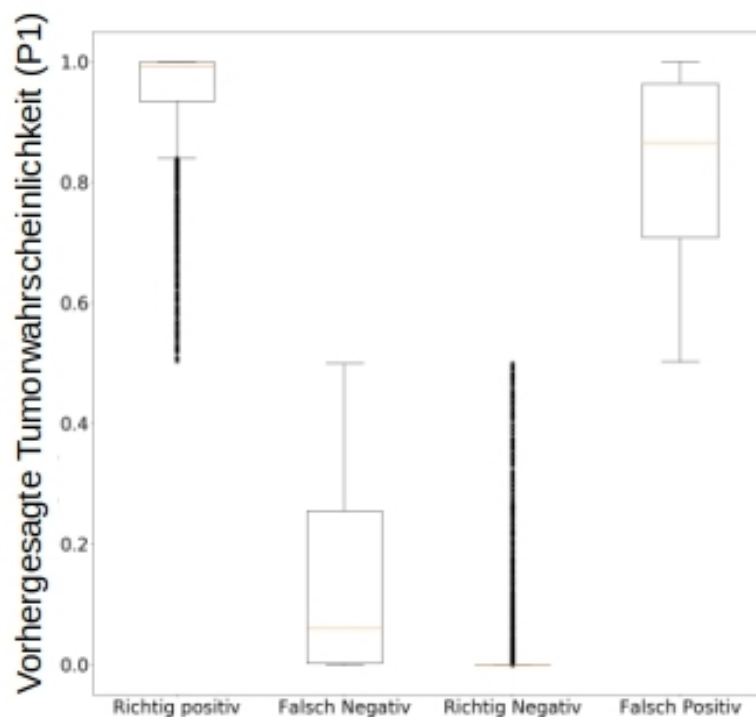


Abbildung 9: Verteilung der Tumorwahrscheinlichkeit P_1

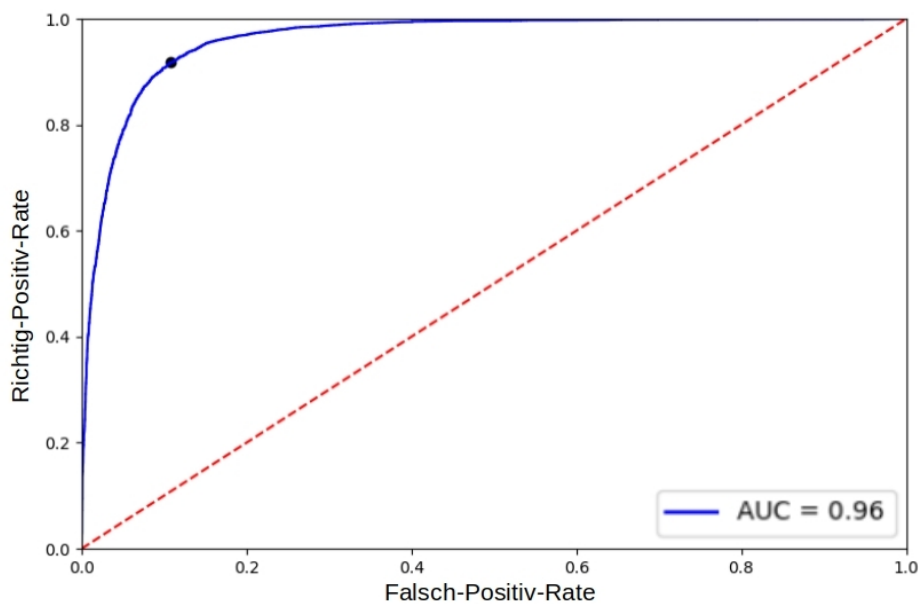


Abbildung 10: ROC-Kurve der Tumordetektion

4.1.2. Klassifikation ganzer Slides durch CNN_{Tumor}

Auch wenn das hier trainierte CNN_{Tumor} vor allem zur automatisierten Klassifikation einzelner Tiles in „Tumor“ und „tumorfrees Gewebe“ zur nachfolgenden Vorhersage eines Progresses anhand der „Tumor“-Tiles genutzt werden sollte, wurde dennoch getestet, inwiefern es auch ganze Slides klassifizieren kann. Da auf histologischen Schnitten von exzidierten cSCC in der Regel auch immer ein gewisser Anteil an Normalgewebe vorhanden ist und sich auf der anderen Seite auf vollständig tumorfreen Slides falsch-positiv klassifizierte Tiles kaum gänzlich vermeiden lassen, stellt sich die Frage, ab welchem Anteil an klassifizierten „Tumor“-Tiles ein ganzes WSI als „Tumor-enthaltend“ eingestuft werden sollte.

Zur Beantwortung dieser Frage wurde zusätzlich zu dem bereits oben beschriebenen Testset, welches ausschließlich aus Schnitten von cSCC-Exzidaten bestand, ein weiteres Testset, bestehend aus 10 Slides mit vollständig tumorfrem kutanem Gewebe hinzugenommen und der jeweilige Anteil an Tumor-abbildenden und tumorfreen Tiles durch das CNN_{Tumor} klassifiziert (s. Abbildung 11). Mithilfe der Berechnung des Youden-Index wurde ein idealer Schwellenwert für einen Tumoranteil von 0,217 ermittelt, ab dem ein ganzes WSI als „Tumor-enthaltend“ eingestuft werden sollte. Damit wurden alle 11 „Tumor“-Slides als richtig-positiv, 9/10 Normalhaut-Slides als richtig-negativ und 1/10 Normalhaut-Slides als falsch-positiv klassifiziert. Dies ergab eine Sensitivität von 100 %, eine Spezifität von 90 %, einen PPW von 92 % und einen NPW von 100 %.

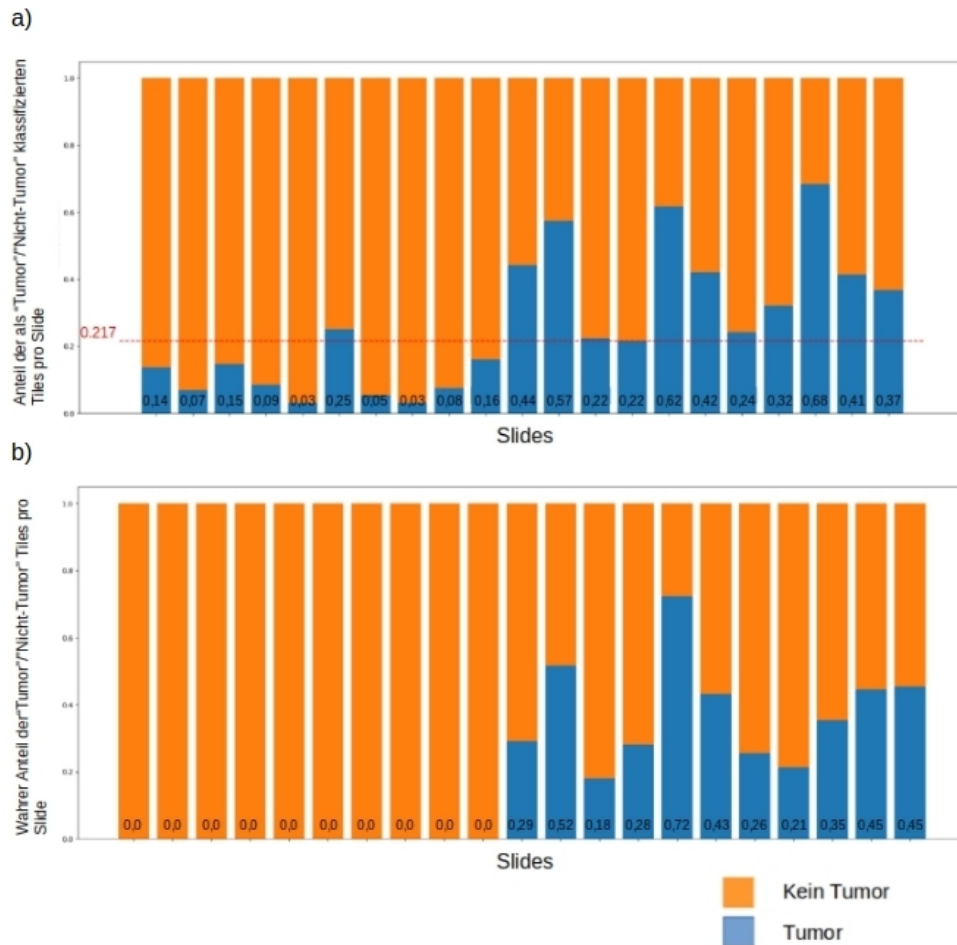


Abbildung 11: Durch das CNN_{Tumor} vorhergesagter (a) sowie wahrer (b) Anteil an "Tumor"-Tiles pro Slide mit Markierung des Youden-Index von 0,217

Abbildung 12 zeigt die Markierung der klassifizierten „Tumor“-Tiles exemplarisch auf drei cSCC-Exzidaten. Hierbei erkennt man eine zuverlässige Markierung des Tumorbereichs bei zusätzlich wenigen falsch-positiven Tiles in tumorfremem Gewebe. Diese finden sich vor allem im Epithel oder in den Hautanhangsgebilden und weniger in der Dermis oder Subkutis.

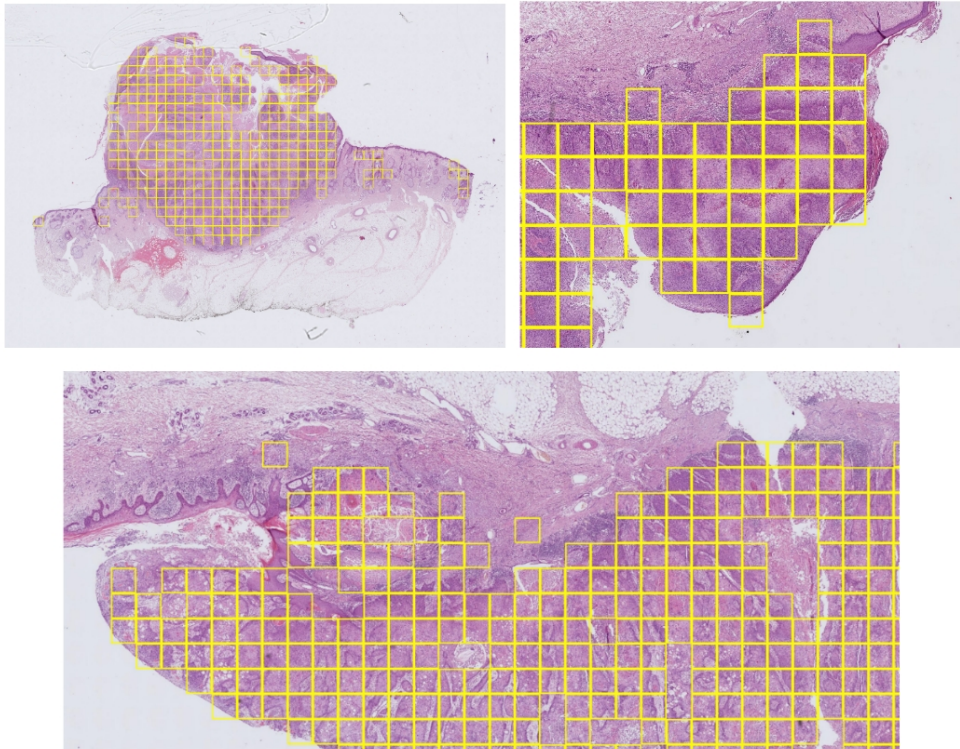


Abbildung 12: Markierung der durch das CNN_{Tumor} als "Tumor" klassifizierten Tiles auf den Slides

4.2. Vorhersage von Progressen

4.2.1. Deskriptive Statistik der Trainings- und Testkohorten

In der Kohorte K_{ALL} ($n = 1241$) trat bei 96 Patienten im Verlauf ein Progress auf, von denen uns 54 Slides der entsprechenden cSCC-Primarien zur Verfügung standen. Daraus wurden zufällig eine Trainingskohorte K_{Train} und eine Testkohorte K_{Test} gebildet und jeweils zu gleichen Teilen mit Slides von Patienten ohne Progress im Verlauf aufgefüllt. K_{Train} bestand somit aus jeweils 42 Schnitten von Patienten mit und ohne Entwicklung eines Progresses ($n = 84$) und K_{Test} aus jeweils 12 Schnitten von Patienten mit und ohne Entwicklung eines Progresses ($n = 24$).

Von allen 54 Patienten, die im Verlauf einen Progress entwickelten und deren Slides uns zur Verfügung standen, trat bei 39 Patienten (72,2 %) eine Fernmetastasierung, bei 24 Patienten (44,4 %) ein Lokalrezidiv und bei 9 Patienten (16,7 %) beides der genannten auf (s. Abbildung 13). Bei 44 der 54 cSCC mit Progress im Verlauf (81,5 %) trat dieses innerhalb der ersten beiden Jahre ein. Bei Auftreten eines Progresses jeglicher Art betrug die 5-Jahres-Überlebensrate (5-JÜR) 90,7 % (91,7 % bei Lokalrezidiv, 87,2 % bei Fernmetastasierung) und 100,0 % bei Progressfreiheit (s. Abbildung 14).

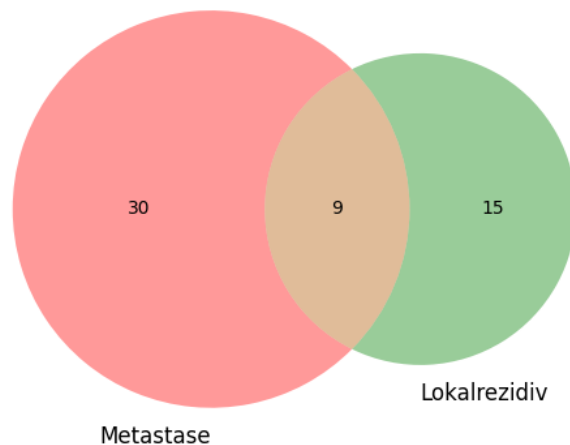


Abbildung 13: Anzahl an Metastasen und Lokalrezidiven in den Kohorten K_{Train} und K_{Test}

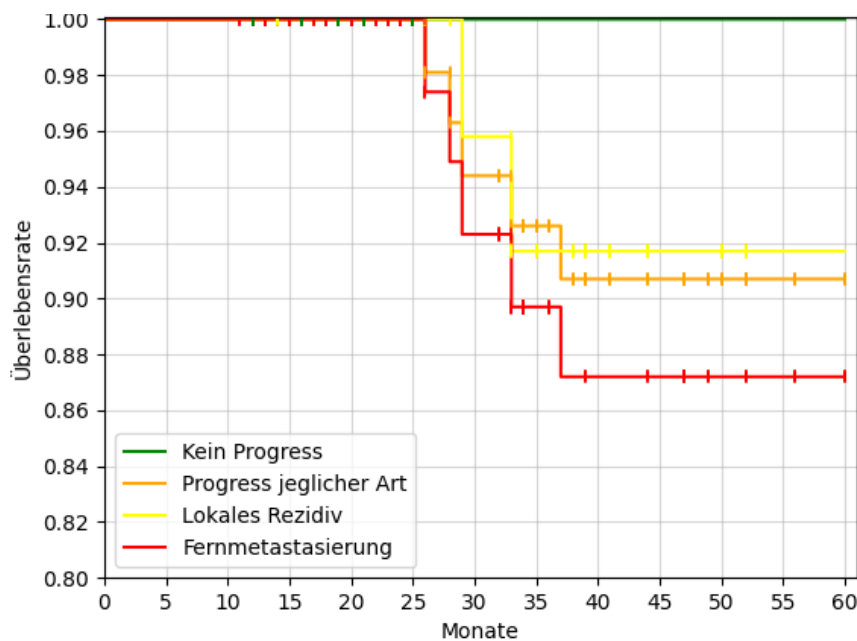


Abbildung 14: Gesamtüberlebensraten in Abhängigkeit des Auftretens eines Progresses

In der Kohorte K_{Train} waren insgesamt 42 Progresse vorhanden, davon bei 30 Patienten (71,4 %) in Form einer Fernmetastasierung, bei 18 Patienten (42,9 %) in Form eines Lokalrezidivs und bei 6 Patienten (14,3 %) trat beides der genannten ein. In K_{Train} betrug das mediane Alter 77,5 Jahre (Min: 46 Jahre, Max: 90 Jahre). Mit 71,4 % waren die Patienten überwiegend männlich. Die mediane Tumordicke betrug 3,2 mm (Min: 0,5 mm, Max: 20,0 mm) und bei Auftreten eines Progresses 5,0 mm (Min: 0,5 mm, Max: 20,0 mm). Der mediane Tumordurchmesser betrug 1,5 cm (Min: 0,1 cm, Max: 8,5 cm) und 2,0 cm (Min: 0,1 cm, Max:

5,0 cm) bei Auftreten eines Progresses. Desmoplastisches Wachstum fand sich bei insgesamt 2 Patienten (2,4 %). Eine Perineuralscheideninvasion konnte bei 5 Patienten (5,9 %) beobachtet werden, eine vaskuläre Invasion bei 3 Patienten (3,6 %) und eine lymphatische Invasion bei keinem Patienten. Bei 7 Patienten (8,3 %) kam es zu einer Invasion über das subkutane Gewebe hinaus.

In der Kohorte K_{Test} traten insgesamt 12 Progresse ein, bei 9 Patienten in Form einer Fernmetastasierung (75 %) und bei 6 Patienten als Lokalrezidiv (50 %). Bei 3 Patienten trat beides der genannten ein (25 %). In K_{Test} betrug das mediane Alter 78 Jahre (Min: 52 Jahre, Max: 90 Jahre). Mit 79,2 % waren die Patienten überwiegend männlich. Die mediane Tumordicke betrug 3,9 mm (Min: 1,3 mm, Max: 17,0 mm) im Vergleich zu 4,5 mm (Min: 1,6, Max: 12,5) bei Auftreten eines Progresses. Der mediane Tumordurchmesser betrug 2,9 cm (Min: 0,6 cm, Max: 4,0 cm) und 3,0 cm (Min: 0,6 cm, Max: 4,0 cm) bei Auftreten eines Progresses. Desmoplastisches Wachstum sowie eine lymphatische oder vaskuläre Invasion fanden sich jeweils bei keinem der Patienten. Eine Perineuralscheideninvasion konnte bei einem Patienten (4,2 %) beobachtet werden. Bei 2 Patienten (8,3 %) kam es zu einer Invasion über das subkutane Gewebe hinaus. Tabelle 2 gibt eine Übersicht über die deskriptive Statistik der Trainings- und Testkohorten.

Klinische Parameter	K_{Train}	K_{Test}	Total
Total	84	24	108
Progress	42	12	54
Lokalrezidiv	18	6	24
Fernmetastase	30	9	39
Männlich	60	19	79
Weiblich	24	5	29
Desmoplasie	2	0	2
Perineuralscheideninvasion	5	1	6
Vaskuläre Invasion	3	0	3
Lymphatische Invasion	0	0	0
Invasion über das subkutane Gewebe hinaus	7	2	9
Medianes Alter [Jahre]	77,5	78,0	78,0
Mediane Tumordicke [mm]	3,2	3,9	3,4
Medianer Tumordurchmesser [cm]	1,5	2,9	1,5

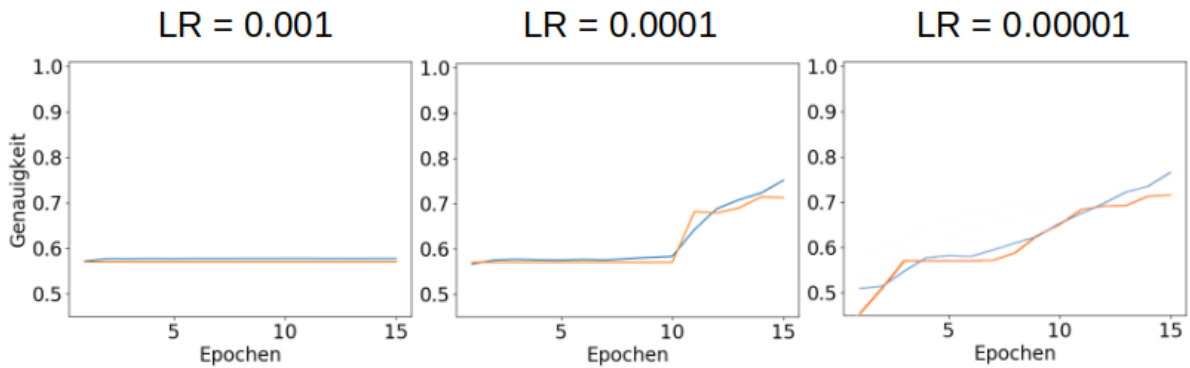
Tabelle 2: Deskriptive Statistik der Trainings- und Testkohorten

4.2.2. Klassifikation der Tiles durch $\text{CNN}_{\text{Progress}}$

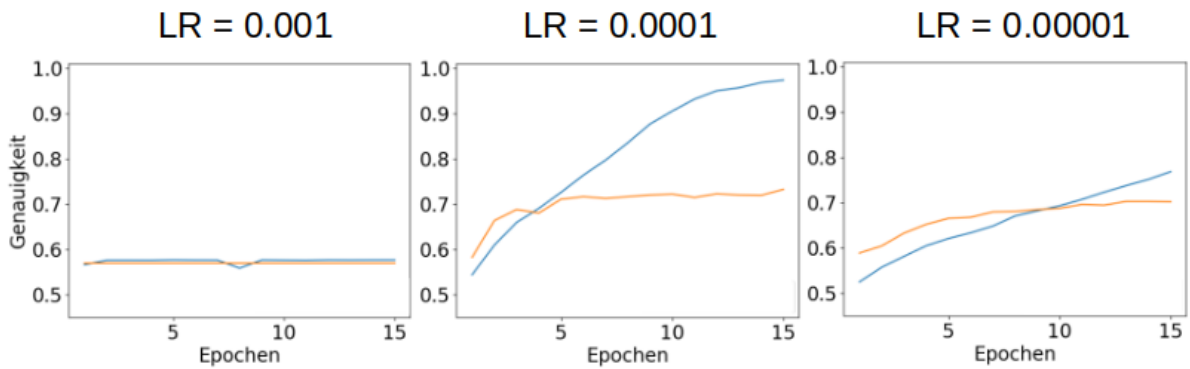
Analog zur Tumordetektion wurde zur Vorhersage von Progressen mittels Transferlernens ein $\text{CNN}_{\text{Progress}}$ an den Tiles der Kohorte K_{Train} trainiert, mit alleiniger Anpassung der Gewichte der FCL. Auch hier wurden verschiedene Trainingsparameter für die „Batch“-Größe, die Lernrate sowie die Anzahl der zu durchlaufenden Trainingsepochen getestet (s. Abbildung 15). Die höchste Validierungsgenauigkeit bei möglichst geringer Überanpassung konnte mit einer „Batch“-Größe von 64, dem Adam-Optimizer mit einer Lernrate von 10^{-4} bzw. 10^{-5} sowie

einer Epochenzahl von 15 erzielt werden, wobei die unten beschriebenen Testergebnisse mit einer Lernrate von 10^{-5} denen mit einer Lernrate von 10^{-4} überlegen waren.

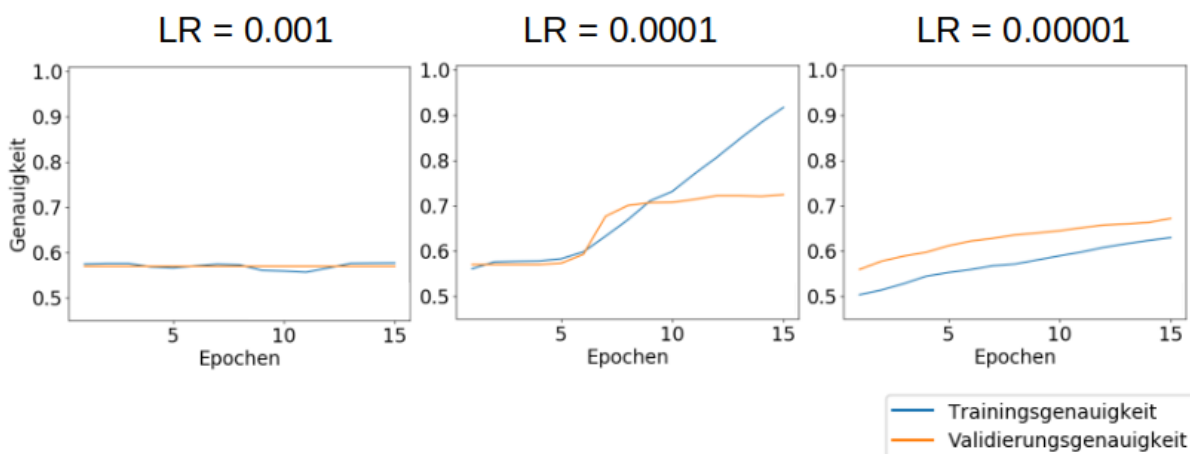
a) Batchsize = 64



b) Batchsize = 128



c) Batchsize = 256



— Trainingsgenauigkeit
— Validierungsgenauigkeit

Abbildung 15: Trainingsergebnisse von $CNN_{Progress}$ bei verschiedenen „Batch“-Größen und Lernraten (LR)

Das trainierte $CNN_{Progress}$ wurde anschließend anhand der im vorherigen Punkt beschriebenen Testkohorte K_{Test} , bestehend aus 24 Slides von cSCC-Exzidaten mit insgesamt jeweils 10054 Tiles, die durch das CNN_{Tumor} als „Tumor“ klassifiziert wurden, evaluiert. Dabei wurde durch das $CNN_{Progress}$ pro Tile jeweils eine Wahrscheinlichkeit für das Auftreten eines Progresses (P_1) und eine für das Ausbleiben eines Progresses (P_0) ausgegeben. Im Vergleich zu der Tumordetektion lagen P_1 und P_0 pro Tile insgesamt häufiger näher beieinander, sodass eine niedrigere Entscheidungssicherheit vorlag (s. Abbildung 16).

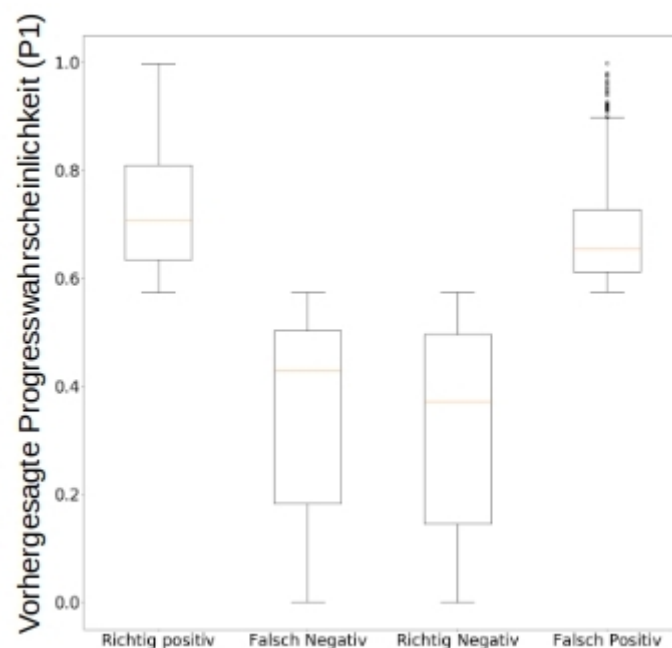


Abbildung 16: Verteilung der Progresswahrscheinlichkeit P_1

Bei einem gewählten Schwellenwert für P_1 , ab dem ein Tile für das Auftreten eines Progresses sprechen soll, von $> 0,5$ wurden 2486 Tiles als richtig-positiv, 3772 Tiles als richtig-negativ, 2204 Tiles als falsch-positiv und 1592 Tiles als falsch-negativ klassifiziert. Dies ergab eine Sensitivität von 61 %, eine Spezifität von 63 %, einen PPW von 53 % und einen NPW von 70 % (s. Abbildung 17a und 17b). Anschließend erfolgte die Bestimmung eines möglicherweise idealen Schwellenwerts für P_1 mittels Erstellung einer ROC-Kurve (s. Abbildung 18) sowie Berechnung des Youden-Index. Dieser betrug 0,575, womit 1918 Tiles als richtig-positiv, 4707 Tiles als richtig-negativ, 1269 Tiles als falsch-positiv und 2160 Tiles als falsch-negativ klassifiziert wurden. Dies ergab eine Sensitivität von 47 %, eine Spezifität von 79 %, einen PPW von 60 % und einen NPW von 69 % (s. Abbildung 17c und 17d). Die Erstellung einer ROC-Kurve ergab eine AUC von 0,66. Aufgrund der höheren Sensitivität und bei einem Schwellenwert von 0,5, wurde dieser für die Klassifikation der Tiles zur Beurteilung ganzer WSI verwendet.

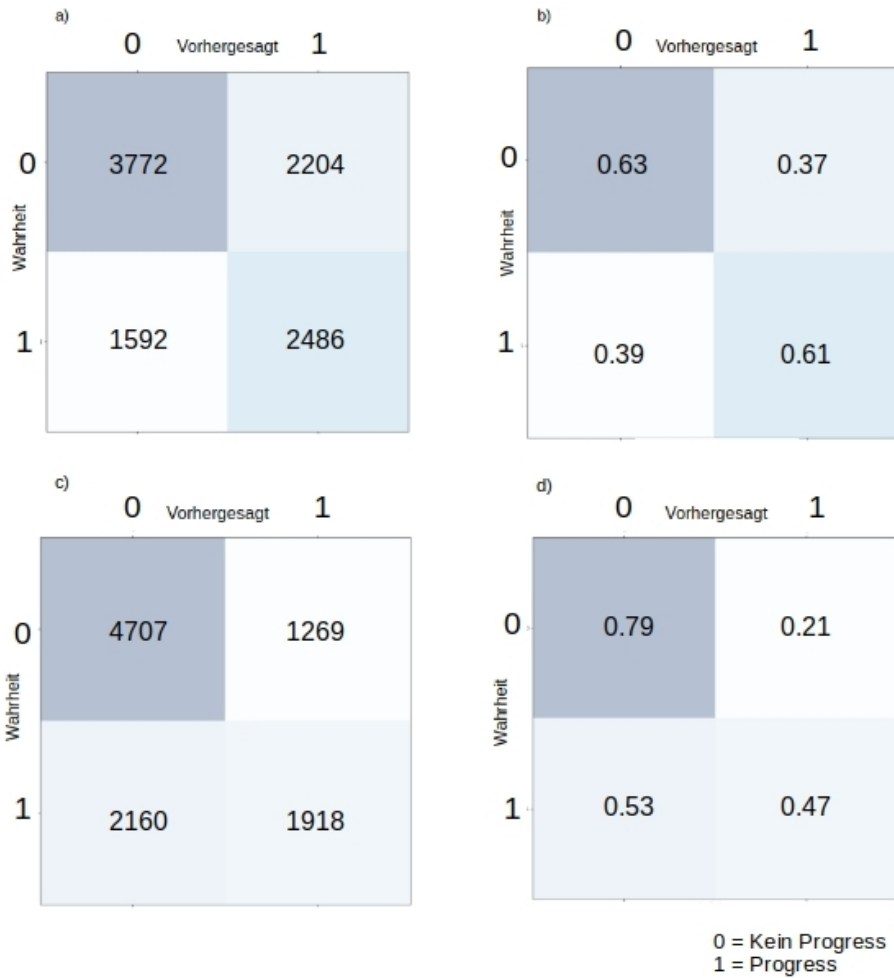


Abbildung 17: Klassifikation der Tiles durch das $CNN_{Progress}$ bei $P1 = 0,5$ absolut (a) und prozentual (b). Klassifikation der Tiles durch das $CNN_{Progress}$ bei $P1 = 0,575$ absolut (c) und prozentual (d)

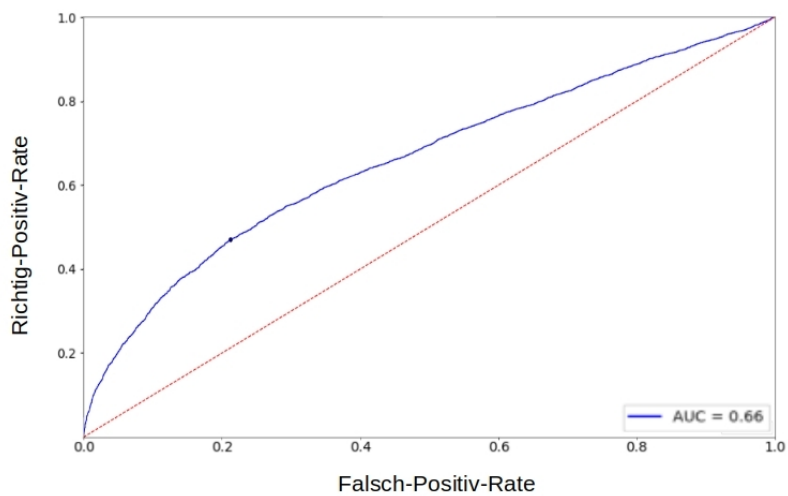


Abbildung 18: ROC-Kurve der Progressprädiktion

4.2.3. Klassifikation ganzer Slides durch CNN_{Progress}

Unter der Annahme, dass auf einem cSCC-Exzidat eines Patienten, der im Verlauf einen Progress entwickelt, nicht alle Tiles innerhalb des Tumors diesbezüglich hinweisend sein müssen, wurde untersucht, ab welchem Anteil von „Progress“-Tiles ein ganzes WSI ein hohes Risiko für das Auftreten eines Progresses in sich birgt. Dies erfolgte mittels Berechnung des Youden-Index, welcher 0,554 betrug.

Damit wurden 5 Slides als richtig-positiv, 10 Slides als richtig-negativ, 2 Slides als falsch-positiv und 7 Slides als falsch-negativ klassifiziert (s. Abbildung 19). Dies ergab eine Sensitivität von 42 %, eine Spezifität von 83 %, einen PPW von 71 % und einen NPW von 59 %.

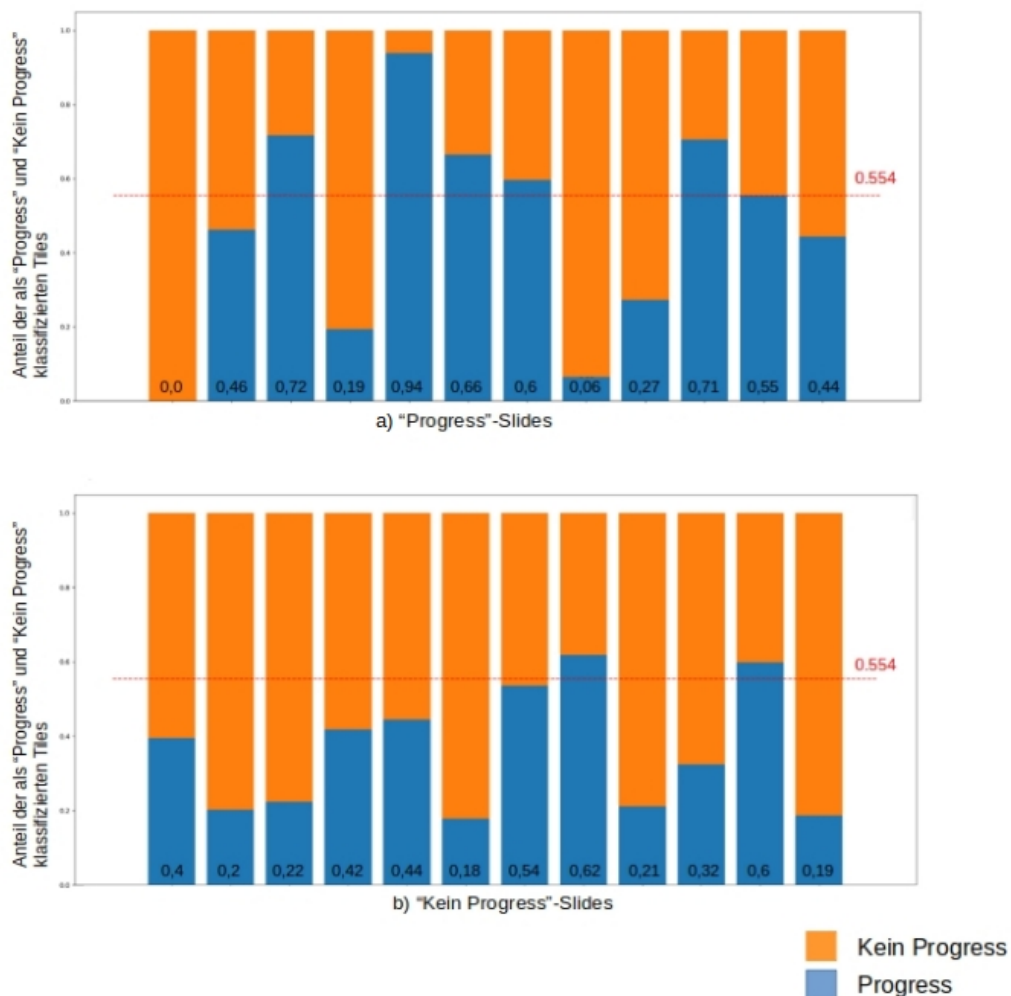


Abbildung 19: Durch das CNN_{Progress} vorhergesagter Anteil an "Progress"- und "Kein Progress"-Tiles auf Slides von Patienten mit (a) und ohne (b) Auftreten eines Progresses im Verlauf mit Markierung des Youden-Index von 0,554

Abschließend erfolgte die Markierung der Tiles auf den WSI, welche jeweils für und gegen das Auftreten eines Progresses sprachen (s. Abbildung 20). Bereiche, die nicht markiert sind, wurden vorher mittels Ausschluss des Hintergrunds sowie der tumorfreen Bereiche mithilfe des CNN_{Tumor} aussortiert, sodass sich die Markierungen hauptsächlich auf den Tumorbereich des Exzidats begrenzen. Dabei verteilen sich die Tiles, die für einen Progress sprechen, über den gesamten Tumorbereich ohne ein Muster erkennen zu lassen.

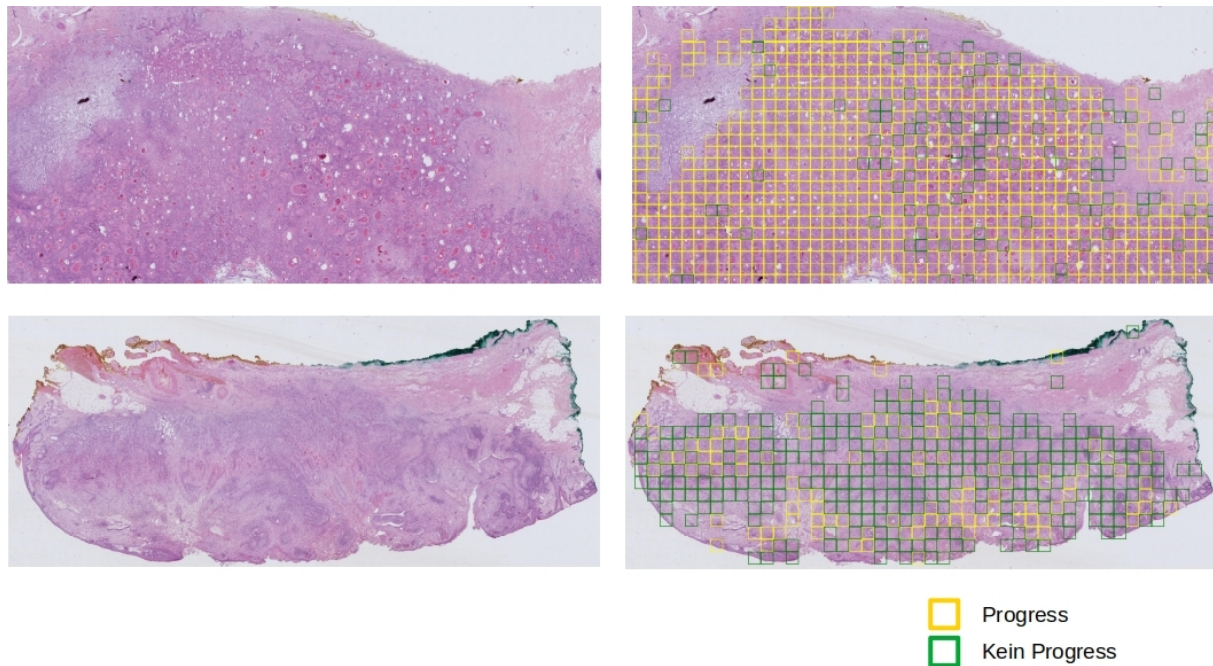


Abbildung 20: Markierung der durch das $CNN_{Progress}$ klassifizierten Tiles

4.2.4. Klinische Merkmale als Prädiktoren für einen Progress

Zusätzlich zu dem Training des $CNN_{Progress}$ zur Vorhersage des Auftretens eines Progresses anhand histomorphologischer Merkmale, wurde untersucht, inwiefern dies anhand von klinischen Parametern möglich ist. Dabei wurden aus den Kohorten K_{Train} und K_{Test} sowohl klinische Merkmale der Patienten als auch Merkmale des Tumors untersucht. Die Ergebnisse wurden in Tabelle 3 zusammengefasst.

Bei einem gewählten Signifikanzniveau von $\alpha = 0,05$ konnte eine signifikante Erhöhung des Risikos für das Auftreten eines Progresses bei Vorliegen einer Perineuralscheideninvasion (RR = 2,13, OR = ∞ , p-Wert = 0,042) beobachtet werden. Zudem wiesen die cSCC, bei denen im Verlauf ein Progress auftrat, eine signifikant größere Tumordicke (Median = 4,75 mm, p-Wert = 0,017) auf im Vergleich zu den cSCC, bei denen im Verlauf kein Progress auftrat (Median = 2,75 mm).

Zu einer nicht-signifikanten Erhöhung des Risikos für das Auftreten eines Progresses führte das männliche Geschlecht (RR = 1,28, OR = 1,61, p-Wert = 0,555), das Vorliegen einer Ulzeration (RR = 1,43, OR = 2,29, p-Wert = 0,301), eines infundibulozystischen Wachstums (RR = 1,35, OR = 2,04, p-Wert = 0,842), eines Diabetes mellitus (RR = 1,16, OR = 1,37, p-Wert = 0,791), einer Psoriasis (RR = 1,53, OR = 3,12, p-Wert = 0,595), einer vaskulären Invasion (RR = 2,06, OR = ∞, p-Wert = 0,214) sowie einer Invasion über das subkutane Gewebe hinaus (RR = 1,91, OR = 9,22, p-Wert = 0,051). Es konnte ein nicht-signifikant höheres Risiko für einen Progress bei der Lokalisation des cSCC am Ohr sowie der Lippe beobachtet werden (p-Wert = 0,76). Die Patienten, die im Verlauf einen Progress entwickelten, hatten zudem einen nicht-signifikant größeren Tumordurchmesser (Median = 2,0 cm, p-Wert = 0,122) im Vergleich zu den Patienten ohne Progress im Verlauf (Median = 1,5 cm). Der Sicherheitsabstand nach Exzision sowie das Vorliegen einer Desmoplasie hatten in unserer Kohorte keinen Einfluss auf das Progressionsrisiko. Bei Patienten mit einer Immunsuppression traten überraschenderweise nicht-signifikant weniger Progressse auf (RR = 0,76, OR = 0,6, p-Wert = 0,462).

	Risikofaktoren	Epidemiologische Maßzahlen
Erhöhtes Progressrisiko (statistisch signifikant)	Perineuralscheideninvasion Tumordicke	RR = 2,13, OR = ∞, p-Wert ¹ = 0,042 Median = 4,75 mm, p-Wert ² = 0,017
Erhöhtes Progressrisiko (statistisch nicht-signifikant)	Männliches Geschlecht Ulzeration Infundibulozystisches Wachstums Diabetes mellitus Psoriasis Vaskuläre Invasion Invasion über das subkutane Gewebe hinaus Lokalisation an Ohr oder Lippe Tumordurchmesser	RR = 1,28, OR = 1,61, p-Wert ¹ = 0,555 RR = 1,43, OR = 2,29, p-Wert ¹ = 0,301 RR = 1,35, OR = 2,04, p-Wert ¹ = 0,842 RR = 1,16, OR = 1,37, p-Wert ¹ = 0,791 RR = 1,53, OR = 3,12, p-Wert ¹ = 0,595 RR = 2,06, OR = ∞, p-Wert ¹ = 0,214 RR = 1,91, OR = 9,22, p-Wert ¹ = 0,051 AR _{Ohr} = 0,75, AR _{Lippe} = 0,75, p-Wert ¹ = 0,76 Median = 2,0 cm, p-Wert ² = 0,122
Kein erhöhtes Progressrisiko	Desmoplasie Immunsuppression Sicherheitsabstand nach Exzision	RR = 1,0, OR = 1,0, p-Wert ¹ = 1,0 RR = 0,76, OR = 0,60, p-Wert ¹ = 0,462 Median = 0,5 cm, p-Wert ² = 0,657

¹Chi-Quadrat-Test

²Mann-Whitney-U-Test

AR = Absolutes Risiko, OR = Odds Ratio, RR = Relatives Risiko

Tabelle 3: Risikofaktoren für das Auftreten eines Progresses bei cSCC in unserer Kohorte

Zudem wurde ein Regressionsmodell entwickelt, welches basierend auf den erhobenen klinischen Parametern eine Vorhersage bezüglich der Progresswahrscheinlichkeit (P_1) vornehmen kann. Bei einem gewählten Schwellenwert für P_1 von 0,5 konnte eine Validierungsgenauigkeit von 0,55 mit einer Sensitivität von 36 % und einer Spezifität von 73 % erreicht werden. Der PPW betrug 57 % und der NPW 53 % (s. Abbildung 21a). Die AUC betrug 0,7 (s. Abbildung 22). Da dieses Regressionsmodell keine gute Vorhersagekraft bezüglich des Auftretens eines Progresses aufweist, sollte auch hier wie bereits bei den

CNN ein idealer Schwellenwert für P_1 errechnet werden, ab dem von einem Progress ausgegangen werden sollte. Dieser betrug 0,185. Damit konnte eine Sensitivität von 100 % und eine Spezifität von 55 % erreicht werden, mit einem PPW von 69 % (s. Abbildung 21b).

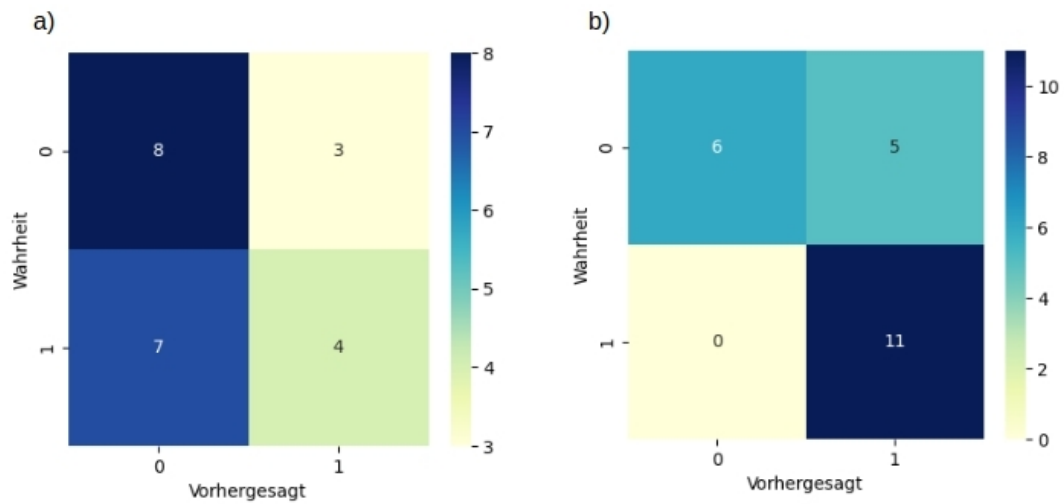


Abbildung 21: Testen des Regressionsmodells: Klassifikation der Tiles durch das Modell bei $P_1 = 0,5$ (a) und $P_1 = 0,185$ (b)

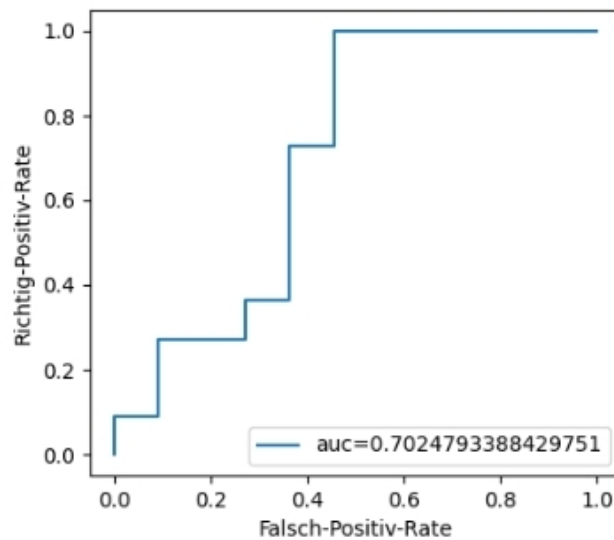


Abbildung 22: ROC-Kurve der Progressprädiktion anhand des Regressionsmodells

5. Diskussion

5.1. Sichtbare histomorphologische Merkmale ermöglichen eine zuverlässige Erkennung von Gewebe durch ein neuronales Netz

In meiner Arbeit konnte mittels Transferlernens ein sehr sensitives sowie gleichermaßen spezifisches neuronales Netz zur Tumordetektion trainiert werden. Dies lag am ehesten daran, dass sich Tumor- und Normalgewebe bezüglich der Zell- und Zellkernmorphologie sowie auch der Gewebearchitektur eindeutig unterscheiden. Da neuronale Netze sichtbar unterschiedliche Muster gut differenzieren können, machte dies das Training trotz einer relativ kleinen Trainingskohorte vergleichsweise einfach.

Auffallend war, dass die falsch-positiv klassifizierten Tiles hauptsächlich das Epithel mit Hautanhangsgebilden und deutlich seltener das Fett- und Bindegewebe betrafen. Dies ist aufgrund des Ursprungs von cSCC und somit histomorphologischer Ähnlichkeit mit normalem Epithel keinesfalls überraschend, jedoch durch Verwendung eines anderen Trainingssets, welches auch Schnitte normaler Haut und somit auch mehr gesundes Epithel beinhaltet, vermutlich reduzierbar. Das Training in meiner Arbeit erfolgte wie beschrieben ausschließlich anhand von cSCC-Exzidaten, auf denen zwar an den Rändern auch normales Epithel abgebildet war, jedoch nicht in dem Maße, wie es auf Schnitten normaler Haut der Fall wäre. Die Hinzunahme von Normalhaut-Slides in die Trainingskohorte würde somit durch einen größeren Anteil an regelrechtem Epithel voraussichtlich zu einer noch höheren Spezifität führen.

Zur Differenzierung ganzer Slides bezüglich des Vorhandenseins von Tumorgewebe wurde errechnet, dass ab einem Anteil von 21,7 % klassifizierter „Tumor“-Tiles ein ganzes WSI als „Tumor-enthaltend“ eingestuft werden sollte. Hinter der Berechnung eines solchen Schwellenwerts steht die Annahme, dass auch auf Normalhaut-Slides immer Tiles sein werden, die falsch-positiv klassifiziert werden und bei realem Vorhandensein eines Tumors, dieser auch mit hoher Wahrscheinlichkeit einen gewissen Mindestanteil des Slides einnimmt. Jedoch wirkt ein Schwellenwert von 21,7 % als Grundlage für das Treffen einer klinischen Entscheidung bezüglich des Vorhandenseins von Tumorgewebe sehr hoch. Zudem muss berücksichtigt werden, dass die Errechnung dieses Schwellenwerts anhand von nur insgesamt 21 Slides erfolgte und sicherlich einer Evaluation an einer größeren Kohorte bedarf. Des Weiteren stellt das Ergebnis des Netzes nur eine erste Einschätzung dar und sollte in jedem Fall von einem Dermatopathologen revidiert werden, denn auch bei einem Anteil klassifizierter „Tumor“-Tiles unterhalb des Schwellenwerts kann selbstverständlich ein kleiner Tumorschnitt abgebildet sein. Hier ist die Markierung der „Tumor“-Tiles auf dem

WSI sinnvoll, da so eine gezielte Betrachtung und Re-Evaluation durch einen Dermatopathologen erfolgen kann.

Das Trainieren eines neuronalen Netzes zur Vorhersage von Progressen gestaltete sich im Vergleich zur Tumordetektion problematischer. Bezüglich prognostischer Aussagen anhand von histologischen Slides gibt es kaum dem menschlichen Auge zugängliche histomorphologische Attribute. Merkmale, anhand derer Dermatologen aktuell prognostische Aussagen treffen können, wie zum Beispiel die Tumordicke, der Tumordurchmesser und die Größe des Resektionsrandes, gehen in der aktuell gebräuchlichen Technik zur Anwendung von Deep Learning auf histologische Slides, nämlich der Zerteilung des gesamten WSI und einzelnen Begutachtung der Tiles, verloren. Schnitte im Ganzen zu beurteilen benötigt jedoch eine immense Rechenleistung, die selbst von Rechenclustern in der Regel nicht bereitgestellt werden kann. Die Minimierung der Auflösung der Schnitte zugunsten einer leichteren Verarbeitung ginge dagegen zulasten von Bildinformationen. Die einzigen histomorphologischen Auffälligkeiten, die bei der Zerteilung der Schnitte nicht verloren gehen, sind die Invasion von Perineuralscheiden, Lymph- und Blutgefäßen sowie die Infiltration des Fettgewebes. All diese Merkmale ließen sich auf den Slides der verwendeten Trainingskohorte K_{Train} insgesamt jedoch sehr selten nachweisen (Perineuralscheideninvasion 5,9 %, vaskuläre Invasion 3,6 %, lymphatische Invasion 0 %, Invasion über das subkutane Fettgewebe hinaus 8,3 %), was zur Folge hatte, dass dem neuronalen Netz wenig Trainingsmaterial diesbezüglich zur Verfügung stand und deshalb diese Strukturen vermutlich wenig zur Entscheidungsfindung beigetragen haben, zumal selbst bei Vorkommen eines dieser Merkmale auf einem Slide, dieses nur auf wenigen Tiles abgebildet war.

Abschließend erfolgte die Markierung der Tiles, die für das Auftreten eines Progresses sprachen, auf den WSI. Dabei konnte augenscheinlich keine bevorzugte Markierung bestimmter Gewebearten oder Bereiche innerhalb des Tumors beobachtet werden. Stattdessen waren die Tiles eher ohne erkennbares Muster über den ganzen Tumor verteilt. Eine weitergehende Analyse der Histomorphologie der „Progress“-Tiles kann bezüglich der Eingrenzung der relevanten Gewebebereiche innerhalb des Tumors gegebenenfalls aufschlussreich sein.

Es ist an sich nicht ausgeschlossen, dass ein neuronales Netz Strukturen erkennt und folglich zu unterscheiden lernt, die dem menschlichen Auge nicht zugänglich sind. Poplin et al.⁵⁹ zeigten beispielsweise, dass ein neuronales Netz in der Lage war, anhand von Bildern von Augenhintergründen auf unter anderem das Geschlecht und das Alter des jeweiligen Patienten zu schließen, obwohl dies dem Ophthalmologen nicht möglich ist und auch bis

dato nicht bekannt war, dass fundoskopische Bilder diese Informationen beinhalten. Zudem konnten bereits in ähnlichen Arbeiten wie der Meinen erfolgversprechende Ergebnisse geliefert werden. So schafften es Kulkarni et al.³⁶ an Schnitten von Exzidaten maligner Melanome prognostische Aussagen bezüglich Überleben und Progresswahrscheinlichkeit zu treffen. Coudray et al.³⁴ konnten zudem anhand von histologischen Schnitten von nicht-kleinzelligen Lungenkarzinomen auf Mutationen des Tumors schließen. Prinzipiell scheint es somit für ein neuronales Netz möglich zu sein, Strukturen zu erlernen, die für uns Menschen nicht sichtbar sind, und anhand derer prognostische Aussagen zu tätigen. Ein Grund, aus dem in meiner Arbeit dies nicht ausreichend gelang, könnte die limitierte Anzahl an Progressen innerhalb der Kohorte sein. Trotz einer Gesamtgröße von 1241 Patienten traten nur bei 96 Patienten ein Progress auf, wovon uns nur 54 Slides zur Verfügung standen. Nach Zurückhalten eines Testsets konnten schließlich lediglich 42 Slides für das Training verwendet werden. Eine größere Trainingskohorte könnte vermutlich zu besseren Ergebnissen führen. Dennoch ist an dieser Stelle sicherlich der Miteinbezug von klinischen Parametern in die Entscheidungsfindung des neuronalen Netzes oder auch die Entwicklung und Anwendung von Scores hilfreich und sinnvoll.

5.2. Vorhersage von Progressen anhand von klinischen Risikofaktoren

In meiner Arbeit konnten zwei Faktoren ermittelt werden, die das Risiko für einen Progress signifikant erhöhen, nämlich das Vorliegen einer Perineuralscheideninvasion sowie eine höhere Tumordicke. Beide Risikofaktoren sind bereits in der Literatur als solcher aufgeführt^{3,8} und konnten somit bestätigt werden. Weitere in der Literatur genannte Risikofaktoren^{3,8-11}, die in unserer Kohorte ebenfalls mit einem erhöhten Risiko für das Auftreten eines Progresses einhergingen, jedoch ohne statistische Signifikanz, waren das Vorliegen einer vaskulären Invasion oder einer Invasion über das Fettgewebe hinaus, die Lokalisation am Ohr oder der Lippe sowie ein größerer Tumordurchmesser. Das Vorliegen einer Desmoplasie sowie der Sicherheitsabstand nach Exzision des Tumors hatten in unserer Kohorte keinen Einfluss auf das Auftreten eines Progresses, sind jedoch als Risikofaktoren beschrieben^{3,8}.

Zusätzlich identifizierte, bisher weniger beschriebene, nicht-signifikante Risikofaktoren waren das Vorliegen von Ulzerationen, infundibulozystischen Wachstums, das männliche Geschlecht sowie die Komorbidität mit Diabetes mellitus oder einer Psoriasis.

Das Vorliegen einer Immunsuppression wird in der Literatur ebenfalls als Risikofaktor beschrieben^{8,9}, wogegen in unserer Kohorte überraschenderweise nicht-signifikant weniger Progressen bei immunsupprimierten Patienten auftraten.

Bisher werden Patienten mit cSCC je nach Vorliegen der genannten Risikofaktoren in zwei Risikogruppen für das Auftreten eines Progresses eingeteilt. Eine konkrete, einheitliche Einteilung fehlt jedoch bisher in der aktuellen S3-Leitlinie³. Mit dem Ziel einer genaueren Risikostratifizierung wurde dafür in meiner Arbeit ein Regressionsmodell entwickelt, welches basierend auf den erhobenen klinischen Merkmalen eine Wahrscheinlichkeit für das Auftreten eines Progresses ausgibt. Da bei einem gewählten Cut-Off von 0,5 für die Progresswahrscheinlichkeit (P_1) pro Patient keine gute Aussagekraft bezüglich des Auftretens eines Progresses vorlag, war das weitere Ziel einen optimalen Schwellenwert für P_1 zu finden. Bei einem errechneten idealen Schwellenwert für P_1 von 0,185 wurde zwar eine Sensitivität von 100 % erreicht, jedoch zulasten der Spezifität, welche nur noch 55 % betrug. Eine Risikostratifizierung ist anhand dessen somit zwar noch nicht erreicht, bildet jedoch möglicherweise einen Ansatz, der mithilfe einer größeren Kohorte, insbesondere in Bezug auf die Zahl der Progressen, sowie durch Hinzunahme weiterer Faktoren wie beispielsweise dem klinischen Lymphknotenstatus noch ausgebaut werden könnte.

5.3. Anwendbarkeit von neuronalen Netzen zur automatisierten Analyse von Slides im klinischen Alltag

Innerhalb der letzten Jahre wurden zunehmend Arbeiten veröffentlicht, in denen neuronale Netze Tumoren auf histologischen Schnitten erkennen^{27,33}, durch Analyse der Histomorphologie prognostische Aussagen treffen³⁶ oder auf Mutationen rückschließen können^{34,35}. Dabei ist, wie beschrieben, die Verarbeitung der gescannten Schnitte häufig sehr aufwendig und besteht aus vielen Zwischenschritten. Für die Anwendung in der klinischen Praxis müssten diese möglichst automatisiert ablaufen. Das zunächst notwendige Einscannen der Slides könnte zwar in Zukunft in Hinblick auf eine immer digitalisiertere Medizin bereits ohnehin alltäglich werden. Die benötigte Rechenleistung für die Klassifikation von histologischen Schnitten ist jedoch vergleichbar hoch wie für das Training des neuronalen Netzes, da der zeitintensivste Schritt, nämlich die Verarbeitung der Tiles in den CL und PL des CNN, auch für die Prädiktion durchlaufen werden muss. Insbesondere, wenn mehrere Schnitte in einem Durchlauf beurteilt werden sollen, stellt dies eine hohe Anforderung an die Rechenkapazität des jeweiligen Computers dar. Um den Prozess zu beschleunigen, ist es möglich, die Berechnungen auf der Grafikkarte durchführen zu lassen, was jedoch nicht auf jedem Computer möglich ist. Alternativ kann, wie in meiner Arbeit, ein Rechencluster mit entsprechend ausreichendem Arbeitsspeicher verwendet werden. Diese stehen jedoch nicht überall und jeder Klinik zur Verfügung.

Nach Diskussion der Praktikabilität muss anschließend jedoch auch der Umgang mit den Ergebnissen des neuronalen Netzes diskutiert werden. Wie bereits in Punkt 5.1 erwähnt, sollten die Prädiktionen neuronaler Netze die Arbeit der Ärzte nicht ersetzen und klinische Entscheidungen nicht allein auf Basis ihrer Klassifikationen erfolgen. Vielmehr sollen sie ein unterstützender Faktor sein. Eine Vorverarbeitung von Slides durch ein CNN inklusive beispielsweise Markierung des Tumorbereichs könnte für Ärzte äußerst hilfreich und zeitsparend sein.

Bezüglich einer zuverlässigen Vorhersage von Progressen ist dies bisher nicht allein durch die Analyse der histologischen Schnitte durch ein neuronales Netz gelungen. Die limitierenden Faktoren wurden in Punkt 5.1 bereits diskutiert. Hierbei könnte auch ein multifaktorieller Ansatz helfen, in dem klinische Risikofaktoren ebenso wie das histologische Bild gemeinsam von einem neuronalen Netz verarbeitet werden. Inwieweit dadurch die Entwicklung eines neuronalen Netzes gelingt, das alltagsrelevante Hilfestellung in der Risikostratifizierung gibt, müsste in weiteren Studien mit einem ausreichend großen Anteil an Progressen untersucht werden.

Insgesamt ist das Potenzial von künstlicher Intelligenz in der Anwendung des klinischen Alltags zur unterstützenden Entscheidungsfindung von klinisch tätigen Ärzten hoch, auch wenn die Notwendigkeit der kritischen Evaluation der Ergebnisse durch einen erfahrenen Mediziner notwendig bleiben wird. Zudem müssen ausreichend leistungsfähige Computer in den Kliniken zur Verfügung stehen, um die benötigte Rechenleistung gewährleisten zu können.

6. Literaturverzeichnis

- 1 Robert-Koch-Institut. Krebs in Deutschland für 2017/2018. 2021 https://www.krebsdaten.de/Krebs/DE/Content/Publikationen/Krebs_in_Deutschland/kid_2021/krebs_in_deutschland_2021.pdf;jsessionid=0F0D293060809C27E3C5D5C3E134212D.internet111?__blob=publicationFile.
- 2 Tumorregister München. Überleben cSCC: Plattenepithelca. Haut [Internet]. 2022. https://www.tumorregister-muenchen.de/facts/surv/sCSCC_G-cSCC-Plattenepithelca.-Haut-Survival.pdf (accessed June 28, 2022).
- 3 Berking C, Steeb T, Leiter U, Heppt M, Garbe C. S3-Leitlinie „Aktinische Keratose und Plattenepithelkarzinom der Haut“: Leitlinienprogramm Onkologie, Deutsche Krebsgesellschaft, Deutsche Krebshilfe, AWMF: Langversion 1.0, 2019, AWMF-Register-Nr.: 032/022OL. *Forum* 2020; **35**: 93–9.
- 4 Stratigos A, Garbe C, Lebbe C, *et al.* Diagnosis and treatment of invasive squamous cell carcinoma of the skin: European consensus-based interdisciplinary guideline. *Eur J Cancer* 2015; **51**: 1989–2007.
- 5 [libtayo-epar-product-information_de.pdf](https://www.ema.europa.eu/en/documents/product-information/libtayo-epar-product-information_de.pdf). https://www.ema.europa.eu/en/documents/product-information/libtayo-epar-product-information_de.pdf (accessed Dec 30, 2022).
- 6 [pembrolizumab-mono-plattenepithelkarzinom-kopf-hals-bereich-dgho_dghno_dgmkg-stellungnahme-20200323.pdf](https://www.dgho.de/publikationen/stellungnahmen/fruehenutzenbewertung/pembrolizumab/pembrolizumab-mono-plattenepithelkarzinom-kopf-hals-bereich-dgho_dghno_dgmkg-stellungnahme-20200323.pdf). https://www.dgho.de/publikationen/stellungnahmen/fruehenutzenbewertung/pembrolizumab/pembrolizumab-mono-plattenepithelkarzinom-kopf-hals-bereich-dgho_dghno_dgmkg-stellungnahme-20200323.pdf (accessed Dec 30, 2022).
- 7 Peris K, Piccerillo A, Regno LD, Stefani AD. Treatment approaches of advanced cutaneous squamous cell carcinoma. *Journal of the European Academy of Dermatology and Venereology* 2021; **36**: 19–22.
- 8 Brantsch KD, Meisner C, Schönfisch B, *et al.* Analysis of risk factors determining prognosis of cutaneous squamous-cell carcinoma: a prospective study. *Lancet Oncol* 2008; **9**: 713–20.
- 9 Thompson AK, Kelley BF, Prokop LJ, Murad MH, Baum CL. Risk Factors for Cutaneous Squamous Cell Carcinoma Recurrence, Metastasis, and Disease-Specific Death: A Systematic Review and Meta-analysis. *JAMA Dermatol* 2016; **152**: 419–28.
- 10 Nuño-González A, Vicente-Martín FJ, Pinedo-Moraleda F, López-Estebanz JL. High-Risk Cutaneous Squamous Cell Carcinoma. *Actas Dermo-Sifiliográficas (English Edition)* 2012; **103**: 567–78.
- 11 Mourouzis C, Boynton A, Grant J, *et al.* Cutaneous head and neck SCCs and risk of nodal metastasis - UK experience. *J Craniomaxillofac Surg* 2009; **37**: 443–7.
- 12 Rowe DE, Carroll RJ, Day CL. Prognostic factors for local recurrence, metastasis, and survival rates in squamous cell carcinoma of the skin, ear, and lip. Implications for treatment modality selection. *J Am Acad Dermatol* 1992; **26**: 976–90.
- 13 Dobrev D. A definition of artificial intelligence. *arXiv preprint arXiv:12101568* 2012.

- 14 Janiesch C, Zschech P, Heinrich K. Machine learning and deep learning. *Electron Markets* 2021; **31**: 685–95.
- 15 Rashid T, Langenau F. Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python. O'Reilly, 2017 <https://books.google.de/books?id=b9N3DwAAQBAJ>.
- 16 Albawi S, Mohammed TA, Al-Zawi S. Understanding of a convolutional neural network. In: 2017 International Conference on Engineering and Technology (ICET). 2017: 1–6.
- 17 LeCun Y, Bengio Y, Hinton G. Deep Learning. *Nature* 2015; **521**: 436–44.
- 18 Yamashita R, Nishio M, Do RKG, Togashi K. Convolutional neural networks: an overview and application in radiology. *Insights Imaging* 2018; **9**: 611–29.
- 19 Jie HJ, Wanda P. RunPool: A Dynamic Pooling Layer for Convolution Neural Network. *Int J Comput Intell Syst* 2020; **13**: 66–76.
- 20 Grodzicki R, Mańdziuk J, Wang L. Improved multilabel classification with neural networks. Springer, 2008: 409–16.
- 21 Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:12070580* 2012.
- 22 Rumelhart D, Hinton GE, Williams RJ. Learning representations by back-propagating errors. *Nature* 1986. <https://www.semanticscholar.org/paper/Learning-representations-by-back-propagating-errors-Rumelhart-Hinton/052b1d8ce63b07fec3de9dbb583772d860b7c769> (accessed Jan 6, 2023).
- 23 Team Keras. Keras documentation: Model training APIs. https://keras.io/api/models/model_training_apis/ (accessed Oct 3, 2022).
- 24 Radiuk PM. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. 2017; published online Dec 1. <http://elar.khmnu.edu.ua/jspui/handle/123456789/11047> (accessed Oct 3, 2022).
- 25 Konar J, Khandelwal P, Tripathi R. Comparison of Various Learning Rate Scheduling Techniques on Convolutional Neural Network. In: 2020 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS). 2020: 1–5.
- 26 Zhuang F, Qi Z, Duan K, et al. A Comprehensive Survey on Transfer Learning. *Proc IEEE* 2021; **109**: 43–76.
- 27 Ferreira CA, Melo T, Sousa P, et al. Classification of Breast Cancer Histology Images Through Transfer Learning Using a Pre-trained Inception Resnet V2. In: Campilho A, Karray F, ter Haar Romeny B, eds. Image Analysis and Recognition. Cham: Springer International Publishing, 2018: 763–70.
- 28 Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014; published online Sept 4. DOI:10.48550/arXiv.1409.1556.
- 29 He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. 2016: 770–8.
- 30 Szegedy C, Liu W, Jia Y, et al. Going Deeper With Convolutions. 2015: 1–9.
- 31 ImageNet. <https://www.image-net.org/challenges/LSVRC/> (accessed Jan 6, 2023).

- 32 ImageNet. <https://www.image-net.org/about.php> (accessed Jan 6, 2023).
- 33 Yari Y, Nguyen TV, Nguyen HT. Deep Learning Applied for Histological Diagnosis of Breast Cancer. *IEEE Access* 2020; **8**: 162432–48.
- 34 Coudray N, Moreira AL, Sakellaropoulos T, Fenyö D, Razavian N, Tsirigos A. Classification and Mutation Prediction from Non-Small Cell Lung Cancer Histopathology Images using Deep Learning. *Cancer Biology*, 2017 DOI:10.1101/197574.
- 35 Fu Y, Jung AW, Torne RV, *et al.* Pan-cancer computational histopathology reveals mutations, tumor composition and prognosis. *Nat Cancer* 2020; **1**: 800–10.
- 36 Kulkarni PM, Robinson EJ, Sarin Pradhan J, *et al.* Deep Learning Based on Standard H&E Images of Primary Melanoma Tumors Identifies Patients at Risk for Visceral Recurrence and Death. *Clinical Cancer Research* 2020; **26**: 1126–34.
- 37 TRACERx Consortium, AbdulJabbar K, Raza SEA, *et al.* Geospatial immune variability illuminates differential evolution of lung adenocarcinoma. *Nat Med* 2020; **26**: 1054–62.
- 38 Kather JN, Pearson AT, Halama N, *et al.* Deep learning can predict microsatellite instability directly from histology in gastrointestinal cancer. *Nat Med* 2019; **25**: 1054–6.
- 39 Klein S, Quaas A, Quantius J, *et al.* Deep Learning Predicts HPV Association in Oropharyngeal Squamous Cell Carcinomas and Identifies Patients with a Favorable Prognosis Using Regular H&E Stains. *Clinical Cancer Research* 2021; **27**: 1131–8.
- 40 Macenko M, Niethammer M, Marron JS, *et al.* A method for normalizing histology slides for quantitative analysis. In: 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro. Boston, MA, USA: IEEE, 2009: 1107–10.
- 41 Reinhard E, Adhikhmin M, Gooch B, Shirley P. Color transfer between images. *IEEE Comput Grap Appl* 2001; **21**: 34–41.
- 42 Vahadane A, Peng T, Sethi A, *et al.* Structure-Preserving Color Normalization and Sparse Stain Separation for Histological Images. *IEEE Trans Med Imaging* 2016; **35**: 1962–71.
- 43 Shorten C, Khoshgoftaar TM. A survey on Image Data Augmentation for Deep Learning. *J Big Data* 2019; **6**: 1–48.
- 44 Welcome to QuPath! — QuPath 0.2.3 documentation. <https://qupath.readthedocs.io/en/0.2/index.html> (accessed Oct 12, 2022).
- 45 Python Release Python 3.5.10. Python.org. <https://www.python.org/downloads/release/python-3510/> (accessed Oct 20, 2022).
- 46 Python Release Python 3.9.13. Python.org. <https://www.python.org/downloads/release/python-3913/> (accessed Oct 20, 2022).
- 47 pandas - Python Data Analysis Library. <https://pandas.pydata.org/> (accessed Oct 20, 2022).
- 48 large-image. PyPI. <https://pypi.org/project/large-image/> (accessed Oct 20, 2022).
- 49 Pillow. PyPI. <https://pypi.org/project/Pillow/> (accessed Oct 20, 2022).
- 50 Keras: the Python deep learning API. <https://keras.io/> (accessed Oct 20, 2022).

- 51 17.2. multiprocessing — Process-based parallelism — Python 3.5.9 documentation. <https://docs.python.org/3.5/library/multiprocessing.html> (accessed Oct 20, 2022).
- 52 Matplotlib — Visualization with Python. <https://matplotlib.org/> (accessed Oct 20, 2022).
- 53 scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation. <https://scikit-learn.org/stable/> (accessed Oct 20, 2022).
- 54 NumPy. <https://numpy.org/> (accessed Oct 20, 2022).
- 55 RRZK: HPC. <https://rrzk.uni-koeln.de/hpc-projekte/hpc> (accessed Oct 12, 2022).
- 56 Exporting annotations — QuPath 0.2.3 documentation. https://qupath.readthedocs.io/en/0.2/docs/advanced/exporting_annotations.html (accessed Oct 14, 2022).
- 57 staintools · PyPI. <https://pypi.org/project/staintools/> (accessed Oct 20, 2022).
- 58 OpenSlide Python — OpenSlide Python 1.2.0 documentation. <https://openslide.org/api/python/> (accessed Oct 20, 2022).
- 59 Poplin R, Varadarajan AV, Blumer K, *et al.* Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nat Biomed Eng* 2018; **2**: 158–64.
- 60 Hasan M, Islam N, Rahman MM. Gastrointestinal Polyp Detection Through a Fusion of Contourlet Transform and Neural Features. *Journal of King Saud University - Computer and Information Sciences* 2020; **34**. DOI:10.1016/j.jksuci.2019.12.013.

7. Anhang

7.1. Abbildungsverzeichnis

Abbildung 1: Architektur des VGG19-Netzes ohne FCL. Adaptiert von Hasan et al. ⁶⁰	15
Abbildung 2: Architektur der hinzugefügten, trainierten Schichten von CNN_{Tumor} und $CNN_{Progress}$	15
Abbildung 3: Funktion der Trainings-, Validierungs- und Testdaten.....	17
Abbildung 4: Übersicht über die Trainingsprozesse von $CNN_{Progress}$ und CNN_{Tumor}	23
Abbildung 5: Manuelle Annotation des Tumorbereichs auf den Slides.....	24
Abbildung 6: Automatisches Aussortieren des Hintergrunds.....	26
Abbildung 7: Trainingsergebnisse von CNN_{Tumor} bei verschiedenen „Batch“-Größen und Lernraten (LR).....	30
Abbildung 8: Klassifikation der Tiles durch das CNN_{Tumor} bei $P1 = 0,5$ absolut (a) und prozentual (b). Klassifikation der Tiles durch das CNN_{Tumor} bei $P1 = 0,278$ absolut (c) und prozentual (d).....	31
Abbildung 9: Verteilung der Tumorwahrscheinlichkeit $P1$	32
Abbildung 10: ROC-Kurve der Tumordetektion.....	33
Abbildung 11: Durch das CNN_{Tumor} vorhergesagter (a) sowie wahrer (b) Anteil an "Tumor"-Tiles pro Slide mit Markierung des Youden-Index von 0,217.....	34
Abbildung 12: Markierung der durch das CNN_{Tumor} als "Tumor" klassifizierten Tiles auf den Slides.....	35
Abbildung 13: Anzahl an Metastasen und Lokalrezidiven in den Kohorten K_{Train} und K_{Test}	36
Abbildung 14: Gesamtüberlebensraten in Abhängigkeit des Auftretens eines Progresses....	36
Abbildung 15: Trainingsergebnisse von $CNN_{Progress}$ bei verschiedenen „Batch“-Größen und Lernraten (LR).....	38
Abbildung 16: Verteilung der Progresswahrscheinlichkeit $P1$	39
Abbildung 17: Klassifikation der Tiles durch das $CNN_{Progress}$ bei $P1 = 0,5$ absolut (a) und prozentual (b). Klassifikation der Tiles durch das $CNN_{Progress}$ bei $P1 = 0,575$ absolut (c) und prozentual (d).....	40
Abbildung 18: ROC-Kurve der Progressprädiktion.....	40
Abbildung 19: Durch das $CNN_{Progress}$ vorhergesagter Anteil an "Progress"- und "Kein Progress"-Tiles auf Slides von Patienten mit (a) und ohne (b) Auftreten eines Progresses im Verlauf mit Markierung des Youden-Index von 0,554.....	41
Abbildung 20: Markierung der durch das $CNN_{Progress}$ klassifizierten Tiles.....	42

Abbildung 21: Testen des Regressionsmodells: Klassifikation der Tiles durch das Modell bei $P1 = 0,5$ (a) und $P1 = 0,185$ (b).....44

Abbildung 22: ROC-Kurve der Progressprädiktion anhand des Regressionsmodells.....44

7.2. Tabellenverzeichnis

Tabelle 1: In der Literatur beschriebene Risikofaktoren für das Auftreten eines Progresses bei cSCC.....	12
Tabelle 2: Deskriptive Statistik der Trainings- und Testkohorten.....	37
Tabelle 3: Risikofaktoren für das Auftreten eines Progresses bei cSCC in unserer Kohorte.	43

7.3. Skripte

7.3.1. Skript 1: Zerteilen der annotierten Slides und Exportieren der Tiles aus QuPath

```
import qupath.lib.images.servers.LabeledImageServer

def imageData = getCurrentImageData()

// Define output path (relative to project)
def name = GeneralTools.getNameWithoutExtension(
imageData.getServer().getMetadata().getName()
)
def pathOutput = buildFilePath("${PATH_TO_OUTPUT}", name)
mkdirs(pathOutput)

// Downsampling.
// 1.0 is original magnification (40x)
// 8.0 is 5x magnification
double downsample = 8.0

// Create an ImageServer where the pixels are derived from annotations
def labelServer = new LabeledImageServer.Builder(imageData)
    // Specify background label (usually 0 or 255)
    .backgroundLabel(0, ColorTools.WHITE)
    // Choose server resolution; this should match
    // the resolution at which tiles are exported
    .downsample(downsample)
    // Choose output labels (the order matters!)
    .addLabel("Tumor", 1)
    .addLabel("Stroma", 2) // "Stroma" here means non-tumor
    // If true, each label is a different channel
    // (required for multiclass probability)
    .multichannelOutput(false)
    .build()

// Create an exporter that requests corresponding tiles
// from the original & labeled image servers
new TileExporter(imageData)
    .downsample(downsample) // Define export resolution
    // Define file extension for original pixels (often .tif, .jpg, .png or .ome.tif)
    .imageExtension(".jpg")
    // Define size of each tile, in pixels
    .tileSize(224)
    // Define the labeled image server to use (i.e. the one we just built)
    .labeledServer(labelServer)
    // If true, only export tiles if there is a (labeled) annotation present
    .annotatedTilesOnly(true)
    // Define overlap, in pixel units at the export resolution
    .overlap(0)
    .writeTiles(pathOutput)

print 'Done!'
```

7.3.2. Skript 2: Sortieren der Annotationen

```
import os

from PIL import Image # type: ignore
from tqdm import tqdm # type: ignore

def count_colors(filename: str) -> tuple:
    img = Image.open(filename)
    img = img.convert("RGB")
    img = list(img.getdata())

    non_tumor: tuple = (150, 200, 150)
    tumor: tuple = (200, 0, 0)
    background: tuple = (255, 255, 255)

    total_count: int = len(img)

    tumor_count: float = img.count(tumor) / total_count * 100
    non_tumor_count: float = img.count(non_tumor) / total_count * 100
    background_count: float = img.count(background) / total_count * 100

    return tumor_count, non_tumor_count, background_count

def classify_images(path: str) -> None:
    files: list = os.listdir(path)
    files_labeled: list = [file for file in files if file[-4:] == ".png"]

    files_raw_image: list = [file for file in files if file[-4:] == ".jpg"]
    files_labeled.sort()
    files_raw_image.sort()

    for directory in ("/tumor", "/non_tumor", "/background", "/not_usable"):
        if not os.path.isdir(path + directory):
            os.mkdir(path + directory)

    for raw_img, labeled_img in tqdm(zip(files_raw_image, files_labeled)):
        if raw_img[-4] not in labeled_img:
            print("not in order")
            break

        tumor_count, non_tumor_count, bg_count = count_colors(
            path + '/' + labeled_img
        )

        if bg_count > 40:
            directory = "/background/"
        elif tumor_count == 0:
            directory = "/non_tumor/"
        elif tumor_count > 30:
            directory = "/tumor/"
        else:
            directory = "/not_usable/"

        new_file_path_raw: str = path + directory + raw_img
        new_file_path_labeled: str = path + directory + labeled_img
```

```

os.rename(path + "/" + raw_img, new_file_path_raw)
os.rename(path + "/" + labeled_img, new_file_path_labeled)

def read_directories(path: str) -> None:
    directories: list = os.listdir(path)
    for directory in tqdm(directories):

```

```

classify_images(path + directory)

```

```

if __name__ == "__main__":
    path: str = "../../../QuPath-0.2.3/tiles_annotated_5x/"
    read_directories(path)

```

7.3.3. Skript 3: Erstellen der Trainingsdaten für CNN_{Tumor}

```

import concurrent.futures
import os
import pickle
import sys
import time
import warnings
from itertools import repeat

import numpy as np
from PIL import Image # type: ignore

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

from keras.applications.vgg19 import VGG19, preprocess_input
from keras.utils import to_categorical

class DataCreator:
    def __init__(self, normalization: str, source_dir: str, target_dir: str):
        self.MAG = 5
        self.NORMALIZATION = normalization
        self.source_dir = source_dir
        self.target_dir = target_dir
        self.label: dict = {"tumor": 1, "non_tumor": 0}
        self.x_train: np.ndarray
        self.x_train_after_vgg: np.ndarray
        self.y_train: np.ndarray
        self.file_names: list = []

    def create_data_lists(self) -> None:
        categories: dict = {"tumor": [], "non_tumor": []}

        # Make 2 lists (tumor/non_tumor) which contain all the images
        # as np.array with their corresponding label and filename
        for key in categories.keys():
            categories[key] = self.open_images(key)

        # Combine them into one list, still attached to their label
        # and filename, in order to shuffle the training data
        train = categories["tumor"] + categories["non_tumor"]

        x_train_ls = []
        y_train_ls = []
        # Split image, label and filename into x, y and z
        for x, y, z in train:
            with warnings.catch_warnings():
                warnings.simplefilter(action='ignore', category=FutureWarning)
                if x == "fail": # may not be possible in further versions
                    continue

```

```

x_train_ls.append(x)
y_train_ls.append(y)
self.file_names.append(z)

# Transforming into np.array
self.x_train = np.asarray(x_train_ls)
self.y_train = np.asarray(y_train_ls).astype("float32")

def open_images(self, key: str) -> list:
    source = self.source_dir + key + "/"

    file_names_ls = [
        file for file in os.listdir(source) if file[-4:] == ".jpg"
    ]
    print("Starting {} files".format(len(file_names_ls), key))

    MAX_WORKERS = int(os.getenv('SLURM_CPUS_PER_TASK'))
    MAX_SIZE_OF_CHUNKS = 48
    NUMBER_OF_CHUNKS = (len(file_names_ls) // MAX_SIZE_OF_CHUNKS) + 1

    file_names = np.asarray(file_names_ls)
    chunks = np.array_split(file_names, NUMBER_OF_CHUNKS)

    start = time.time()
    images = []

    for i in range(NUMBER_OF_CHUNKS):
        with concurrent.futures.ProcessPoolExecutor(
            max_workers=MAX_WORKERS
        ) as executor:
            images += list(executor.map(
                self.process_image, repeat(key),
                repeat(source), chunks[i].tolist()
            ))

    end = time.time()
    print("Processing all images took {} seconds".format(end-start))
    return images

def process_image(self, key: str, source: str, img_filename: str):
    try:

        # WITHOUT Normalization.
        img = Image.open(source + img_filename)
        img = np.asarray(img)
        img = img.reshape(224, 224, 3).astype("float32")

    return (img, to_categorical(self.label[key], 2), img_filename)

```

```

except:
    print("could not process file {}".format(img_filename))
    return ("fail", "fail", "fail")

# Builds x_train_after_vgg.
def preprocess_data(self) -> None:

    SIZE_OF_CHUNKS = 10
    print("x_train shape: {}".format(self.x_train.shape))
    NUMBER_OF_CHUNKS = self.x_train.shape[0] // SIZE_OF_CHUNKS
    MAX_WORKERS = 5
    start = time.time()
    chunks = np.array_split(self.x_train, NUMBER_OF_CHUNKS, axis=0)
    print("chunks have shape of {}".format(chunks[0].shape))
    print("NUMBER OF CHUNKS: {}, SIZE_OF_CHUNKS: {}".format(NUMBER_OF_CHUNKS, SIZE_OF_CHUNKS))

    with concurrent.futures.ProcessPoolExecutor(
        max_workers=MAX_WORKERS
    ) as executor:
        self.x_train_after_vgg = np.asarray(
            list(executor.map(self.vgg_preprocessing,
                             chunks[0].tolist()))
        ).astype("float32")

    for i in range(1, NUMBER_OF_CHUNKS):
        with concurrent.futures.ProcessPoolExecutor(
            max_workers=MAX_WORKERS
        ) as executor:
            x_train_after_vgg_chunk = np.asarray(
                list(executor.map(self.vgg_preprocessing,
                                 chunks[i].tolist()))
            ).astype("float32")

        self.x_train_after_vgg = np.concatenate(
            (self.x_train_after_vgg, x_train_after_vgg_chunk)
        )

    print("Processing data in Convolutional VGG19 done.")
    end = time.time()
    print("Processing in VGG19 took {} seconds.".format(end-start))

def vgg_preprocessing(self, image) -> np.ndarray:
    image = np.asarray(image)

```

```

image = image.reshape(
    1, image.shape[0], image.shape[1], image.shape[2]
)
image = preprocess_input(image)
vgg19_model = VGG19(include_top=False, input_shape=(224, 224, 3))
vgg19_model.trainable = False
image_after_vgg: np.ndarray = vgg19_model.predict(image, verbose=1)
image_after_vgg = np.squeeze(image_after_vgg, axis=0)
return image_after_vgg

```

```

def store_data(self) -> None:
    number_of_splits = (
        sys.getsizeof(self.x_train_after_vgg) // (4 * 1000000000) + 1
    )
    print("bytes: {}".format(sys.getsizeof(self.x_train_after_vgg)))
    print("number of splits: {}".format(number_of_splits))
    splits = np.array_split(
        self.x_train_after_vgg, number_of_splits, axis=0
    )
    for idx, split in enumerate(splits):
        with open(
            self.target_dir + f"x_train_after_vgg_{idx}.pickle",
            "wb"
        ) as file:
            pickle.dump(split, file)

    with open(self.target_dir + "y_train.pickle", "wb") as file:
        pickle.dump(self.y_train, file)
    with open(self.target_dir + "filenames_train.pickle", "wb") as file:
        pickle.dump(self.filenames, file)

    print("Sucessully stored pickle files")

```

```

if __name__ == "__main__":
    NORMALIZATION = "non_normalized"
    SOURCE_DIR = "PATH/TO/TILES"
    TARGET_DIR = "pickles/5x/dataset_1/{}".format(NORMALIZATION)

    data_creator = DataCreator(NORMALIZATION, SOURCE_DIR, TARGET_DIR)

    data_creator.create_data_lists()
    data_creator.preprocess_data()
    data_creator.store_data()

```

7.3.4. Skript 4: Trainieren von CNN_{Tumor}

```

import pickle
import sys
from sys import argv
from time import time

import matplotlib.pyplot as plt # type: ignore
import numpy as np
from keras.layers import Dense, Dropout, Flatten # type: ignore
from keras.models import Sequential, load_model # type: ignore
from keras.optimizers import Adam # type: ignore
from keras.utils.vis_utils import plot_model # type: ignore

```

```
plt.switch_backend("agg")
```

```

class TumorDetector:
    def __init__(self, model_name: str, normalization: str, batch_size: int,
                 num_of_epochs: int, lr: float, loss: str, val_split: float,
                 source_dir: str):
        self.MODEL_NAME = model_name
        self.NORMALIZATION = normalization
        self.BATCH_SIZE = batch_size

```

```

self.NUM_OF_EPOCHS = num_of_epochs
self.LR = lr
self.LOSS = loss
self.VAL_SPLIT = val_split
self.source_dir = source_dir
self.x_train_after_vgg, self.y_train = self.load_training_data()
self.model = None
self.training_time = None
self.training_time_epoch = None
self.history: dict

def load_training_data(self):
    with open(self.source_dir +
              "x_train_after_vgg_0.pickle", "rb") as file:
        x_train_after_vgg = pickle.load(file).astype("float32")
    with open(self.source_dir +
              "y_train.pickle", "rb") as file:
        y_train = pickle.load(file).astype("float32")

    print(y_train.shape)
    y_train = np.squeeze(y_train, axis=1)
    print(y_train.shape)
    print("Successfully loaded pickle files")

    return x_train_after_vgg, y_train

def create_model(self):
    self.model = Sequential()

    self.model.add(Flatten(input_shape=(7, 7, 512)))
    self.model.add(Dense(1024, activation="relu"))
    self.model.add(Dropout(0.2))
    self.model.add(Dense(512, activation="relu"))
    self.model.add(Dropout(0.2))
    self.model.add(Dense(128, activation="relu"))
    self.model.add(Dropout(0.2))
    self.model.add(Dense(16, activation="relu"))
    self.model.add(Dropout(0.2))
    self.model.add(Dense(2, activation="softmax"))

    self.model.compile(optimizer=Adam(lr=self.LR),
                       loss=self.LOSS, metrics=["acc"])

def load_model(self):
    self.model = load_model("models/" + self.MODEL_NAME)

def train_model(self):
    print("Starting to train model.")
    plot_name = self.MODEL_NAME + "_plotted.png"
    plot_model(self.model, to_file=plot_name,
               show_shapes=True, show_layer_names=True)
    start = time()
    history = self.model.fit(self.x_train_after_vgg, self.y_train,
                             epochs=self.NUM_OF_EPOCHS,
                             batch_size=self.BATCH_SIZE,
                             validation_split=self.VAL_SPLIT,
                             shuffle=True)
    self.training_time = time() - start
    self.training_time_epoch = self.training_time / self.NUM_OF_EPOCHS

    self.history = history.history

    self.model.save("models/" + self.MODEL_NAME)

def plot_model_performance(self):
    print("Starting to plot model performance")
    x = [i for i in range(1, self.NUM_OF_EPOCHS+1)]
    fig, axs = plt.subplots(2, 2, figsize=(20, 20))

```

```

ax_loss = axs[0][0]
ax_acc = axs[1][0]
ax_sum = axs[0][1]
ax_text = axs[1][1]
fig.suptitle("Performance of " + self.MODEL_NAME)

ax_loss.plot(x, self.history["loss"], label="loss")
ax_loss.plot(x, self.history["val_loss"], label="val_loss")
ax_loss.set_ylabel("loss")
ax_loss.legend(loc="upper left")
ax_loss.xaxis.get_major_locator().set_params(integer=True)
ax_loss.title.set_text("Loss")

ax_acc.plot(x, self.history["acc"], label="acc")
ax_acc.plot(x, self.history["val_acc"], label="val_acc")
ax_acc.set_ylabel("acc")
ax_acc.set_xlabel("epochs")
ax_acc.legend(loc="upper left")
ax_acc.xaxis.get_major_locator().set_params(integer=True)
ax_acc.title.set_text("Accuracy")

ax_sum.axis("off")
image = plt.imread(self.MODEL_NAME + "_plotted.png")
ax_sum.imshow(image)
ax_sum.title.set_text("Model Architecture")

ax_text.axis("off")
ax_text.text(0.1, 0.9, "Total Training Time: %0.2f sec" %
             self.training_time, fontsize=15)
ax_text.text(0.1, 0.8, "Training Time / Epoch: %0.2f sec" %
             self.training_time_epoch, fontsize=15)
ax_text.text(0.1, 0.7, "Number of Epochs: %i" %
             self.NUM_OF_EPOCHS, fontsize=15)
ax_text.text(0.1, 0.6, "Batch Size: %i" % self.BATCH_SIZE, fontsize=15)
ax_text.text(0.1, 0.5, "Learning Rate: %0.6f" % self.LR, fontsize=15)
ax_text.text(0.1, 0.4, "Loss: %s" % self.LOSS, fontsize=15)
ax_text.text(0.1, 0.3, "Validation Split: %0.2f" %
             self.VAL_SPLIT, fontsize=15)
ax_text.text(0.1, 0.2, "Size of Training Set: %i" %
             self.y_train.shape[0], fontsize=15)
fig.savefig(self.MODEL_NAME + "_performance2.png")

if __name__ == "__main__":
    if len(argv) < 5:
        print("""
        You gave too less arguments.
        1. Create new model or load existing model (create/load)
        2. Name of the model to create/load
        3. Batch_size (32/64/128)
        4. Epochs
        5. Learning rate
        """)
        sys.exit()

    MODEL_NAME = argv[1]
    NORMALIZATION = "non_normalized"
    BATCH_SIZE = int(argv[2])
    NUM_OF_EPOCHS = int(argv[3])
    LR = float(argv[4])
    LOSS = "categorical_crossentropy"
    VAL_SPLIT = 0.2
    SOURCE_DIR = "pickles/5x/dataset_1/{}/".format(NORMALIZATION)

    tumor_detector = TumorDetector(
        MODEL_NAME, NORMALIZATION, BATCH_SIZE,
        NUM_OF_EPOCHS, LR, LOSS, VAL_SPLIT, SOURCE_DIR

```

```
)

if argv[0] == "create":
    tumor_detector.create_model()
elif argv[0] == "load":
```

```
tumor_detector.load_model()

tumor_detector.train_model()
tumor_detector.plot_model_performance()
```

7.3.5. Skript 5: Evaluation von CNN_{tumor}

```
import os
import pickle
import sys
from statistics import mean, median, stdev
from sys import argv

import matplotlib.pyplot as plt # type:ignore
import numpy as np # type:ignore
import pandas as pd # type:ignore
from keras.models import load_model # type:ignore
from scipy import stats # type:ignore
from sklearn.metrics import auc, confusion_matrix, roc_curve # type:ignore

plt.switch_backend("agg")

class Evaluator:
    def __init__(self, pickle_src: str, model_name: str, normalization: str):
        self.model_name = model_name
        self.model = load_model("models/" + self.model_name)
        self.pickle_src = pickle_src
        self.normalization = normalization
        self.x_test_after_vgg = np.ndarray
        self.y_test = np.ndarray # 2 dimensional
        self.filenamees: list
        self.y_true: list # 1 dimensional
        self.pred: np.ndarray
        self.pred_proba: list
        self.pred_dich: list # 1 dimensional
        self.pred_tumor: list # 1 dimensional
        self.optimal_thresh: int

        self.false_tumor_bp: list
        self.false_non_tumor_bp: list
        self.right_tumor_bp: list
        self.right_non_tumor_bp: list

        self.tumor_all = 0
        self.non_tumor_all = 0
        self.false_tumor = 0
        self.right_tumor = 0
        self.false_non_tumor = 0
        self.right_non_tumor = 0

        self.fig, self.axs = plt.subplots(3, 2, figsize=(20, 20))
        self.ax_conf1 = self.axs[0][0]
        self.ax_conf2 = self.axs[0][1]
        self.ax_bp1 = self.axs[1][0]
        self.ax_bp2 = self.axs[1][1]
        self.ax_roc = self.axs[2][0]
        self.ax_text = self.axs[2][1]

    def open_pickles(self):
        with open(self.pickle_src + "x_test_after_vgg_0.pickle", "rb") as file:
            self.x_test_after_vgg = pickle.load(file).astype("float32")

        for i in range(1, 1000):
            current_file = (
```

```
                self.pickle_src + "x_test_after_vgg_{}.pickle".format(i)
            )
        if os.path.isfile(current_file):
            with open(current_file, "rb") as file:
                x_test_after_vgg_chnk = pickle.load(file).astype("float32")
            self.x_test_after_vgg = np.concatenate(
                (self.x_test_after_vgg, x_test_after_vgg_chnk)
            )
        else:
            break

        with open(self.pickle_src + "y_test.pickle", "rb") as file:
            self.y_test = pickle.load(file).astype("float32")
        self.y_test = np.squeeze(self.y_test, axis=1)
        with open(self.pickle_src + "filenamees_test.pickle", "rb") as file:
            self.filenamees = pickle.load(file)

        print("Successfully loaded pickle files")

    def test_model(self):
        print("Starting to test model")
        pred = self.model.predict(self.x_test_after_vgg, verbose=1)

        self.y_true = pd.Series(
            np.argmax(self.y_test, axis=1), name="actual"
        ).tolist()

        self.pred_proba = pred.tolist()
        self.pred_tumor = [
            self.pred_proba[i][1] for i in range(len(self.pred_proba))
        ]

        # Find optimal threshold.
        self.build_roc_curve()
        print(self.optimal_thresh)

        # Final classification.
        self.pred_dich = [
            1 if self.pred_tumor[i] > self.optimal_thresh else
            0 for i in range(len(self.pred_tumor))
        ]

        for i in range(len(self.y_true)):
            if self.y_true[i] == 1:
                self.tumor_all += 1
                if self.pred_dich[i] == 1:
                    self.right_tumor += 1
                    self.right_tumor_bp.append(pred[i][1])
                else:
                    self.false_non_tumor += 1
                    self.false_non_tumor_bp.append(pred[i][0])
            elif self.y_true[i] == 0:
                self.non_tumor_all += 1
                if self.pred_dich[i] == 0:
                    self.right_non_tumor += 1
                    self.right_non_tumor_bp.append(pred[i][0])
                else:
                    self.false_tumor += 1
                    self.false_tumor_bp.append(pred[i][1])
```

```

def build_confusion_matrix(self) -> None:
    conf_matrix = confusion_matrix(
        y_true=self.y_true, y_pred=self.pred_dich
    )
    self.ax_conf1.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            self.ax_conf1.text(x=j, y=i, s=conf_matrix[i, j],
                               va="center", ha="center", size="xx-large")
    conf_matrix_normalized = np.around(
        conf_matrix.astype("float") /
        (conf_matrix.sum(axis=1)[, np.newaxis]),
        decimals=2
    )
    self.ax_conf2.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(conf_matrix_normalized.shape[0]):
        for j in range(conf_matrix_normalized.shape[1]):
            self.ax_conf2.text(x=j, y=i, s=conf_matrix_normalized[i, j],
                               va="center", ha="center", size="xx-large")
    self.ax_conf1.set_xlabel("Predictions")
    self.ax_conf1.set_ylabel("Actuals")
    self.ax_conf2.set_xlabel("Predictions")
    self.ax_conf2.set_ylabel("Actuals")
    self.ax_conf1.title.set_text("Confusion Matrix")
    self.ax_conf2.title.set_text("Confusion Matrix normalized")

def build_boxplot(self) -> None:
    data_bp_tumor = [self.right_tumor_bp, self.false_tumor_bp]
    data_bp_non_tumor = [self.right_non_tumor_bp, self.false_non_tumor_bp]
    self.ax_bp1.boxplot(data_bp_tumor)
    self.ax_bp2.boxplot(data_bp_non_tumor)
    self.ax_bp1.set_xticklabels(["right_tumor", "false_tumor"])
    self.ax_bp2.set_xticklabels(["right_non_tumor", "false_non_tumor"])
    self.ax_bp1.set_ylabel("prediction probability tumor")
    self.ax_bp2.set_ylabel("prediction probability non_tumor")
    self.ax_bp1.set_ylim((0.25, 1.05))
    self.ax_bp2.set_ylim((0.25, 1.05))
    self.ax_bp1.title.set_text("Probabilities of tumor predictions")
    self.ax_bp2.title.set_text("Probabilities of non_tumor predictions")

def test_normal_distribution(self) -> None:
    p1_rt = self.right_tumor_bp
    p1_ft = self.false_tumor_bp
    p1_rnt = [1-x for x in self.right_non_tumor_bp]
    p1_fnt = [1-x for x in self.false_non_tumor_bp]
    p1_all = [p1_rt, p1_ft, p1_rnt, p1_fnt]
    for ls in p1_all:
        _, pvalue = stats.normaltest(ls)
        if pvalue <= 0.05:
            med = median(ls)
            maxi = max(ls)
            mini = min(ls)
            print(med, mini, maxi)
        else:
            print(mean(ls), stdev(ls))

def build_roc_curve(self) -> None:
    fpr, tpr, thresholds = roc_curve(self.y_true, self.pred_tumor)
    au_curve = auc(fpr, tpr)
    idx = np.argmax(tpr - fpr)
    self.optimal_thresh = thresholds[idx]
    self.ax_roc.plot(fpr, tpr, 'b', label="AUC = %0.2f" % au_curve)
    self.ax_roc.legend(loc="lower right")
    self.ax_roc.plot([0, 1], [0, 1], "r--")
    self.ax_roc.scatter(
        fpr[idx], tpr[idx], marker='o', color="black", label="Best"
    )
    self.ax_roc.set_xlim([0, 1])
    self.ax_roc.set_ylim([0, 1])
    self.ax_roc.set_xlabel("False Positive Rate")
    self.ax_roc.set_ylabel("True Positive Rate")
    self.ax_roc.title.set_text("ROC curve")

def plot_specifications(self) -> None:
    sens = self.right_tumor / self.tumor_all
    spec = self.right_non_tumor / self.non_tumor_all
    lr = self.model.optimizer.get_config()["lr"]
    optimizer = self.model.optimizer.__class__.__name__
    loss_func = self.model.loss

    score = self.model.evaluate(self.x_test_after_vgg, self.y_test, verbose=1)
    loss = score[0]
    acc = score[1]

    self.ax_text.axis("off")
    self.ax_text.text(0.1, 0.9, "Sensitivity: %0.2f" % sens, fontsize=15)
    self.ax_text.text(0.1, 0.85, "Specificity: %0.2f" % spec, fontsize=15)
    self.ax_text.text(
        0.1, 0.75, "Optimizer: %s with lr of %0.8f" % (optimizer, lr), fontsize=15
    )
    self.ax_text.text(0.1, 0.7, "Loss Function: %s" % loss_func, fontsize=15)
    self.ax_text.text(0.1, 0.65, "Loss: %s" % loss, fontsize=15)
    self.ax_text.text(0.1, 0.6, "Accuracy: %s" % acc, fontsize=15)
    self.ax_text.text(0.1, 0.5, "%s" % self.normalization, fontsize=15)
    self.ax_text.text(
        0.1, 0.4, "Optimal Threshold: %s" % self.optimal_thresh, fontsize=15
    )
    print(self.model.summary())

def save(self):
    self.fig.suptitle("Evaluation of " + self.model_name[:3])
    self.fig.savefig(MODEL_NAME + "_evaluation.png")

if __name__ == "__main__":
    if len(argv) < 1:
        print("""
        You gave too less arguments.
        1. Name of the model to test
        """)
        sys.exit()

    MODEL_NAME = argv[0]
    DATASET = "1"
    NORMALIZATION = "non_normalized"
    pickle_src = "pickles/5x/dataset_{}/{}".format(DATASET, NORMALIZATION)

    evaluator = Evaluator = Evaluator(
        pickle_src, MODEL_NAME, NORMALIZATION
    )

    evaluator.open_pickles()
    evaluator.test_model()
    evaluator.build_confusion_matrix()
    evaluator.build_boxplot()
    evaluator.test_normal_distribution()
    evaluator.plot_specifications()
    evaluator.save()

```

7.3.6. Skript 6: Zerteilen der nicht-annotierten Slides in Tiles

```
import os # type: ignore

import large_image # type: ignore
from PIL import Image # type: ignore
from tqdm import tqdm # type: ignore

# TODO: Specify path.
SLIDE_PATH = "PATH/TO/SLIDES/"
OUTPUT = SLIDE_PATH + "tiles/"

slides = [slide for slide in os.listdir(SLIDE_PATH) if slide[-5:] == ".ndpi"]

for slide in tqdm(slides):
    ts = large_image.getTileSource(SLIDE_PATH + slide)
    print(slide)

    for tile_info in ts.tileiterator(
        scale=dict(magnification=5),
        tile_size=dict(width=224, height=224),
        tile_overlap=dict(x=0, y=0),
        format=large_image.tilesource.TILE_FORMAT_PIL
    ):
        rgba_tile = tile_info["tile"]
        x = tile_info["tile_position"]["level_x"]
        y = tile_info["tile_position"]["level_y"]

        rgba_tile.load() # required for png.split()

    im_tile = Image.new("RGB", rgba_tile.size, (255, 255, 255))
    im_tile.paste(rgba_tile, mask=rgba_tile.split()[3])

    im_tile.save(
        OUTPUT + "{}_{}.jpeg".format(slide[:-5], x, y),
        "JPEG", quality=100
    )
```

7.3.7. Skript 7: Herausfiltern des Hintergrunds

```
import os

import numpy as np # type: ignore
from PIL import Image, ImageOps # type: ignore
from tqdm import tqdm # type: ignore

def is_tissue(tile_path) -> bool:
    img = Image.open(tile_path).convert('L') # opens and converts the img
    # to grayscale

    img = ImageOps.invert(img) # usually the foreground is defined as white
    img = np.asarray(img)

    thresh: int = 50

    foreground = (img > thresh)

    total_pixels: int = img.size
    foreground_pixels: int = 0

    for row in foreground:
        for pixel in row:
            if pixel:
                foreground_pixels += 1

    foreground_percentage: float = (foreground_pixels / total_pixels) * 100

    if foreground_percentage > 40:
        return True
    return False

if __name__ == "__main__":
    # TODO: Specify path.
    # source: str = "/PATH/TO/TILES/"

    tiles = [f for f in os.listdir(source) if f[-5:] == ".jpeg"]
    print(len(tiles))

    for tile in tqdm(tiles):
        current_path: str = source + tile
        if not is_tissue(current_path):
            new_path: str = source + "background/" + tile
            os.rename(current_path, new_path)
```

7.3.8. Skript 8: Markieren der Tumor-Tiles

```
import os
import pickle

import numpy as np # type: ignore
import pandas as pd # type: ignore
from openslide import OpenSlide # type: ignore
from PIL import Image, ImageDraw # type: ignore
from tensorflow.keras.models import load_model # type: ignore

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

pickle_src = "../pickles/tumor_detection/"

SLIDE_PATH = (
    "PATH/TO/SLIDES/"
)

MODEL_NAME = "model_01_12_2021_5_non_normalized.h5"
DOWNSAMPLE = 0.05

with open(pickle_src + "x_test_after_vgg_0.pickle", "rb") as file:
    x_test_after_vgg = pickle.load(file).astype("float32")
```



```

with open(pickle_src + "y_test.pickle", "rb") as file:
    y_test = pickle.load(file).astype("float32")
y_test = np.squeeze(y_test, axis=1)
with open(pickle_src + "filenames_test.pickle", "rb") as file:
    filenames = pickle.load(file)

print("Successfully loaded pickle files")
print(x_test_after_vgg.shape)

if os.path.exists(pickle_src + "y_pred.pickle"):
    with open(pickle_src + "y_pred.pickle", "rb") as file:
        y_pred = pickle.load(file)
else:
    print("Starting to test model")
    model = load_model(MODEL_NAME)
    pred = model.predict(x_test_after_vgg, verbose=1)
    y_pred = pd.Series(np.argmax(pred, axis=1), name="pred").tolist()

    with open(pickle_src + "y_pred.pickle", "wb") as file:
        pickle.dump(y_pred, file)

slides = [slide for slide in os.listdir(SLIDE_PATH)
          if slide[-5:] == ".ndpi"]

for slide in slides:
    print(slide)

    # Make a thumbnail that we can draw on.
    wsi = OpenSlide(SLIDE_PATH + slide)
    width = wsi.dimensions[0]
    height = wsi.dimensions[1]

```

```

small_slide = wsi.get_thumbnail((width*DOWNSAMPLE, height*DOWNSAMPLE))
small_slide.save("thumbnails/{}.jpeg".format(slide[:-5]))
small_slide = Image.open("thumbnails/{}.jpeg".format(slide[:-5]))
marked = ImageDraw.Draw(small_slide)

# Get all tiles for current slide that indicate no_tumor
tiles_nt = [filenames[i] for i in range(len(filenames))
            if filenames[i].split(" ")[0] == slide[:-5]
            and y_pred[i] == 0]

# Get all tiles for current slide that indicate tumor
tiles = [filenames[i] for i in range(len(filenames))
         if filenames[i].split(" ")[0] == slide[:-5]
         and y_pred[i] == 1]

for tile in tiles:
    x_start_idx = tile.index("x=") + 2
    x_end_idx = tile.index("y=") - 1
    y_start_idx = tile.index("y=") + 2
    y_end_idx = tile.index("w=") - 1
    x_pos = int(tile[x_start_idx: x_end_idx])
    y_pos = int(tile[y_start_idx: y_end_idx])

    x1 = x_pos * DOWNSAMPLE
    x2 = (x_pos + (8 * 224)) * DOWNSAMPLE
    y1 = y_pos * DOWNSAMPLE
    y2 = (y_pos + (8 * 224)) * DOWNSAMPLE

    marked.rectangle((x1, y1, x2, y2), outline="yellow", width=5)
    small_slide.save("thumbnails/marked_{}.jpg".format(slide[:-5]))

```

7.3.9. Skript 9: Erstellen der Trainingsdaten für CNN_{Progress}

```

import concurrent.futures
import os
import pickle
import random
import sys
import time
import warnings
from itertools import repeat
from sys import argv

import numpy as np
from PIL import Image # type: ignore

os.environ["TF_CPP_MIN_LOG_LEVEL"] = '2'

from keras.applications.vgg19 import VGG19, preprocess_input
from keras.models import load_model
from keras.utils import to_categorical

class DataCreator:
    def __init__(self, tumor_detector: str, normalization: str,
                 source_dir: str, target_dir: str):
        self.MAG = 5
        self.TUMOR_DETECTOR = tumor_detector
        self.NORMALIZATION = normalization
        self.source_dir = source_dir
        self.target_dir = target_dir
        self.label: dict = {"progress": 1, "non_progress": 0}
        self.x_train: np.ndarray
        self.x_train_after_vgg: np.ndarray

```

```

self.y_train: np.ndarray
self.filenames: list = []

def create_data_lists(self):
    categories = {"progress": [], "non_progress": []}

    # Make 2 lists (progress/non_progress) which contain all
    # the images as np.array with their corresponding label and filename
    for key in categories.keys():
        categories[key] = self.open_images(key)

    # Combine them into one list, still attached to their
    # label and filename, in order to shuffle the training data
    train = categories["progress"] + categories["non_progress"]

    x_train_ls = []
    y_train_ls = []
    # Split image, label and filename into x, y and z
    for x, y, z in train:
        with warnings.catch_warnings():
            warnings.simplefilter(action='ignore', category=FutureWarning)
            if x == "fail": # throws a warning
                continue
            x_train_ls.append(x)
            y_train_ls.append(y)
            self.filenames.append(z)

    # Transforming into np.array
    self.x_train = np.asarray(x_train_ls)
    self.y_train = np.asarray(y_train_ls).astype("float32")

```

```

def open_images(self, key):
    source = self.source_dir + key + "/"

    file_names = [
        file for file in os.listdir(source) if file[-5:] == ".jpeg"
    ]
    print("Starting {} files".format(len(file_names), key))

    MAX_WORKERS = int(os.getenv("SLURM_CPUS_PER_TASK"))
    MAX_SIZE_OF_CHUNKS = 48
    NUMBER_OF_CHUNKS = (len(file_names) // MAX_SIZE_OF_CHUNKS) + 1
    file_names = np.asarray(file_names)
    chunks = np.array_split(file_names, NUMBER_OF_CHUNKS)

    start = time.time()
    images = []

    for i in range(NUMBER_OF_CHUNKS):
        with concurrent.futures.ProcessPoolExecutor(
            max_workers=MAX_WORKERS
        ) as executor:
            images += list(
                executor.map(self.process_image, repeat(key),
                    repeat(source), chunks[i].tolist())
            )

    end = time.time()
    print("Processing all images took {} seconds".format(end-start))
    return images

def process_image(self, key: str, source: str, img_filename: str):
    try:
        # WITHOUT Normalization.
        img = Image.open(source + img_filename)
        img = np.asarray(img)
        img = img.reshape(224, 224, 3).astype("float32")

        return (img, to_categorical(self.label[key], 2), img_filename)
    except:
        print("could not process file {}".format(img_filename))
        return ("fail", "fail", "fail")

def preprocess_data(self):
    SIZE_OF_CHUNKS = 10
    print("x_train shape: {}".format(self.x_train.shape))
    NUMBER_OF_CHUNKS = self.x_train.shape[0] // SIZE_OF_CHUNKS
    MAX_WORKERS = 5
    start = time.time()
    chunks = np.array_split(self.x_train, NUMBER_OF_CHUNKS, axis=0)
    print("chunks have shape of {}".format(chunks[0].shape))
    print("NUMBER OF CHUNKS: {}, SIZE_OF_CHUNKS: {}".format(
        NUMBER_OF_CHUNKS, SIZE_OF_CHUNKS))

    with concurrent.futures.ProcessPoolExecutor(
        max_workers=MAX_WORKERS
    ) as executor:
        self.x_train_after_vgg = np.asarray(
            list(executor.map(
                self.vgg_preprocessing, chunks[0].tolist()
            )
        ).astype("float32")

    for i in range(1, NUMBER_OF_CHUNKS):
        with concurrent.futures.ProcessPoolExecutor(
            max_workers=MAX_WORKERS
        ) as executor:
            x_train_after_vgg_chunk = np.asarray(
                list(executor.map(
                    self.vgg_preprocessing, chunks[i].tolist()
                )
            ).astype("float32")

        self.x_train_after_vgg = np.concatenate(
            (self.x_train_after_vgg, x_train_after_vgg_chunk)
        )
        print(self.x_train_after_vgg.shape)

    print("Processing data in Convolutional VGG19 done.")
    end = time.time()
    print("Processing in VGG19 took {} seconds.".format(end-start))

def vgg_preprocessing(self, image):
    image = np.asarray(image)
    image = image.reshape(
        1, image.shape[0], image.shape[1], image.shape[2]
    )
    image = preprocess_input(image)
    vgg19_model = VGG19(include_top=False, input_shape=(224, 224, 3))
    vgg19_model.trainable = False
    image_after_vgg = vgg19_model.predict(image, verbose=1)
    image_after_vgg = np.squeeze(image_after_vgg, axis=0)

    return image_after_vgg

def tumor_detector(self):
    train_tumor = []
    tumor_detector = load_model(
        "../tumor_detector/models/" + self.TUMOR_DETECTOR
    )
    pred = tumor_detector.predict(self.x_train_after_vgg, verbose=1)

    # Skip non_tumor tiles.
    for idx, self.filename in enumerate(self.filenamees):
        if np.argmax(pred[idx]) == 1:
            train_tumor.append(
                [self.x_train_after_vgg[idx],
                 self.y_train[idx], self.filename]
            )

    print("Training data successfully build.")

    random.shuffle(train_tumor)

    # Split image, label and filename into x, y and z

    x_train_after_vgg_ls = []
    y_train_ls = []
    self.filenamees = []
    for x, y, z in train_tumor:
        x_train_after_vgg_ls.append(x)
        y_train_ls.append(y)
        self.filenamees.append(z)

    print("Training data successfully splitted.")

    # Transforming into np.array
    self.x_train_after_vgg = np.asarray(x_train_after_vgg_ls)
    self.y_train = np.asarray(y_train_ls).astype("float32")

def store_data(self):
    # Store them in pickles to open them elsewhere
    number_of_splits = (
        sys.getsizeof(self.x_train_after_vgg) //

```

```

        (4 * 100000000) + 1
    )
    print("bytes: {}".format(sys.getsizeof(self.x_train_after_vgg)))
    print("number of splits: {}".format(number_of_splits))
    splits = np.array_split(
        self.x_train_after_vgg, number_of_splits, axis=0
    )
    for idx, split in enumerate(splits):
        with open(
            self.target_dir +
            "x_train_after_vgg_{}.pickle".format(idx),
            "wb"
        ) as file:
            pickle.dump(split, file)

    with open(
        self.target_dir +
        "y_train.pickle",
        "wb"
    ) as file:
        pickle.dump(self.y_train, file)

    with open(
        self.target_dir +
        "filenames_train.pickle",
        "wb"
    ) as file:
        pickle.dump(self.filenames, file)

```

```

    print("Sucessully stored pickle files")

if __name__ == "__main__":

    if len(argv) <= 2:
        print("""
            1. Name of the Tumor Detector
            """)
        sys.exit()

    TUMOR_DETECTOR = argv[1]
    NORMALIZATION = "non_normalized"

    # TODO: Specify path.
    SOURCE_DIR = "PATH/TO/TILES/"
    TARGET_DIR = "pickles/5x/{}/".format(NORMALIZATION)

    data_creator: DataCreator = DataCreator(
        TUMOR_DETECTOR, NORMALIZATION, SOURCE_DIR, TARGET_DIR
    )

    data_creator.create_data_lists()
    data_creator.preprocess_data()
    data_creator.tumor_detector()
    data_creator.store_data()

```

7.3.10. Skript 10: Trainieren von CNN Progress

```

import pickle
import sys
from sys import argv
from time import time

import matplotlib.pyplot as plt # type: ignore
import numpy as np
from keras.layers import Dense, Dropout, Flatten # type: ignore
from keras.models import Sequential, load_model # type: ignore
from keras.optimizers import Adam # type: ignore
from keras.utils.vis_utils import plot_model # type: ignore

plt.switch_backend("agg")

class ProgressDetector:
    def __init__(self, model_name: str, normalization: str, batch_size: int,
                 num_of_epochs: int, lr: float, loss: str, val_split: float,
                 source_dir: str):
        self.MODEL_NAME = model_name
        self.NORMALIZATION = normalization
        self.BATCH_SIZE = batch_size
        self.NUM_OF_EPOCHS = num_of_epochs
        self.LR = lr
        self.LOSS = loss
        self.VAL_SPLIT = val_split
        self.source_dir = source_dir
        self.x_train_after_vgg, self.y_train = self.load_training_data()
        self.model = None
        self.training_time = None
        self.training_time_epoch = None
        self.history: dict

    def load_training_data(self):

```

```

        with open(self.source_dir +
                  "x_train_after_vgg_0.pickle", "rb") as file:
            x_train_after_vgg = pickle.load(file).astype("float32")
        with open(self.source_dir +
                  "y_train.pickle", "rb") as file:
            y_train = pickle.load(file).astype("float32")

        print(y_train.shape)
        y_train = np.squeeze(y_train, axis=1)
        print(y_train.shape)
        print("Successfully loaded pickle files")

        return x_train_after_vgg, y_train

    def create_model(self):
        self.model = Sequential()

        self.model.add(Flatten(input_shape=(7, 7, 512)))
        self.model.add(Dense(1024, activation="relu"))
        self.model.add(Dropout(0.2))
        self.model.add(Dense(512, activation="relu"))
        self.model.add(Dropout(0.2))
        self.model.add(Dense(128, activation="relu"))
        self.model.add(Dropout(0.2))
        self.model.add(Dense(16, activation="relu"))
        self.model.add(Dropout(0.2))
        self.model.add(Dense(2, activation="softmax"))

        self.model.compile(optimizer=Adam(lr=self.LR),
                           loss=self.LOSS, metrics=["acc"])

    def load_model(self):
        self.model = load_model("models/" + self.MODEL_NAME)

```

```

def train_model(self):
    print("Starting to train model.")
    plot_name = self.MODEL_NAME + "_plotted.png"
    plot_model(self.model, to_file=plot_name,
               show_shapes=True, show_layer_names=True)
    start = time()
    history = self.model.fit(self.x_train_after_vgg, self.y_train,
                            epochs=self.NUM_OF_EPOCHS,
                            batch_size=self.BATCH_SIZE,
                            validation_split=self.VAL_SPLIT,
                            shuffle=True)
    self.training_time = time() - start
    self.training_time_epoch = self.training_time / self.NUM_OF_EPOCHS

    self.history = history.history

    self.model.save("models/" + self.MODEL_NAME)

def plot_model_performance(self):
    print("Starting to plot model performance")
    x = [i for i in range(1, self.NUM_OF_EPOCHS+1)]
    fig, axes = plt.subplots(2, 2, figsize=(20, 20))
    ax_loss = axes[0][0]
    ax_acc = axes[1][0]
    ax_sum = axes[0][1]
    ax_text = axes[1][1]
    fig.suptitle("Performance of " + self.MODEL_NAME)

    ax_loss.plot(x, self.history["loss"], label="loss")
    ax_loss.plot(x, self.history["val_loss"], label="val_loss")
    ax_loss.set_ylabel("loss")
    ax_loss.legend(loc="upper left")
    ax_loss.xaxis.get_major_locator().set_params(integer=True)
    ax_loss.title.set_text("Loss")

    ax_acc.plot(x, self.history["acc"], label="acc")
    ax_acc.plot(x, self.history["val_acc"], label="val_acc")
    ax_acc.set_ylabel("acc")
    ax_acc.set_xlabel("epochs")
    ax_acc.legend(loc="upper left")
    ax_acc.xaxis.get_major_locator().set_params(integer=True)
    ax_acc.title.set_text("Accuracy")

    ax_sum.axis("off")
    image = plt.imread(self.MODEL_NAME + "_plotted.png")
    ax_sum.imshow(image)
    ax_sum.title.set_text("Model Architecture")

    ax_text.axis("off")
    ax_text.text(0.1, 0.9, "Total Training Time: %0.2f sec" %

```

```

                self.training_time, fontsize=15)
    ax_text.text(0.1, 0.8, "Training Time / Epoch: %0.2f sec" %
                self.training_time_epoch, fontsize=15)
    ax_text.text(0.1, 0.7, "Number of Epochs: %i" %
                self.NUM_OF_EPOCHS, fontsize=15)
    ax_text.text(0.1, 0.6, "Batch Size: %i" % self.BATCH_SIZE, fontsize=15)
    ax_text.text(0.1, 0.5, "Learning Rate: %0.6f" % self.LR, fontsize=15)
    ax_text.text(0.1, 0.4, "Loss: %s" % self.LOSS, fontsize=15)
    ax_text.text(0.1, 0.3, "Validation Split: %0.2f" %
                self.VAL_SPLIT, fontsize=15)
    ax_text.text(0.1, 0.2, "Size of Training Set: %i" %
                self.y_train.shape[0], fontsize=15)
    fig.savefig(self.MODEL_NAME + "_performance2.png")

```

```

if __name__ == "__main__":

```

```

    if len(argv) < 5:

```

```

        print("""

```

```

            You gave too less arguments.

```

```

            1. Create new model or load existing model (create/load)

```

```

            2. Name of the model to create/load

```

```

            3. Batch_size (32/64/128)

```

```

            4. Epochs

```

```

            5. Learning rate

```

```

            """)

```

```

        sys.exit()

```

```

MODEL_NAME = argv[1]

```

```

NORMALIZATION = "non_normalized"

```

```

BATCH_SIZE = int(argv[2])

```

```

NUM_OF_EPOCHS = int(argv[3])

```

```

LR = float(argv[4])

```

```

LOSS = "categorical_crossentropy"

```

```

VAL_SPLIT = 0.2

```

```

SOURCE_DIR = "pickles/5x/dataset_1/{}/".format(NORMALIZATION)

```

```

progress_detector = ProgressDetector(

```

```

    MODEL_NAME, NORMALIZATION, BATCH_SIZE,

```

```

    NUM_OF_EPOCHS, LR, LOSS, VAL_SPLIT, SOURCE_DIR

```

```

)

```

```

if argv[0] == "create":

```

```

    progress_detector.create_model()

```

```

elif argv[0] == "load":

```

```

    progress_detector.load_model()

```

```

progress_detector.train_model()

```

```

progress_detector.plot_model_performance()

```

7.3.11. Skript 11: Evaluation von CNN Progress

```

import pickle

```

```

import sys

```

```

from sys import argv

```

```

import matplotlib.pyplot as plt # type: ignore

```

```

import numpy as np

```

```

import pandas as pd # type: ignore

```

```

from keras.models import load_model # type: ignore

```

```

from sklearn.metrics import auc, confusion_matrix, roc_curve # type: ignore

```

```

plt.switch_backend("agg")

```

```

class Evaluator:

```

```

    def __init__(self, pickle_src: str, model_name: str, normalization: str):

```

```

        self.model_name = model_name

```

```

        self.model = load_model("models/" + self.model_name)

```

```

        self.pickle_src = pickle_src

```

```

        self.normalization = normalization

```

```

        self.x_test_after_vgg: np.ndarray

```

```

        self.y_test: np.ndarray # 2 dimensional

```

```

        self.file_names: list

```

```

        self.y_true: list # 1 dimensional

```

```

        self.pred: np.ndarray

```

```

self.pred_proba: list # 2 dimensional
self.pred_dich: list # 1 dimensional
self.pred_progress: list # 1 dimensional
self.optimal_thresh: int

self.false_progress_bp: list
self.false_non_progress_bp: list
self.right_progress_bp: list
self.right_non_progress_bp: list

self.progress_all = 0
self.non_progress_all = 0
self.false_progress = 0
self.right_progress = 0
self.false_non_progress = 0
self.right_non_progress = 0

self.fig, self.axs = plt.subplots(4, 2, figsize=(50, 50))
self.ax_conf1 = self.axs[0][0]
self.ax_conf2 = self.axs[0][1]
self.ax_bp1 = self.axs[1][0]
self.ax_bp2 = self.axs[1][1]
self.ax_roc = self.axs[2][0]
self.ax_text = self.axs[2][1]
self.ax_per_slide_p = self.axs[3][0]
self.ax_per_slide_np = self.axs[3][1]

def open_pickles(self):
    with open(pickle_src + "x_test_after_vgg_0.pickle", "rb") as file:
        self.x_test_after_vgg = pickle.load(file).astype("float32")
    with open(pickle_src + "y_test.pickle", "rb") as file:
        self.y_test = pickle.load(file).astype("float32")
    self.y_test = np.squeeze(self.y_test, axis=1)
    with open(pickle_src + "filenames_test.pickle", "rb") as file:
        self.filenames = pickle.load(file)

    print("Successfully loaded pickle files")

def test_model(self):
    print("Starting to test model")
    self.pred = self.model.predict(self.x_test_after_vgg, verbose=1)

    self.y_true = pd.Series(
        np.argmax(self.y_test, axis=1), name="actual"
    ).tolist()

    self.pred_proba = self.pred.tolist()
    self.pred_progress = [
        self.pred_proba[i][1] for i in range(len(self.pred_proba))
    ]

    # Find optimal threshold.
    self.build_roc_curve()
    print(self.optimal_thresh)

    # Final classification.
    self.pred_dich = [
        1 if self.pred_progress[i] > self.optimal_thresh
        else 0 for i in range(len(self.pred_progress))
    ]

    for i in range(len(self.y_true)):
        if self.y_true[i] == 1:
            self.progress_all += 1
            if self.pred_dich[i] == 1:
                self.right_progress += 1
                self.right_progress_bp.append(max(self.pred_proba[i]))
            else:
                self.false_non_progress += 1
                self.false_non_progress_bp.append(max(self.pred_proba[i]))
        elif self.y_true[i] == 0:
            self.non_progress_all += 1
            if self.pred_dich[i] == 0:
                self.right_non_progress += 1
                self.right_non_progress_bp.append(max(self.pred_proba[i]))
            else:
                self.false_progress += 1
                self.false_progress_bp.append(max(self.pred_proba[i]))

def build_roc_curve(self):
    fpr, tpr, thresholds = roc_curve(self.y_true, self.pred_progress)
    aucurve = auc(fpr, tpr)
    idx = np.argmax(tpr - fpr)
    self.optimal_thresh = thresholds[idx]

    self.ax_roc.plot(fpr, tpr, 'b', label="AUC = %0.2f" % aucurve)
    self.ax_roc.legend(loc="lower right")
    self.ax_roc.plot([0, 1], [0, 1], "r--")
    self.ax_roc.scatter(
        fpr[idx], tpr[idx], marker='o', color="black", label="Best"
    )
    self.ax_roc.set_xlim([0, 1])
    self.ax_roc.set_ylim([0, 1])
    self.ax_roc.set_xlabel("False Positive Rate")
    self.ax_roc.set_ylabel("True Positive Rate")
    self.ax_roc.title.set_text("ROC curve")

def build_boxplot(self):
    data_bp_progress = [
        self.right_progress_bp, self.false_progress_bp
    ]
    data_bp_non_progress = [
        self.right_non_progress_bp, self.false_non_progress_bp
    ]
    self.ax_bp1.boxplot(data_bp_progress)
    self.ax_bp2.boxplot(data_bp_non_progress)
    self.ax_bp1.set_xticklabels(
        ["right_progress", "false_progress"]
    )
    self.ax_bp2.set_xticklabels(
        ["right_non_progress", "false_non_progress"]
    )
    self.ax_bp1.set_ylabel("prediction probability progress")
    self.ax_bp2.set_ylabel("prediction probability non_progress")
    self.ax_bp1.set_ylim([0.25, 1.05])
    self.ax_bp2.set_ylim([0.25, 1.05])
    self.ax_bp1.title.set_text("Probabilities of progress predictions")
    self.ax_bp2.title.set_text("Probabilities of non_progress predictions")

def build_confusion_matrix(self):
    conf_matrix = confusion_matrix(
        y_true=self.y_true, y_pred=self.pred_dich
    )

    self.ax_conf1.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            self.ax_conf1.text(
                x=j, y=i, s=conf_matrix[i, j],
                va="center", ha="center", size="xx-large"
            )
    conf_matrix_normalized = np.around(
        conf_matrix.astype("float") /
        (conf_matrix.sum(axis=1)[;], np.newaxis),
        decimals=2
    )

```

```

self.ax_conf2.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix_normalized.shape[0]):
    for j in range(conf_matrix_normalized.shape[1]):
        self.ax_conf2.text(
            x=j, y=i, s=conf_matrix_normalized[i, j],
            va="center", ha="center", size="xx-large"
        )

self.ax_conf1.set_xlabel("Predictions")
self.ax_conf1.set_ylabel("Actuals")
self.ax_conf2.set_xlabel("Predictions")
self.ax_conf2.set_ylabel("Actuals")
self.ax_conf1.title.set_text("Confusion Matrix")
self.ax_conf2.title.set_text("Confusion Matrix normalized")

def plot_specifications(self):
    sens = self.right_progress / self.progress_all
    spec = self.right_non_progress / self.non_progress_all
    lr = self.model.optimizer.get_config()["lr"]
    optimizer = self.model.optimizer.__class__.__name__
    loss_func = self.model.loss

    score = self.model.evaluate(
        self.x_test_after_vgg, self.y_test, verbose=1
    )
    loss = score[0]
    acc = score[1]

    self.ax_text.axis("off")
    self.ax_text.text(0.1, 0.9, "Sensitivity: %0.2f" % sens, fontsize=15)
    self.ax_text.text(0.1, 0.85, "Specificity: %0.2f" % spec, fontsize=15)
    self.ax_text.text(
        0.1, 0.75, "Optimizer: %s with lr of %8f"
        % (optimizer, lr), fontsize=15
    )
    self.ax_text.text(
        0.1, 0.7, "Loss Function: %s"
        % loss_func, fontsize=15
    )
    self.ax_text.text(0.1, 0.65, "Loss: %s" % loss, fontsize=15)
    self.ax_text.text(0.1, 0.6, "Accuracy: %s" % acc, fontsize=15)
    self.ax_text.text(0.1, 0.5, "%s" % NORMALIZATION, fontsize=15)
    self.ax_text.text(
        0.1, 0.4, "Optimal Threshold: %s"
        % self.optimal_thresh, fontsize=15
    )
    print(self.model.summary())

def save(self):
    self.fig.suptitle("Evaluation of " + self.model_name[:-3])
    self.fig.savefig(self.model_name + "_evaluation.png")

def progress_per_slide(self):
    slide_classification = {}
    slide_average_progress = {}
    slide_classification_true = {}
    for idx, filename in enumerate(self filenames):
        histo_id = filename.split("_")[0]
        if histo_id not in slide_classification.keys():
            slide_classification[histo_id] = [0, 0]
        if histo_id not in slide_average_progress.keys():
            slide_average_progress[histo_id] = 0
        if histo_id not in slide_classification_true.keys():
            slide_classification_true[histo_id] = self.y_true[idx]

        if self.pred_dich[idx] == 0:
            slide_classification[histo_id][0] += 1
        elif self.pred_dich[idx] == 1:
            slide_classification[histo_id][1] += 1

        slide_average_progress[histo_id] += self.pred_proba[idx][1]

    histo_ids = list(slide_classification.keys())
    histo_ids.sort()

    histo_ids_p = [
        histo_id for histo_id in histo_ids
        if slide_classification_true[histo_id] == 1
    ]
    histo_ids_np = [
        histo_id for histo_id in histo_ids
        if slide_classification_true[histo_id] == 0
    ]

    for histo_id in histo_ids_p:
        sums = sum(slide_classification[histo_id])
        slide_classification[histo_id][0] = (
            slide_classification[histo_id][0] / sums
        )
        slide_classification[histo_id][1] = (
            slide_classification[histo_id][1] / sums
        )
        slide_average_progress[histo_id] = (
            slide_average_progress[histo_id] / sums
        )

    no_progress = [
        slide_classification[histo_id][0] for histo_id in histo_ids_p
    ]
    progress_p = [
        slide_classification[histo_id][1] for histo_id in histo_ids_p
    ]
    average_progress = [
        slide_average_progress[histo_id] for histo_id in histo_ids_p
    ]
    true_progress = [
        slide_classification_true[histo_id] for histo_id in histo_ids_p
    ]
    x = range(1, len(histo_ids_p) + 1)
    self.ax_per_slide_p.bar(x, progress_p, label="Progress")
    self.ax_per_slide_p.bar(
        x, no_progress, bottom=progress_p, label="No Progress"
    )
    self.ax_per_slide_p.set_xticks(x)
    self.ax_per_slide_p.set_xticklabels(histo_ids_p, rotation=45)
    self.ax_per_slide_p.set_xlabel("Histo IDs")
    self.ax_per_slide_p.set_ylabel("Progress / No Progress")

    for idx, value in enumerate(progress_p):
        self.ax_per_slide_p.text(
            idx+1, 0.05, str(round(value, 2)),
            color="black", ha="center"
        )
    for idx, value in enumerate(average_progress):
        self.ax_per_slide_p.text(
            idx+1, -0.15, str(round(value, 2)),
            color="black", ha="center"
        )
    for idx, value in enumerate(true_progress):
        if value == 0:
            self.ax_per_slide_p.text(
                idx+1, -0.2, str(value),
                color="black", ha="center"
            )
        elif value == 1:
            self.ax_per_slide_p.text(

```

```

        idx+1, -0.2, str(value),
        color="red", ha="center"
    )

for histo_id in histo_ids_np:
    sums = sum(slide_classification[histo_id])
    slide_classification[histo_id][0] = (
        slide_classification[histo_id][0] / sums
    )
    slide_classification[histo_id][1] = (
        slide_classification[histo_id][1] / sums
    )
    slide_average_progress[histo_id] = (
        slide_average_progress[histo_id] / sums
    )

no_progress = [
    slide_classification[histo_id][0] for
    histo_id in histo_ids_np
]
progress_np = [
    slide_classification[histo_id][1] for
    histo_id in histo_ids_np
]
average_progress = [
    slide_average_progress[histo_id] for
    histo_id in histo_ids_np
]
true_progress = [
    slide_classification_true[histo_id] for
    histo_id in histo_ids_np
]

x = range(1, len(histo_ids_np) + 1)
self.ax_per_slide_np.bar(x, progress_np, label="Progress")
self.ax_per_slide_np.bar(
    x, no_progress, bottom=progress_np, label="No Progress"
)
self.ax_per_slide_np.set_xticks(x)
self.ax_per_slide_np.set_xticklabels(histo_ids_np, rotation=45)
self.ax_per_slide_np.set_xlabel("Histo IDs")
self.ax_per_slide_np.set_ylabel("Progress / No Progress")
self.ax_per_slide_np.legend()

for idx, value in enumerate(progress_np):
    self.ax_per_slide_np.text(
        idx+1, 0.05, str(round(value, 2)),
        color="black", ha="center"
    )
for idx, value in enumerate(average_progress):
    self.ax_per_slide_np.text(
        idx+1, -0.15, str(round(value, 2)),
        color="black", ha="center"
    )
for idx, value in enumerate(true_progress):
    if value == 0:

```

```

        self.ax_per_slide_np.text(
            idx+1, -0.2, str(value),
            color="black", ha="center"
        )
    elif value == 1:
        self.ax_per_slide_np.text(
            idx+1, -0.2, str(value),
            color="red", ha="center"
        )

# Get optimal threshold per slide.
progress = progress_p + progress_np
true_progress = [1 for _ in progress_p] + [0 for _ in progress_np]
fpr, tpr, thresholds = roc_curve(true_progress, progress)
aucurve = auc(fpr, tpr)
idx = np.argmax(tpr - fpr)
optimal_thresh = thresholds[idx]
print(optimal_thresh)
self.ax_per_slide_p.plot(
    [0, 13], [optimal_thresh, optimal_thresh], "r--"
)
self.ax_per_slide_np.plot(
    [0, 13], [optimal_thresh, optimal_thresh], "r--"
)
self.ax_per_slide_p.text(
    15, optimal_thresh, str(optimal_thresh),
    color="red", ha="center"
)

if __name__ == "__main__":
    if len(argv) < 3:
        print("""
        You gave too less arguments.
        1. Name of the model to test
        """)
        sys.exit()

    MODEL_NAME = argv[1]
    NORMALIZATION = "non_normalized"
    pickle_src = "pickles/5x/{}/testing/".format(NORMALIZATION)

    evaluator = Evaluator(
        pickle_src, MODEL_NAME, NORMALIZATION
    )

    evaluator.open_pickles()
    evaluator.test_model()
    evaluator.build_confusion_matrix()
    evaluator.build_boxplot()
    evaluator.plot_specifications()
    evaluator.progress_per_slide()
    evaluator.save()

```

7.3.12. Skript 12: Markieren der als „Progress“ eingestufteten Tiles

```

import os
import pickle

import numpy as np # type: ignore
import pandas as pd # type: ignore
from openslide import OpenSlide # type: ignore
from PIL import Image, ImageDraw # type: ignore

```

```

from tensorflow.keras.models import load_model # type: ignore

os.environ["TF_CPP_MIN_LOG_LEVEL"] = '2'

pickle_src = "pickles/progress_detection/largeimage/"

# TODO: Specify Slide Path.

```

```

SLIDE_PATH = (
    "PATH/TO/SLIDES/"
)
MODEL_NAME = "model_14_03_2022_5_non_normalized_64_10_0.00001.h5"
DOWNSAMPLE = 0.05

with open(pickle_src + "x_test_after_vgg_0.pickle", "rb") as file:
    x_test_after_vgg = pickle.load(file).astype("float32")
with open(pickle_src + "y_test.pickle", "rb") as file:
    y_test = pickle.load(file).astype("float32")
y_test = np.squeeze(y_test, axis=1)
with open(pickle_src + "filenames_test.pickle", "rb") as file:
    filenames = pickle.load(file)

print("Successfully loaded pickle files")
print(x_test_after_vgg.shape)

model = load_model(MODEL_NAME)
print("Starting to test model")
pred = model.predict(x_test_after_vgg, verbose=1)
y_pred = pd.Series(np.argmax(pred, axis=1), name="pred").tolist()

slides = [slide for slide in os.listdir(SLIDE_PATH)
          if slide[-5:] == ".ndpi"]

for slide in slides:
    print(slide)

    pred_progress_dir = (
        SLIDE_PATH + "tiles_by_largeimage_jpeg/pred_progress/" +
        slide[-5:]
    )
    if not os.path.isdir(pred_progress_dir):
        os.mkdir(pred_progress_dir)

    pred_non_progress_dir = (
        SLIDE_PATH + "tiles_by_largeimage_jpeg/pred_non_progress/" +
        slide[-5:]
    )
    if not os.path.isdir(pred_non_progress_dir):
        os.mkdir(pred_non_progress_dir)

    # Make a thumbnail that we can draw on.
    wsi = OpenSlide(SLIDE_PATH + slide)
    width = wsi.dimensions[0]
    height = wsi.dimensions[1]

    small_slide = wsi.get_thumbnail((width*DOWNSAMPLE, height*DOWNSAMPLE))
    small_slide.save(SLIDE_PATH + "thumbnails/{}.jpeg".format(slide[-5:]))
    small_slide = Image.open(
        SLIDE_PATH + "thumbnails/{}.jpeg".format(slide[-5:]))
    marked = ImageDraw.Draw(small_slide)

)

```

```

# Get all tiles for current slide that indicate non_progress
tiles_np = [filenames[i] for i in range(len(filenames))
            if filenames[i].split("_")[0] == slide[:-5]
            and y_pred[i] == 0]

# Get all tiles for current slide that indicate progress
tiles = [filenames[i] for i in range(len(filenames))
        if filenames[i].split("_")[0] == slide[:-5]
        and y_pred[i] == 1]

for tile in tiles:
    try:
        src = SLIDE_PATH + "tiles_by_largeimage_jpeg/" + tile
        dest = (
            SLIDE_PATH + "tiles_by_largeimage_jpeg/pred_non_progress/" +
            slide[:-5] + "/" + tile
        )
        os.rename(src, dest)
    except:
        print("could not find tiles for ", slide)
    x_pos = int(tile.split("_")[1])
    y_pos = int(tile.split("_")[2][:-5])

    x1 = (x_pos * 8 * 224) * DOWNSAMPLE
    x2 = ((x_pos * 8 * 224) + (8 * 224)) * DOWNSAMPLE
    y1 = (y_pos * 8 * 224) * DOWNSAMPLE
    y2 = ((y_pos * 8 * 224) + (8 * 224)) * DOWNSAMPLE

    marked.rectangle((x1, y1, x2, y2), outline="yellow", width=5)
    small_slide.save(
        SLIDE_PATH + "thumbnails/all_marked_{}.jpg".format(slide[-5:]))
)

for tile in tiles_np:
    try:
        src = SLIDE_PATH + "tiles_by_largeimage_jpeg/" + tile
        dest = (
            SLIDE_PATH + "tiles_by_largeimage_jpeg/pred_non_progress/" +
            slide[:-5] + "/" + tile
        )
        os.rename(src, dest)
    except:
        print("could not find tiles for ", slide)
    x_pos = int(tile.split("_")[1])
    y_pos = int(tile.split("_")[2][:-5])

    x1 = (x_pos * 8 * 224) * DOWNSAMPLE
    x2 = ((x_pos * 8 * 224) + (8 * 224)) * DOWNSAMPLE
    y1 = (y_pos * 8 * 224) * DOWNSAMPLE
    y2 = ((y_pos * 8 * 224) + (8 * 224)) * DOWNSAMPLE

    marked.rectangle((x1, y1, x2, y2), outline="green", width=5)
    small_slide.save(
        SLIDE_PATH + "thumbnails/all_marked_{}.jpg".format(slide[-5:]))
)

```

7.3.13. Skript 13: Testen der Risikofaktoren auf Signifikanz und Logistische Regression

```

from statistics import median # type: ignore

import matplotlib.pyplot as plt # type: ignore
import numpy as np # type: ignore
import pandas as pd # type: ignore

```

```

import seaborn as sns # type: ignore
from scipy.stats import (chi2_contingency, kstest, levene,
                        mannwhitneyu, ttest_ind) # type: ignore
from sklearn import metrics # type: ignore
from sklearn.linear_model import LogisticRegression # type: ignore

```



```
from sklearn.model_selection import train_test_split # type: ignore
```

```
class Plotter:
```

```
def __init__(self, df):
```

```
    self.df = df
    self.metric = [
        "Age", "Tumor thickness [mm]",
        "Tumor diameter [cm]", "Safety margin [cm]",
        "Grading"
    ]
    self.significant = []
```

```
def plot_metric(self, column):
```

```
    print(column)
    df_progression = self.df[self.df["progress"] == 1]
    df_no_progression = self.df[self.df["progress"] == 0]

    progression = df_progression[column].dropna().tolist()
    no_progression = df_no_progression[column].dropna().tolist()

    med_pro = median(progression)
    med_no_pro = median(no_progression)

    fig, axs = plt.subplots(1, 2, figsize=(20, 10))

    axs[0].bar(["no_progression", "progression"], [med_no_pro, med_pro])
    axs[0].text(0, med_no_pro, str(round(med_no_pro, 2)), ha="center")
    axs[0].text(1, med_pro, str(round(med_pro, 2)), ha="center")
    axs[0].set_ylabel("Median {}".format(column))
    axs[0].set_title("Median {}".format(column))
```

```
    axs[1].boxplot([no_progression, progression])
    axs[1].set_xticks([1, 2], ["no_progression", "progression"])
    axs[1].set_title("{} Distribution".format(column))
```

```
# Test if normal distribution.
```

```
if (
    kstest(no_progression, "norm").pvalue <= 0.05 and
    kstest(progression, "norm").pvalue <= 0.05
):
    # T-Test not possible.
    print("No normal distribution. Using Mann Whitney U Test.")
    p_value = mannwhitneyu(no_progression, progression).pvalue
    test = "Mann Whitney U"
```

```
else:
```

```
    # T-Test is possible.
    print("Normal distribution. Using t-test.")

    # Test if variances are equal.
    if levene(no_progression, progression).pvalue <= 0.05:
        print("Variances unequal")
        p_value = ttest_ind(no_progression,
                            progression,
                            equal_var=False).pvalue
        test = "T-Test with unequal variances"
    else:
        print("Variances equal.")
        p_value = ttest_ind(no_progression, progression).pvalue
        test = "T-Test"
```

```
if p_value <= 0.05:
    sig = "Significant."
    self.significant.append(column)
else:
    sig = "Not significant."
```

```
msg = "p-value = {} ({}); {}".format("%.14f" % p_value, test, sig)
msg_y = med_pro / 10
axs[0].text(0, (-1 * msg_y), msg, fontsize=10)
print(msg)
```

```
if p_value <= 0.05:
    fig.savefig("plots/significant/{}".format(column))
else:
    fig.savefig("plots/not significant/{}".format(column))
plt.clf()
```

```
def contingency(self, column):
```

```
    contingency = pd.crosstab(
        self.df[column],
        self.df["progress"],
        margins=True
    )
    value = np.array([contingence.iloc[0][0:5].values,
                     contingency.iloc[1][0:5].values])
    p_value = chi2_contingency(value)[1]

    return contingency, p_value
```

```
def get_OR(self, cont):
```

```
    risk_rec = cont[1][1]
    risk_no_rec = cont[0][1]
    no_risk_rec = cont[1][0]
    no_risk_no_rec = cont[0][0]

    return (risk_rec * no_risk_no_rec) / (no_risk_rec * risk_no_rec)
```

```
def plot_nominal(self, column):
```

```
    values = self.df[column].dropna().unique().tolist()
    values = [int(value) for value in values]
    values.sort()
    progression_risks = []
    progressions = []
    value_sums = []
    for value in values:
        current_df = self.df[self.df[column] == value]
        value_sum = len(current_df.index)

        progression = len(current_df[current_df["progress"] == 1].index)
        risk = progression / value_sum
        progression_risks.append(risk)
        progressions.append(progression)
        value_sums.append(value_sum)
```

```
fig, axs = plt.subplots(1, 2, figsize=(20, 10))
```

```
x = [str(value) for value in values]
axs[0].bar(x, progression_risks)
```

```
if max(progression_risks) >= 0.5:
    axs[0].set_ylim((0, 1.0))
else:
    axs[0].set_ylim((0, 0.5))
```

```
for index, risk in enumerate(progression_risks):
    axs[0].text(index, risk, str(round(risk, 2)), ha="center")
```

```
for index, progression in enumerate(progressions):
    if max(progression_risks) >= 0.5:
        y = -0.1
    else:
        y = -0.05
    value_sum = value_sums[index]
    txt = f"n(total) = {value_sum}\nn(progression) = {progression}"
    axs[0].text(index, y, txt, ha="center", va="center", fontsize=7.5)
```

```

    axes[0].set_title("Risk for Progression in {}".format(column))
    axes[0].set_ylabel("Risk for Progression")

    cont, p_value = self.contingence(column)
    axes[1].text(0, 0.5, str(cont), va="center", fontsize=15)
    if p_value <= 0.05:
        msg = "p-value = {}; Significant.".format("%.8f" % p_value)
        self.significant.append(column)
    else:
        msg = "p-value = {}; Not significant.".format("%.8f" % p_value)
    axes[1].text(0, 0.1, msg, va="center", fontsize=15)
    if len(progression_risks) == 2:
        rr = progression_risks[1] / progression_risks[0]
        OR = self.get_OR(cont)
        axes[1].text(0, 0.2, "RR: {}".format(rr), va="center", fontsize=15)
        axes[1].text(0, 0.25, "OR: {}".format(OR), va="center", fontsize=15)
    axes[1].axis("off")

    if p_value <= 0.05:
        fig.savefig("plots/significant/{}".format(column))
    else:
        fig.savefig("plots/not significant/{}".format(column))

plt.clf()

def log_reg(self):
    x = self.df.fillna(-1).drop("progress", axis=1).values
    # x = x.fillna(x.mean())
    y = self.df["progress"].values
    x_train, x_test, y_train, y_test = train_test_split(
        x, y,
        test_size=0.2,
        random_state=0
    )
    logreg = LogisticRegression(max_iter=500)
    logreg.fit(x_train, y_train)
    y_pred_proba = logreg.predict_proba(x_test)[::, 1].tolist()

    # Create a ROC Curve.
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)

    idx = np.argmax(tpr - fpr)
    thresh = thresholds[idx]
    print(thresh)

    y_pred = [
        1 if float(y_pred_proba[i]) >= thresh
        else 0 for i in range(len(y_pred_proba))
    ]
    # y_pred = logreg.predict(x_test)
    print("acc score: ", str(logreg.score(x_test, y_test)))

    # Create a confusion matrix.
    cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
    print(cnf_matrix)

```

```

sens = cnf_matrix[1][1] / sum(cnf_matrix[1])
spec = cnf_matrix[0][0] / sum(cnf_matrix[0])

# Create a heatmap.
class_names = ["No Progression", "Progression"]
fig, axes = plt.subplots(1, 2, figsize=(25, 10))
tick_marks = np.arange(len(class_names))
axes[1].set_xticks(tick_marks, class_names)
axes[1].set_yticks(tick_marks, class_names)

sns.heatmap(
    pd.DataFrame(cnf_matrix),
    annot=True,
    cmap="YlGnBu",
    fmt='g'
)

axes[1].set_title(
    "Confusion matrix test set (n = {})".format(len(y_pred))
)
axes[1].set_ylabel("Actual label")
axes[1].set_xlabel("Predicted label")

# Plot ROC Curve.
axes[0].plot(fpr, tpr, label="auc=" + str(auc))
axes[0].legend(loc=4)
msg1 = "Sensitivity: " + "%.2f" % sens
msg2 = "Specificity: " + "%.2f" % spec
axes[0].text(0.75, 0.04, msg1, va="center", fontsize=10)
axes[0].text(0.75, 0.02, msg2, va="center", fontsize=10)
axes[0].set_title("ROC Curve")
axes[0].set_ylabel("True Positive Rate")
axes[0].set_xlabel("False Positive Rate")

fig.suptitle("Logistic Regression of significant variables")
fig.savefig("plots/log_reg")
plt.clf()

if __name__ == "__main__":
    df = pd.read_excel("sheets/clinical_data.xlsx",
        usecols="A:H,M,AC:AR")

    plotter = Plotter(df)

    for column in plotter.df:
        if column == "progress":
            continue
        elif column in plotter.metric:
            plotter.plot_metric(column)
        else:
            plotter.plot_nominal(column)

    plotter.log_reg()

```

8. Vorabveröffentlichungen von Ergebnissen

Bisher wurden keine Ergebnisse dieser Arbeit veröffentlicht.