Diplomarbeit

Melanom-Naevus Klassifizierung mit neuronalen Netzwerken und digitalen Bildern

eingereicht von

Michael Peter Siemmeister

zur Erlangung des akademischen Grades

Doktor(in) der gesamten Heilkunde
(Dr. med. univ.)

an der

Medizinischen Universität Graz

ausgeführt am

Gottfried Schatz Forschungszentrum
für zelluläre Signaltransduktion, Stoffwechsel und Altern
Lehrstuhl für Biophysik

unter der Anleitung von

Ao.Univ.-Prof. Mag. Dr.rer.nat. Helmut Ahammer

Graz, am 10. März 2020

Eidesstaatliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe, andere als die angegebenen Quellen nicht verwendet habe und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 10. März 2020 Michael Peter Siemmeister eh.

Danksagungen

Ich möchte mich an dieser Stelle bei meiner Familie bedanken, die mich in all den Jahren sowohl zwischenmenschlich als auch finanziell unterstützt hat – das ist nicht selbstverständlich. In der Eierstocklotterie habe ich echt Glück gehabt.

Weiters möchte ich ich mich bei Prof. Helmut Ahammer bedanken, der es mir ermöglicht hat, ein spannendes Thema zu verfolgen. Neben der fachlichen Expertise und Hilfestellungen möchte ich mich außerdem für die unbürokratische Zusammenarbeit und den Fokus aufs Wesentliche bedanken.

Graz, am 10. März 2020

Inhaltsverzeichnis

Abkürzungsverzeichnis

random	Zufallsklassifikator *random*. 24, 26–30, 36, 37
simple	*simple*-Modell. 19, 20, 24–30, 36
twoLayer	*twoLayer*-Modell. 19, 20, 24–36
AUC	*area under the curve*, die Fläche unter der receiver-operating-characteristic. 24–27, 36–38
aveP	*average precision*. 24–27, 36, 38
CNN	Convolutional Neural Network. 18, 36
FN	*false negative*, in Wirklichkeit positiv und vom Test als negativ eingestuft. 23
FP	*false positive*, in Wirklichkeit negativ und vom Test als positiv eingestuft. 23
FPR	falsch-positive Rate. 23, 24
ISIC2016	ISIC-Challenge-2016-Part-3-Datensatz. 22–24
MClass	MClass-Datensatz. 22–24, 36
PRC	*precision-recall*. 24, 25, 27–32, 34, 37
Prec	*precision*. 24

ROC *receiver-operating-characteristic*. 24, 25, 27, 28, 30–32, 35

SE Sensitivität. 23, 24

SP Spezifität. 23, 24

TN *true negative*, in Wirklichkeit negativ und vom Test als negativ eingestuft. 23

TP *true positive*, in Wirklichkeit positiv und vom Test als positiv eingestuft. 23, 24

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Melanome zählen zu den aggressivsten, malignen Hauttumoren. Allein in den USA sterben jedes Jahr cirka 10.000 Menschen an Melanomen, daher ist Früherkennung durch Dermatologen und Dermatologinnen von zentraler Bedeutung. Doch selbst Experten und Expertinnen können sich irren, weshalb Bedarf an automatisierter Bilderkennung besteht. Künstliche neuronale Netzwerke können in Kombination mit leistungsstarker Hardware digitale Bilder in einigen Spezialgebieten besser klassifizieren als Menschen. Das Klassifizieren von dermatologischen Läsionen ist ein Gebiet aktiver Forschung. Es gibt bereits neuronale Netzwerke, die Melanome an bestimmten Datensätzen besser klassifizieren können als Dermatologen und Dermatologinnen. Bislang gibt es jedoch kein einheitliches Verfahren, verschiedene Modelle miteinander und mit Dermatologen und Dermatologinnen zu vergleichen, auch ist die Reproduzierbarkeit der Modelle gering. In dieser Arbeit werden zwei Convolutional-neural-network-Modelle basierend auf ResNet50 zur Melanom-Naevus-Klassifizierung mit open-source Software implementiert. Speziell wird die Bibliothek TensorFlow verwendet. Die Modelle werden mit öffentlich zugänglichen Melanom- und Naevus-Bildern trainiert und mit zwei öffentlich zugänglichen Datensätzen ausgewertet. Bei diesen Datensätzen handelt es sich um MClass sowie den Test-Datensatz der ISIC Challenge 2016. Das bessere der beiden Modelle erreicht am MClass-Datensatz eine AUC von 83,8%, eine average precision von 64,2% und wird mit den öffentlich zugänglichen Ergebnissen von 157 Dermatologinnen und Dermatologen am selben Datensatz verglichen. Die Modelle werden mitsamt dem Quellcode zum Trainieren und Auswerten auf GitHub veröffentlicht und stellen eine Grundlage für weitere Arbeiten dar.

Abstract

Melanoma is one of the most aggressive, malignant skin tumours. In the United States alone, approximately 10,000 people die from melanoma each year, therefore early detection by dermatologists is essential. But even experts can be misinformed, which is why there is a need for automated image recognition. Artificial neural networks in combination with powerful hardware can classify digital images better than humans in some specific fields. Classifying dermatological lesions using these artificial neural networks is an area of active research. There are already artificial neural networks that can better classify melanomas on certain data sets than dermatologists. To date, however, there is no uniform procedure for comparing different models with each other and with dermatologists' findings. The reproducibility of the models is low. In this work, two convolutional-neural-network models based on ResNet50 for melanoma-nevus classifications are implemented with open-source software using the library TensorFlow. The models are trained with publicly available melanoma and nevus images. The models are evaluated using two publicly available data sets. These data sets are MClass and the test data set of ISIC Challenge 2016. The better model achieved an AUC of 83.8% on the MClass data set and an average precision of 64.2%. This model is compared with publicly-available results of 157 dermatologists on the same data set. The models and the source code for training and evaluation are published on GitHub and represent a basis for further research and development in this field.

Angaben von bereits erfolgten Veröffentlichungen

Diese Arbeit wurde bis heute, 10. März 2020, noch nicht veröffentlicht.

1. Einleitung

Melanozytäre Naevi oder Naevi (auch Nävi) sind gutartige melanozytäre Tumoren der Haut, die umgangssprachlich Muttermale genannt werden (Kumar u. a., 2014). Es gibt verschiedene Unterformen wie unter anderem zum Beispiel den Junktionsnaevus, Compound-Naevus, Halonaevus und konnatalen Naevus. Selten können Naevi auch zu Melanomen entarten. Die meisten Naevi werden im Laufe des Lebens erworben und sind an ihrer meist braunen Farbe und gleichmäßigen Pigmentierung erkennbar. Zudem weisen sie meist einen gut abgrenzbaren Rand auf und nehmen mit der Zeit nicht an Größe zu.

Eine Sonderform der Naevi sind die dysplastischen Naevi, die eine Vorstufe von Melanomen sein können. In dieser Arbeit werden dysplastische Naevi als maligne eingestuft, da dies auch im ISIC-Archive der Fall ist (International Society for Digital Imaging of the Skin, 2019a).

Bei malignen Melanomen oder kurz Melanomen handelt es sich um bösartige Tumoren der Haut, die melanozytären Ursprungs sind (Stewart, Wild u. a., 2014). Die Inzidenz in den USA beträgt 21,8 pro 100.000 Einwohner und Jahr (Centers for Disease Control and Prevention, 2019). Melanome führen in den USA zu rund 9.000 Todesfällen pro Jahr. Sie treten häufig an sonnenexponierten Stellen der Haut auf, können jedoch auch auf Schleimhäuten – wie zum Beispiel bei anorektalen Melanomen – entstehen. Die vorliegende Arbeit ist auf Melanome der Haut beschränkt. Ein zentraler Risikofaktor für die Entstehung von Melanomen ist eine UV-Exposition ohne ausreichenden Sonnenschutz. Ausgehend davon zielt die Melanomprävention auf verkürzte Sonnen-

aufenthalte und ausreichenden Sonnenschutz ab. Die Morphologie von Melanomen ist im Vergleich zu jener der Naevi unregelmäßiger - sie sind asymmetrisch und haben keinen glatten Rand. Auch können Melanome mehrere Farben aufweisen und zudem wachsen. Die Therapie der Wahl für Melanome ist die chirurgische Exzision, weshalb Früherkennung von zentraler Bedeutung ist. Was metastasierte Melanome anbelangt, so sind diese gegenüber klassischer Strahlen- oder Chemotherapie meist resistent. Neue Therapien zielen unter Verwendung von monoklonalen Antikörpern auf die spezielle Signaltransduktionskaskaden in Melanomen ab (Domingues u. a., 2018).

Eine Erleichterung für die Unterscheidung von Melanomen und Naevi bietet Dermatologen und Dermatologinnen der Einsatz von Dermoskopen. (Kittler u. a., 2002). Bei dieser Technik wird die Läsion beleuchtet und vergrößert (Binder u. a., 1995).

Künstliche neuronale Netzwerke sind erstmals von McCulloch und Pitts, 1943 beschrieben worden. Sie stellen einen Versuch dar, die menschlichen Nervenzellen nachzuahmen. Eine Verbesserung der Bildklassifikationsfähigkeiten konnte durch bessere Algorithmen und Hardware erzielt werden. (Krizhevsky, Sutskever und Hinton, 2012). Neuronale Netzwerke weisen diverse Einsatzgebiete auf: Sie haben die besten Go-Spieler der Welt geschlagen (Silver u. a., 2016) und klassifizieren Bilder teilweise besser als Menschen (Cireşan, Meier und Schmidhuber, 2012).

Das Gebiet der automatischen Melanom-Naevus Klassifizierung ist Teil aktiver Forschung. 2017 konnte gezeigt werden, dass neuronale Netzwerke Hautkrebs so gut wie Dermatologen und Dermatologinnen erkennen können (Esteva u. a., 2017). Seit 2016 finden auf diesem Gebiet jährlich Wettbewerbe der International Skin Imaging Collaboration statt (International Society for Digital Imaging of the Skin, 2019a; Gutman u. a., 2016; Codella u. a., 2019), im Zuge derer verschiedene Netzwerk-Architekturen ausprobiert werden (Yu u. a., 2016; Menegola u. a., 2017). Das Netzwerk von Menegola u. a., 2017 ist veröffentlicht worden. Brinker vergleicht neuronale Netzwerke mit 157 Dermatologinnen und Dermatologen (Brinker, Hekler, A. H. Enk u. a., 2019). Bislang gibt es jedoch noch kein veröffentlichtes Netzwerk, das mit Dermatologen und Dermatologinnen verglichen worden ist. Des Weiteren gibt es noch keinen veröffentlichten Quellcode

zum Training neuronaler Netzwerke an öffentlich zugänglichen Melanom-Bildern und Auswertung dieser trainierten Netzwerke mit öffentlichen Test-Datensätzen.

Ziel dieser Arbeit ist die Implementierung eines neuronalen Netzwerk-Modells zur Melanom-Naevus-Klassifizierung unter Verwendung von *open-source* Software, sowie dessen Training und Auswertung anhand öffentlich zugänglicher Melanom- und Naevus-Bilder. Das Modell wird mit 157 Dermatologen und Dermatologinnen verglichen, veröffentlicht und kann eine Grundlage für weitere Arbeiten darstellen.

2. Material und Methoden

2.1. Modell

Das *neuronale Netzwerk*-Modell dieser Arbeit soll zwischen malignen und benignen melanozytären Läsionen unterscheiden. Den Input für das Modell stellen digitale Bilder dieser Läsionen dar. Mathematisch formuliert würde ein perfektes Modell der Funktion

$$f(\mathbf{x}) = \begin{cases} 1 & \text{wenn } \mathbf{x} \text{ eine maligne Läsion darstellt,} \\ 0 & \text{wenn } \mathbf{x} \text{ eine benigne Läsion darstellt,} \end{cases} \tag{2.1}$$

entsprechen. $\mathbf{x}$ ist ein Vektor, der die Bildinformation darstellt. Die Aufgabe ist, die Funktion f mit einem neuronalen Netzwerk zu approximieren. Das neuronale Netzwerk soll jeden Input mit einem Score s bewerten, der für benigne und maligne Bilder deutlich unterscheidbare Werte annehmen soll. Allgemein handelt es sich bei f um einen binären Klassifikator. Binäre Klassifikatoren ordnen einem Argument oder Input einen Score s zu. Oft wird dieser Score durch die logistische Funktion

$$g(x) = \frac{1}{1 + e^{-a(x-b)}} \tag{2.2}$$

in das Intervall $[0, 1]$ abgebildet (Codella u. a., 2019). Die Parameter a und b können frei gewählt werden, ihre Wahl kann jedoch die Metriken zur Auswertung beeinflussen.

Neuronale Netzwerke wurden erstmals von McCulloch und Pitts, 1943 beschrieben. Sie stellen einen Versuch dar, die menschlichen Nervenzellen nachzuahmen. Die vereinfachte Nervenzelle hat n Inputs I_i, die mit Gewichten w_i multipliziert und dann summiert

werden. Der Output O der Nervenzelle wird durch eine nichtlineare Schwellenfunktion h berechnet. Mathematisch formuliert wird die Nervenzelle durch

$$O\left(I_i\right) = h\left(\sum_i^n I_i w_i + w_0\right) \tag{2.3}$$

dargestellt. w_0 wird als *bias* bezeichnet. Ein einfaches neuronales Netzwerk würde nur aus einer Nervenzelle – auch Neuron genannt – bestehen. Beim Training des Netzwerkes wird versucht, die optimalen Werte der Gewichte w_i zu finden, um jedem Input I_i den gewünschten Output d zuzuordnen. Es wird also versucht, eine Funktion $\mathcal{L}\left(d, O; w_i\right)$ zu minimieren. $\mathcal{L}$ ist eine Funktion von d und O mit den zu minimierenden Parametern w_i. Diese Funktion $\mathcal{L}$ wird *loss*-Funktion genannt.

Neuronale Netzwerke können aus mehreren Neuronen bestehen. Diese werden meist in Schichten, den *Layers*, aneinandergereiht (Rosenblatt, 1961). Die Neuronen in einem *fully-connected* Layer bekommen als Input den Output von Neuronen im vorherigen Layer. Die Neuronen eines Layers bekommen jedoch keinen Input von Neuronen desselben Layers. Ein Modell mit mehreren fully-connected Layers nennt man *multilayer-perceptron*. Klassischerweise sind beim multilayer-perceptron alle Neuronen eines Layers mit allen Neuronen des nächsten Layers verbunden.

Zum Minimieren der loss-Funktion wird *stochastic-gradient-descent* verwendet (Kiefer, Wolfowitz u. a., 1952). Dabei wird die loss-Funktion nicht mit allen Bildern im Trainings-Datensatz, sondern nur mit einer Teilmenge, dem *batch*, evaluiert. Die Größe, auch *batch-size* genannt, bestimmt, wie viele Bilder in einem Batch sind. Die batch-size kann dabei willkürlich gewählt werden. Parameter, die willkürlich gesetzt werden können, bezeichnet man als *Hyperparameter*. Bei stochastic-gradient-descent werden die Gewichte w_i nach der Regel

$$w_i^{neu} = w_i^{alt} - \alpha \frac{\partial \mathcal{L}}{\partial w_i} \tag{2.4}$$

verändert. Dabei wird der Gradient von $\mathcal{L}$ nach den w_i gebildet. α ist willkürlich zu wählen und wird als *learning-rate* bezeichnet, welche ein Hyperparameter ist. Neben dem stochastic-gradient-descent gibt es noch weitere Methoden die optimalen w_i zu finden (Duchi, Hazan und Singer, 2011).

Um das Finden der Gewichte w_i zu erleichtern, wird *backpropagation* genutzt (Werbos, 1974). Dabei werden die Gewichte w_i Layer für Layer modifiziert. Dabei spart man Rechenaufwand, da man die selben Berechnungen häufig durchführt und die Ergebnisse speichern und wiederverwenden kann. Backpropagation ist eine Anwendung des *dynamic-programming*-Prinzips (Bellman u. a., 1954).

Beim Training neuronaler Netzwerke werden dieselben Trainingsbilder in sogenannten *Epochen* erneut durchlaufen.

In dieser Arbeit wird das neuronale Netzwerk durch ein Convolutional Neural Network (CNN) dargestellt (LeCun u. a., 1998). Ein CNN ist ein neuronales Netzwerk, das nach der mathematischen Operation der Konvolution arbeitet. Durch Training lernt das CNN relevante *features* wie Konturen und Texturen zu erkennen.

In einem *convolutional-layer* werden *kernels*, 3-dimensionale Zahlenarrays mit Werten w_i, verwendet. Der Input für ein CNN-Layer ist ebenfalls 3-dimensional. Die ersten beiden Input-Dimensionen des ersten Layers stellen Breite und Höhe des Bildes dar. Bei den folgenden Layers stellen die ersten beiden Input-Dimensionen die ersten beiden Output-Dimensionen des vorherigen Layers dar. Die dritte Input-Dimension, die Tiefe d, ist die Anzahl der kernels im vorherigen Layer. Beim ersten Layer ist die Input-Tiefe die Anzahl der Farbkanäle des Input-Bildes. Sollte der Input 2-dimensional sein, wird die Tiefe 1 gesetzt.

Die kernels weisen meist die Größe $3 \times 3 \times d$ oder $5 \times 5 \times d$ auf.

Bei der Berechnung des Outputs werden die kernels schrittweise über den Input geschoben. Anschließend wird an jeder Stelle die Konvolution berechnet und einer nichtlinearen Funktion zugeführt. Die Konvolution ist die Summe des elementbasierten Produktes von Input und kernel. Die Anzahl der Pixel, um die die kernels bei jedem Schritt verschoben werden, wird als *stride* bezeichnet. Die Output-Tiefe eines CNN-Layers ist gleich der Anzahl der kernels, da jeder kernel ein 2-d-Zahlenarray errechnet. Die Werte w_i der kernels werden durch stochastic-gradient-descent iterativ gefunden.

Mit der seit Pratt, 1993 bekannten Technik des *transfer-learnings* ist es möglich einen Großteil der Layer eines bereits trainierten Modells wiederzuverwenden. Dabei werden die *convolutional-layers* eines bereits trainierten CNNs als Filter benutzt, der eine bessere Respräsentation des Input-Bildes darstellen soll.

Was CNNs anbelangt, so gibt es verschiedene Architekturen. Hier wird ein *ResNet50*-Modell als Grundlage für transfer-learning verwendet (He u. a., 2016). Das ResNet50 ist mit den ImageNet-Bildern trainiert und schafft state-of-the-art-Ergebnisse (He u. a., 2016).

In dieser Arbeit werden zwei Modelle trainiert und evaluiert. Beide nutzen ein am-ImageNet-trainiertes ResNet50 als Grundlage. Bei beiden wird das letzte ResNet50-Layer, welches der eigentlichen Klassifizierung am ImageNet-Datensatz dient, verworfen.

Beim ersten Modell, dem *simple*-Modell (*simple*), wird das verworfene Layer durch ein fully-connected Layer mit einem Neuron ersetzt.

Beim zweiten Modell, dem *twoLayer*-Modell (*twoLayer*), wird das verworfene Layer durch ein fully-connected Layer mit 1024 Neuronen ersetzt. An dieses Layer mit 1024 Neuronen wird noch ein fully-connected Layer mit einem Neuron angeschlossen.

2.2. Implementierung

Die Implementierung der Modelle erfolgt in der Programmiersprache *Python* (Van Rossum und Drake Jr, 1995). Die Bibliothek *TensorFlow* (Abadi u. a., 2015) und deren Erweiterung *Keras* (Chollet u. a., 2015) stellen die benötigten Klassen, Funktionen und Variablen zur Verfügung. Weitere benutzte Bibliotheken sind unter anderem *Pandas* (McKinney u. a., 2010) und *Jupyter* (Kluyver u. a., 2016). Eine genaue Auflistung der verwendeten Bibliotheken ist im Quellcode zu finden.

Das Trainieren neuronaler Netzwerke erfordert Rechenleistung. Daher wird eine virtuelle Maschine der *Google Cloud Platform* (Google LLC, 2019) mit 26 GB RAM und einer *NVIDIA-Tesla-K80-Grafikkarte* zum Trainieren der beiden Modelle verwendet.

Die Trainingsbilder werden vor dem Training mit einer zufälligen Rotation um bis zu 359 Grad und einer zufälligen horizontalen Spiegelung augmentiert. Durch diese Augmentierung sollen die Modelle robust gegenüber Rotation und Spiegelung gemacht werden. Zusätzlich werden die Bilder durch eine für ResNet50 spezifische Funktion modifiziert (Chollet u. a., 2015). Dabei werden die Bilder unter anderem auf eine Größe von 224 mal 224 Pixel skaliert.

Beide Modelle werden mit stochastic-gradient-descent trainiert. Der Hyperparameter *momentum* (Rumelhart, Hinton, Williams u. a., 1988) wird auf 0.9 gesetzt. In den ersten drei Epochen werden nur die neu hinzugefügten Layers von *simple* und *twoLayer* trainiert. Dadurch soll verhindert werden, dass ResNet50 seine Fähigkeit, features zu extrahieren, verliert. Nach diesen ersten 3 Epochen wird das gesamte neuronale Netzwerk trainiert.

Die learning-rate ist in den ersten 3 Epochen 10^{-3}, 10^{-4} und 10^{-5}. Danach beginnt sie bei 10^{-5} und fällt exponentiell ab, sodass sie nach je 3 Epochen um den Faktor 10 abnimmt.

Das Training wird manuell beendet, wenn sich der Wert der loss-Funktion am Validierungs-Datensatz nach 10 Epochen nicht mehr verändert. Der Validierungsdatensatz wird in Abschnitt 2.3 behandelt. An diesem wird nach jeder Epoche der Trainingsfortschritt gemessen.

2.3. Datensätze

Als Inputs für die neuronalen Netzwerke werden digitale Bilder benutzt, die nach Dignität in Kategorien eingeteilt werden. Unter Dignität von Tumoren versteht man die Einteilung in gutartige, sprich *benigne*, oder bösartige, sprich *maligne* Tumoren (Kumar u. a., 2014). Es gibt daher die Kategorien *maligne* und *benigne*. Ein Bild wird als maligne oder benigne bezeichnet, wenn es in der jeweiligen Kategorie ist.

Für das Training der neuronalen Netzwerke wird eine Teilmenge der Bilder des *ISIC-Archives* des ISIC Melanoma Project (International Society for Digital Imaging of the Skin, 2019a) verwendet. Das ISIC-Archive enthält Bilder von dermatologischen Läsionen, welche von Dermatologen und Dermatologinnen nach verschiedenen Kriterien gelabelt worden sind. Die Bilder werden nach folgenden Kriterien ausgewählt:

- Die Bilder sind histopathologisch verifiziert.
- Die Bilder sind mit einem Dermoskop aufgenommen worden.
- Die Bilder haben eine bekannte Dignität. Ein Teil der Bilder hat keinen Dignitäts-Eintrag - diese werden nicht verwendet. Im ISIC-Archive sind dysplastische Naevi als maligne eingestuft.
- In den Bildern ist eine melanozytäre Läsion dargestellt. Basaliome, und andere nicht-melanozytäre Läsionen werden daher nicht ausgewählt.
- Die Bilder sind nicht Teil eines der Datensätze, die zur Auswertung verwendet werden.

Aus den Bildern, die die Kriterien erfüllen, werden gleich viele Bilder jeder Kategorie (benigne und maligne) zufällig ausgewählt, sprich es wird ein *undersampling* durchgeführt. Durch dieses undersampling soll ein Bias beim Trainieren der neuronalen Netzwerke vermieden werden.

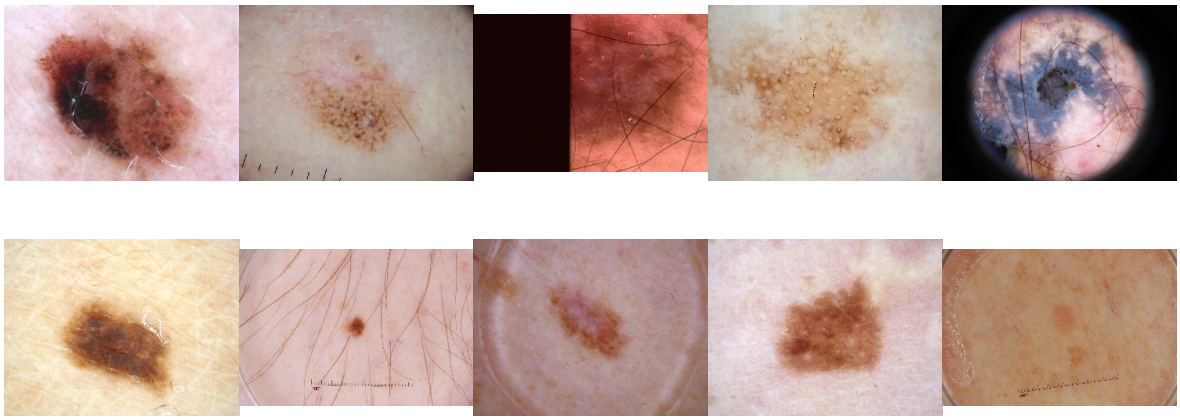

Abbildung 2.1.: Beispielbilder des Trainingsdatensatzes. In der oberen Reihe sind 5 maligne Läsionen dargestellt. In der unteren Reihe sind 5 benigne Läsionen dargestellt. Die Bilder sind zufällig aus dem Trainings-Datensatz ausgewählt worden.

Die Bilder werden zufällig auf 3 Gruppen aufgeteilt. Die erste Gruppe erhält 200 Bilder, die zweite 100 Bilder und die dritte die restlichen Bilder. Die Gruppengrößen werden dabei willkürlich gewählt. Bei dieser Aufteilung wird auf Stratifizierung geachtet. Das bedeutet, dass in jeder Gruppe die Bilder verschiedener Klassen (benigne und maligne) gleich häufig sind. Die erste Gruppe wird Validierungs-Datensatz genannt. Die zweite Gruppe wird Test-Datensatz genannt. Da im Verlauf dieser Arbeit 2 neue Datensätze zur Auswertung gefunden werden konnten, wird der Test-Datensatz nicht weiter verwendet. Die dritte Gruppe besteht aus 3822 Bildern und wird Trainings-Datensatz genannt – dieser wird später zum Training der neuronalen Netzwerke verwendet. In Abbildung 2.1 sind Beispielbilder aus dem Trainings-Datensatz dargestellt.

Das Selektieren, Aufteilen und Herunterladen der Bilder wird vom Python Script melanomaclassification.py (A.3) durchgeführt.

Bei der Auswertung der Modelle werden die Datensätze ISIC-Challenge-2016-Part-3-Datensatz (ISIC2016) und MClass-Datensatz (MClass) (Brinker, Hekler, Hauschild u. a., 2019) verwendet. Beide Datensätze sind Teilmengen des ISIC-Archives. Die Bilder beider Datensätze wurden vom ISIC Melanoma Project (International Society for Digital Imaging of the Skin, 2019a) heruntergeladen.

Der ISIC2016 ist bei der ISIC Challenge 2016 (Gutman u. a., 2016) verwendet worden. Er besteht aus 304 benignen und 75 malignen Bildern.

Der MClass besteht aus 80 benignen und 20 malignen Bildern. An MClass sind von Brinker, Hekler, Hauschild u. a., 2019 157 Dermatologen und Dermatologinnen getestet worden.

2.4. Auswertung

Zur Auswertung der Modelle wird die Programmiersprache Python (Van Rossum und Drake Jr, 1995) zusammen mit den Bibliotheken *Jupyter Lab* (Kluyver u. a., 2016), Pandas (McKinney u. a., 2010), *Matplotlib* (Hunter, 2007) und *Scikit-learn* (Pedregosa u. a., 2011) verwendet.

Es werden die Metriken aus der ISIC 2016 Challenge (Gutman u. a., 2016) verwendet. In der folgenden Beschreibung der Metriken werden die Abkürzungen *true positive*, in Wirklichkeit positiv und vom Test als positiv eingestuft (TP), *false positive*, in Wirklichkeit negativ und vom Test als positiv eingestuft (FP), *true negative*, in Wirklichkeit negativ und vom Test als negativ eingestuft (TN) sowie *false negative*, in Wirklichkeit positiv und vom Test als negativ eingestuft (FN), verwendet. Die Metriken sind wie folgt definiert:

Die Sensitivität (SE) oder Trefferquote:

$$SE = \frac{TP}{TP + FN} \tag{2.5}$$

Die Spezifität (SP):

$$SP = \frac{TN}{TN + FP} \tag{2.6}$$

Die falsch-positive Rate (FPR) ist als $1-$ SP definiert.

Die *precision* (Prec):

$$Prec = \frac{TP}{TP + FP} \tag{2.7}$$

Die *area under the curve*, die Fläche unter der receiver-operating-characteristic (AUC):

$$AUC = \int_0^1 SE(FPR)\,\mathrm{d}FPR \tag{2.8}$$

Die *average precision* (aveP), welche die Fläche unter der *precision-recall* (PRC)-Kurve darstellt:

$$aveP = \int_0^1 Prec(SE)\,\mathrm{d}SE \tag{2.9}$$

Die AUC und aveP werden bei binären Klassifikatoren eingesetzt. Zur Berechnung wird der Schwellenwert T variiert, über dem man den Score eines binären Klassifikators s als positiv wertet. Für einen festgelegten Schwellenwert T' wäre ein Bild in TP, wenn $s > T'$ und das Bild maligne ist. Durch dieses Variieren von T kann man verschiedene Werte für SE, SP und Prec erzeugen. Um binäre Klassifikatoren zu vergleichen, kann man Diagramme zeichen, bei denen man verschiedene Metriken zusammen in ein Diagramm einzeichnet. In einem *receiver-operating-characteristic* (ROC)-Diagramm trägt man die FPR auf der x-Achse und die SE auf der y-Achse auf. In einem PRC-Diagramm trägt man die SE auf der x-Achse und die Prec auf der y-Achse auf.

Dabei gilt es zu beachten, dass die Metriken SE, SP und Prec ohne Festlegung eines Schwellenwertes T' bei binären Klassifikatoren nicht definiert sind.

Die Modelle werden mit dem Zufallsklassifikator *random* (*random*) verglichen. Dieser ordnet jedem Bild einen Score r_i zu. Die r_i sind Zufallsvariablen, die nach einer Gleichverteilung auf $[0, 1]$ verteilt sind. *random* ist kein Tensorflow-Modell.

Die Modelle *twoLayer*, *simple* und der Zufallsklassifikator *random* werden an den beiden Datensätzen ISIC2016 und MClass ausgewertet. Dabei wird von jedem Modell für jedes Bild der Score berechnet. Mit diesen Scores werden anschließend ROC-Diagramme

und PRC-Diagramme gezeichnet. Ebenfalls werden die Metriken aveP und AUC berechnet.

Bei der ISIC Challenge 2016 (Gutman u. a., 2016) sind die teilnehmenden Modelle nach der Metrik aveP gereiht worden. Daher wird aus den Modellen *twoLayer* und *simple* jenes ausgewählt, das eine höhere aveP-Metrik hat – in diesem Fall *twoLayer*. Die Ergebnisse der Dermatologen und Dermatologinnen von Brinker, Hekler, Hauschild u. a., 2019 werden mit denen des *twoLayer*-Modells verglichen. Dabei werden die Sensitivitäten und Spezifitäten der einzelnen Dermatologen und Dermatologinnen mit jenen des *twoLayer*-Modells in das ROC Diagramm als Punkte eingetragen. Die Precisons und Sensitivitäten der Dermatologen und Dermatologinnen werden zusammen mit jenen des *twoLayer*-Modells in ein PRC-Diagramm eingetragen. Mit einem Python Script wird anschließend gezählt, wie viele Dermatologen und Dermatologinnen über, auf oder unter den ROC- und PRC-Kurven des *twoLayer*-Modells liegen.

Bislang ist nicht bekannt, welche Metrik die Dermatologinnen und Dermatologen von Brinker, Hekler, Hauschild u. a., 2019 am ehesten zu optimieren versucht haben. Aus diesem Grund wird hier auf eine induktive statistische Analyse verzichtet.

2.5. Veröffentlichung des Quellcodes und der Modelle

Die Modelle und der Quellcode werden auf der Website GitHub (GitHub Inc., 2019) veröffentlicht.

3. Ergebnisse

3.1. Ergebnisse der beiden Modelle an den Datensätzen ISIC2016 und MClass

Die Ergebnisse der Modelle *twoLayer* und *simple* sind zusammen mit dem Zufallsklassifikator *random* in der Tabelle 3.1 dargestellt.

Am Datensatz ISIC2016 erreicht *twoLayer* eine aveP von 0.52, *simple* 0.48 und *random* 0.25. Am selben Datensatz erreicht *twoLayer* eine AUC von 0.78, *simple* 0.75 und

Tabelle 3.1.: Auswertung der Modelle. In dieser Tabelle sind die Werte der Metriken AUC und AveP für die Modelle *twoLayer* und *simple* und für den *random*-Klassifikator für die Datensätze ISIC2016 und MClass dargestellt.

		AveP	AUC
Datensatz	Modell		
ISIC2016	*twoLayer*	0.5250	0.7845
	simple	0.4782	0.7480
	random	0.2466	0.5011
MClass	*twoLayer*	0.6424	0.8375
	simple	0.6343	0.8381
	random	0.2569	0.5756

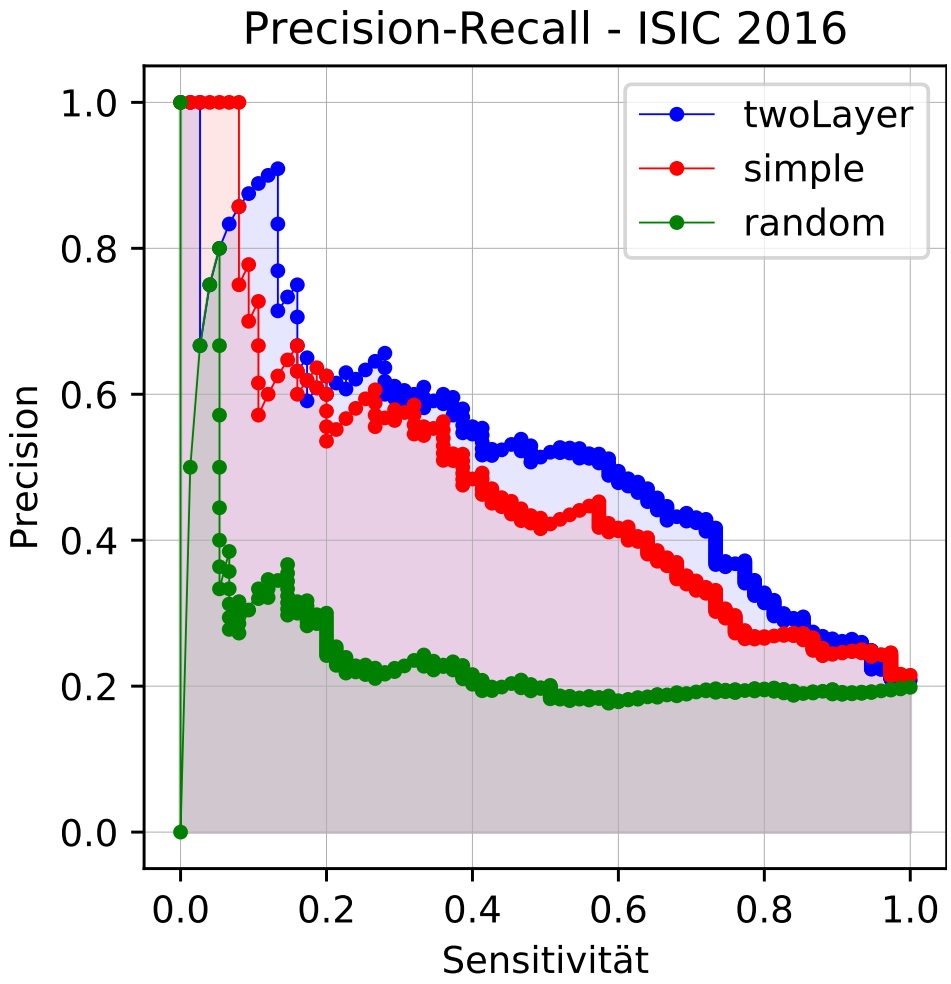

Abbildung 3.1.: PRC-Diagramm des ISIC2016 Datensatzes. Die Fläche unter den Kurven entspricht der AveP-Metrik. Die AveP-Werte sind 52,5 % für das *twoLayer*-Modell, 47,8 % für das *simple*-Modell und 24,7 % für den *random*-Klassifikator.

random 0.50.

Am Datensatz MClass erreicht *twoLayer* eine aveP von 0.64, *simple* 0.63 und *random* 0.26. Am selben Datensatz erreicht *twoLayer* eine AUC von 0.84, *simple* 0.84 und *random* 0.58.

In Abbildung 3.1 sind die PRC Kurven von *twoLayer*, *simple* und *random* am Datensatz ISIC2016 abgebildet. In Abbildung 3.2 sind die ROC-Kurven von *twoLayer*, *simple* und *random* am Datensatz ISIC2016 abgebildet.

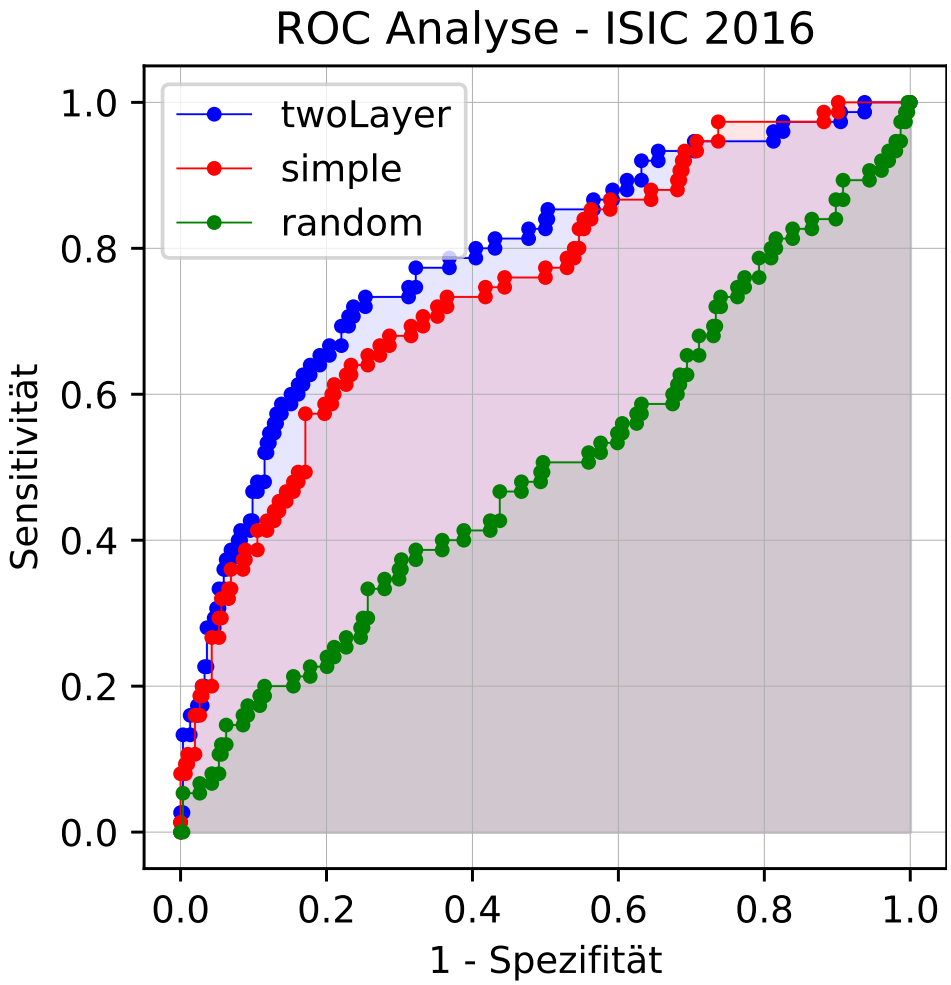

Abbildung 3.2.: ROC-Analyse des ISIC2016-Datensatzes. Die Fläche unter den Kurven entspricht der AUC-Metrik. Die AUC-Werte sind 78,5 % für das *twoLayer*-Modell, 74,8 % für das *simple*-Modell und 50,1 % für den *random*-Klassifikator.

In Abbildung 3.3 sind die PRC Kurven von *twoLayer*, *simple* und *random* am Datensatz MClass abgebildet. In Abbildung 3.4 sind die ROC-Kurven von *twoLayer*, *simple* und *random* am Datensatz MClass abgebildet.

In Abbildung 3.5 sind die Scores von *twoLayer* für maligne und benigne Bilder des MClass Datensatzes als Boxplot dargestellt.

In Abbildung 3.6 sind zwanzig Bilder des MClass Datensatzes abgebildet. Bei den Bildern in den ersten beiden Zeilen handelt es sich um Melanome. In der ersten Zeile

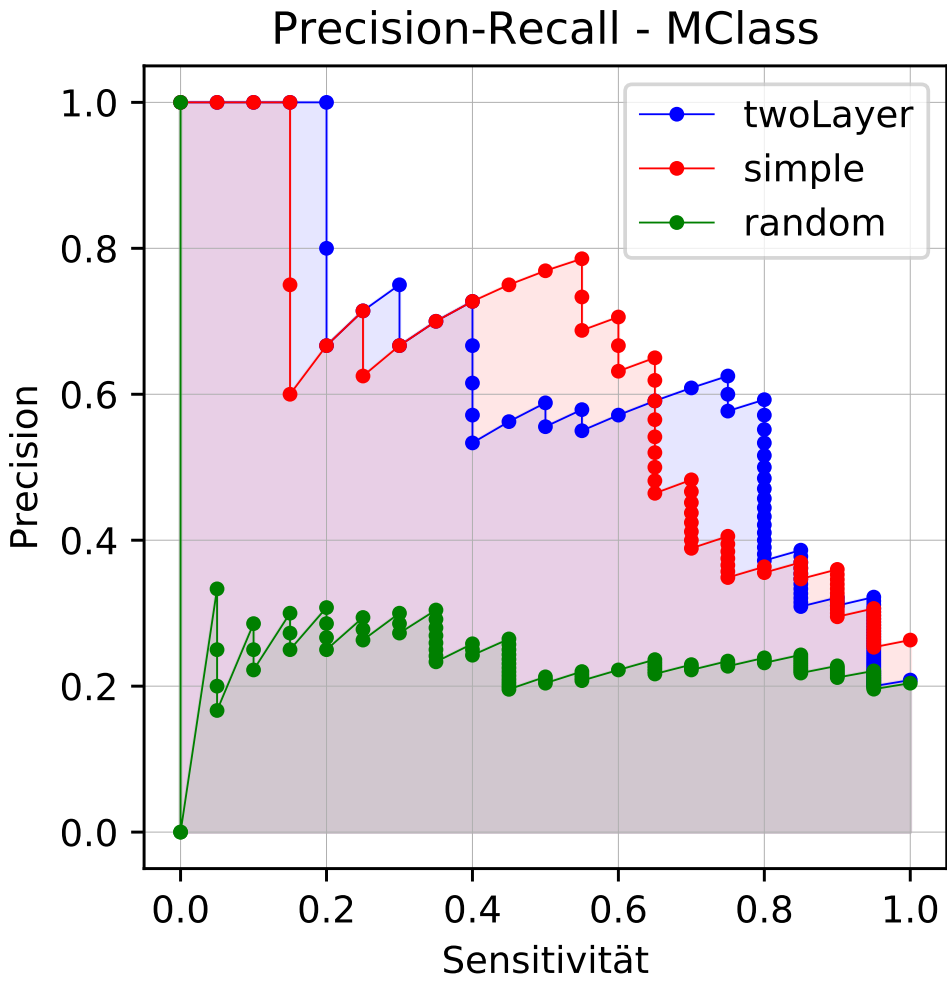

Abbildung 3.3.: PRC-Diagramm des MClass-Datensatzes. Die Fläche unter den Kurven entspricht der AveP-Metrik. Die AveP-Werte sind 64,2 % für das *twoLayer*-Modell, 63,4 % für das *simple*-Modell und 25,7 % für den *random*-Klassifikator.

sind die fünf Melanom-Bilder, die das Modell mit den höchsten Scores bewertet hat, also am ehesten als maligne einstuft. In der zweiten Zeile sind die fünf Melanom-Bilder, die das Modell mit den niedrigsten Scores bewertet hat, also am ehesten als benigne einstuft. Bei den Bildern in den letzten beiden Zeilen handelt es sich um Naevi. In der dritten Zeile sind die fünf Naevi-Bilder, die das Modell mit den niedrigsten Scores bewertet hat, also am ehesten als benigne einstuft. In der vierten Zeile sind die fünf Naevi-Bilder, die das Modell mit den höchsten Scores bewertet hat, also am ehesten als maligne einstuft.

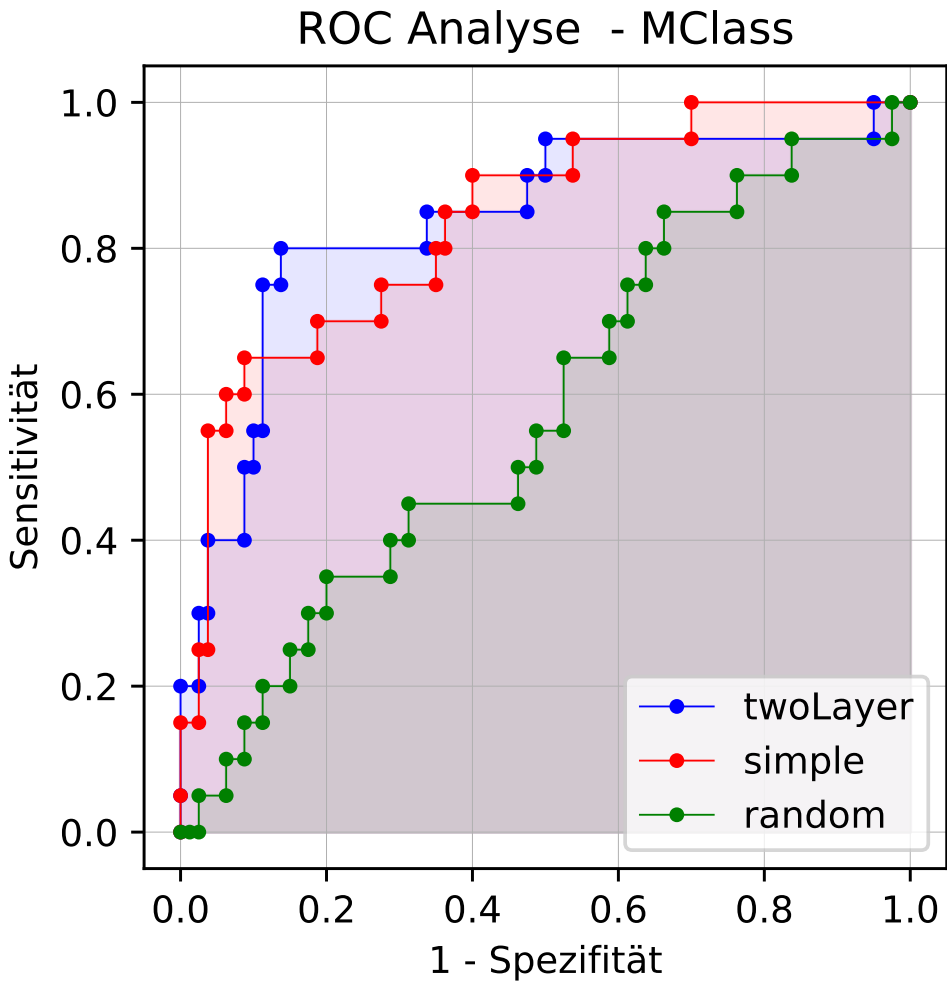

Abbildung 3.4.: ROC-Analyse des MClass Datensatzes. Die Fläche unter den Kurven entspricht der AUC-Metrik. Die AUC-Werte sind 83,8 % für das *twoLayer*-Modell, 83,8 % für das *simple*-Modell und 57,6 % für den *random*-Klassifikator.

3.2. Vergleich mit Dermatologen und Dermatologinnen

In der Tabelle 3.2 sind die Ergebnisse des Vergleiches zwischen den Dermatologinnen und Dermatologen aus Brinker, Hekler, Hauschild u. a., 2019 und *twoLayer* am MClass Datensatz anhand der PRC-Kurve und ROC-Kurve dargestellt.

118 Dermatologen und Dermatologinnen liegen unter der PRC-Kurve von *twoLayer*, 27

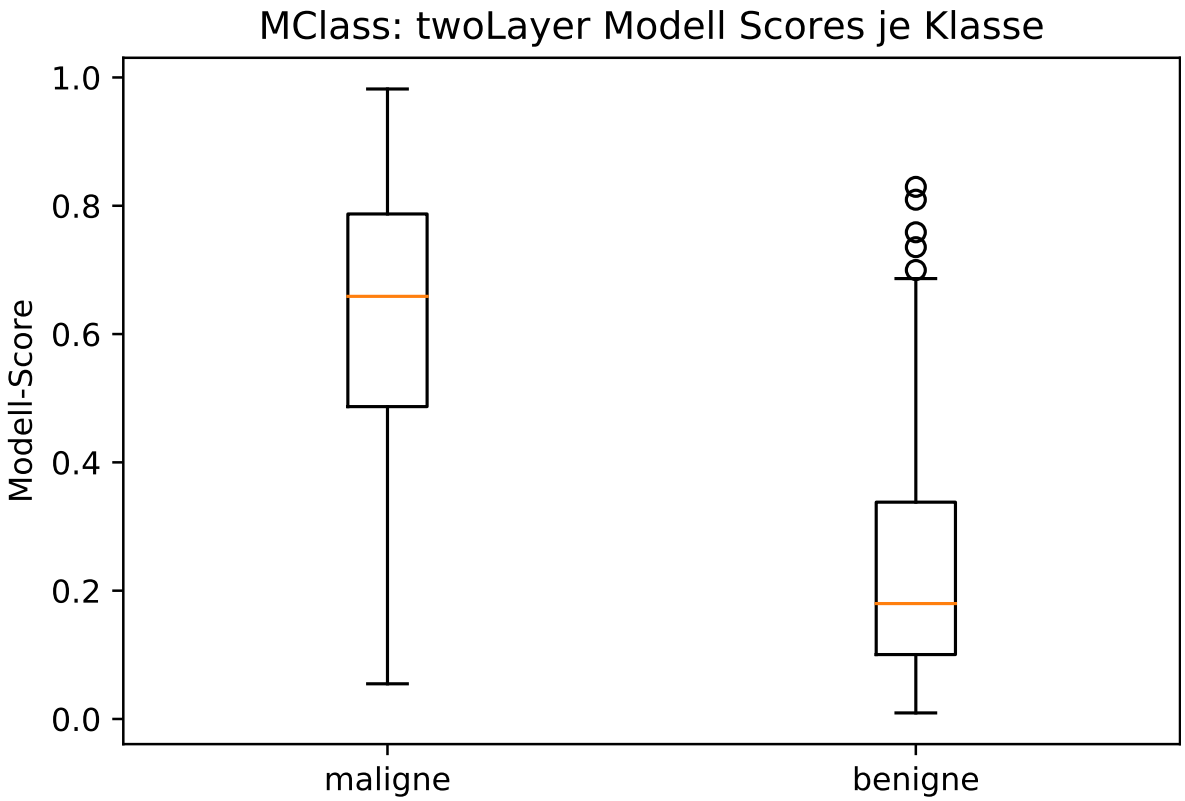

Abbildung 3.5.: In dieser Abbildung sind die errechneten Scores des *twoLayer*-Modells für die Bilder im MClass Datensatz je Kategorie (benigne, maligne) dargestellt. Die Scores sind im Intervall $[0, 1]$. Erst nach Festlegung eines Schwellenwertes T können die Bilder als maligne oder benigne klassifiziert werden. Die Bilder mit Score $s > T$ werden als maligne klassifiziert. Die Kreise im rechten Boxplot stellen Ausreißer dar. In der verwendeten Grafik-Bibliothek *matplotlib* werden Ausreißer nach einer voreingestellten Regel definiert.

auf, und 12 darüber. 118 Dermatologen und Dermatologinnen liegen unter der ROC-Kurve von *twoLayer*, 27 auf, und 12 darüber.

In den Abbildungen 3.7 und 3.8 sind die PRC- und ROC-Kurven von *twoLayer* zusammen mit den Ergebnissen der Dermatologinnen und Dermatologen dargestellt.

Tabelle 3.2.: PRC und ROC Vergleich zwischen *twoLayer* und Dermatologen und Dermatologinnen am MClass Datensatz. Die Tabelle stellt die Anzahl der Dermatologen und Dermatologinnen dar, die unter, auf und über den PRC- und ROC-Kurven von *twoLayer* liegen.

	PRC	ROC
unter	118	118
auf	27	27
über	12	12

3.3. Veröffentlichung

Der Quellcode und die Modelle sind auf der GitHub (GitHub Inc., 2019) unter dem folgenden Link `https://github.com/michaelsiemmeister/melanomaclassification` veröffentlicht. Die Modelle können mithilfe des `release` Buttons heruntergeladen werden.

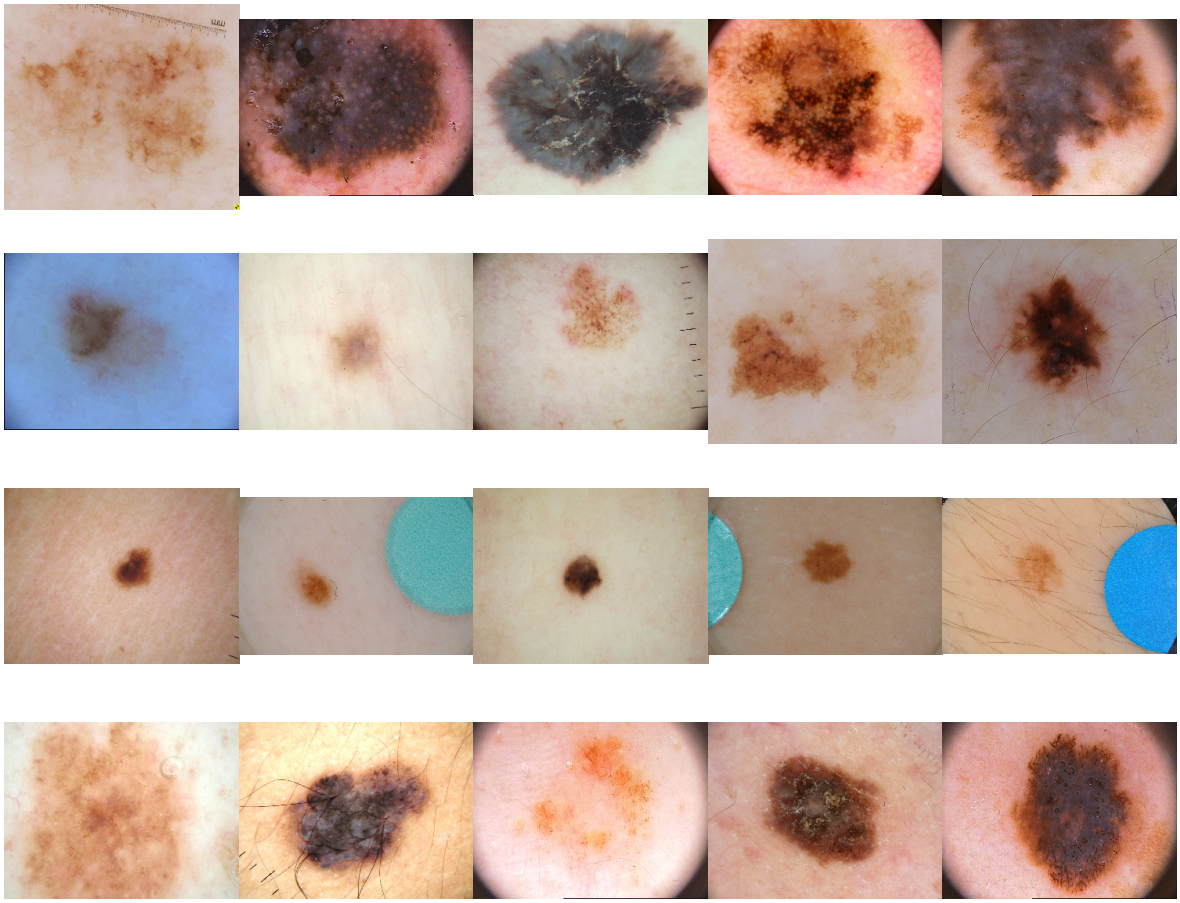

Abbildung 3.6.: Auswertung des *twoLayer*-Modells anhand des MClass Datensatzes. Bei den Bildern in den ersten beiden Zeilen handelt es sich um Melanome. In der ersten Zeile sind die fünf Melanom-Bilder, die das Modell mit den höchsten Scores bewertet hat, also am ehesten als maligne einstuft. In der zweiten Zeile sind die fünf Melanom-Bilder, die das Modell mit den niedrigsten Scores bewertet hat, also am ehesten als benigne einstuft. Bei den Bildern in den letzten beiden Zeilen handelt es sich um Naevi. In der dritten Zeile sind die fünf Naevi-Bilder, die das Modell mit den niedrigsten Scores bewertet hat, also am ehesten als benigne einstuft. In der vierten Zeile sind die fünf Naevi-Bilder, die das Modell mit den höchsten Scores bewertet hat, also am ehesten als maligne einstuft.

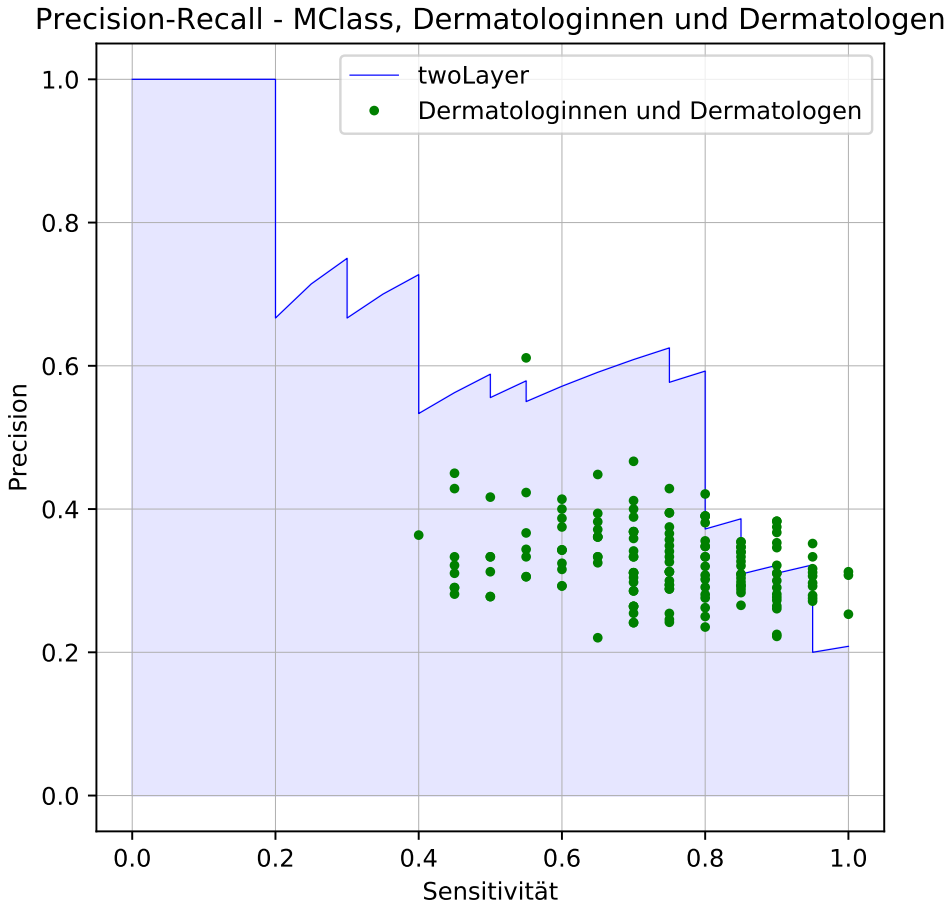

Abbildung 3.7.: PRC-Diagramm des *twoLayer*-Modells im Vergleich mit Dermatologen und Dermatologinnen anhand des MClass-Datensatzes. Die grünen Punkte stellen die Ergebnisse der einzelnen Dermatologen und Dermatologinnen dar.

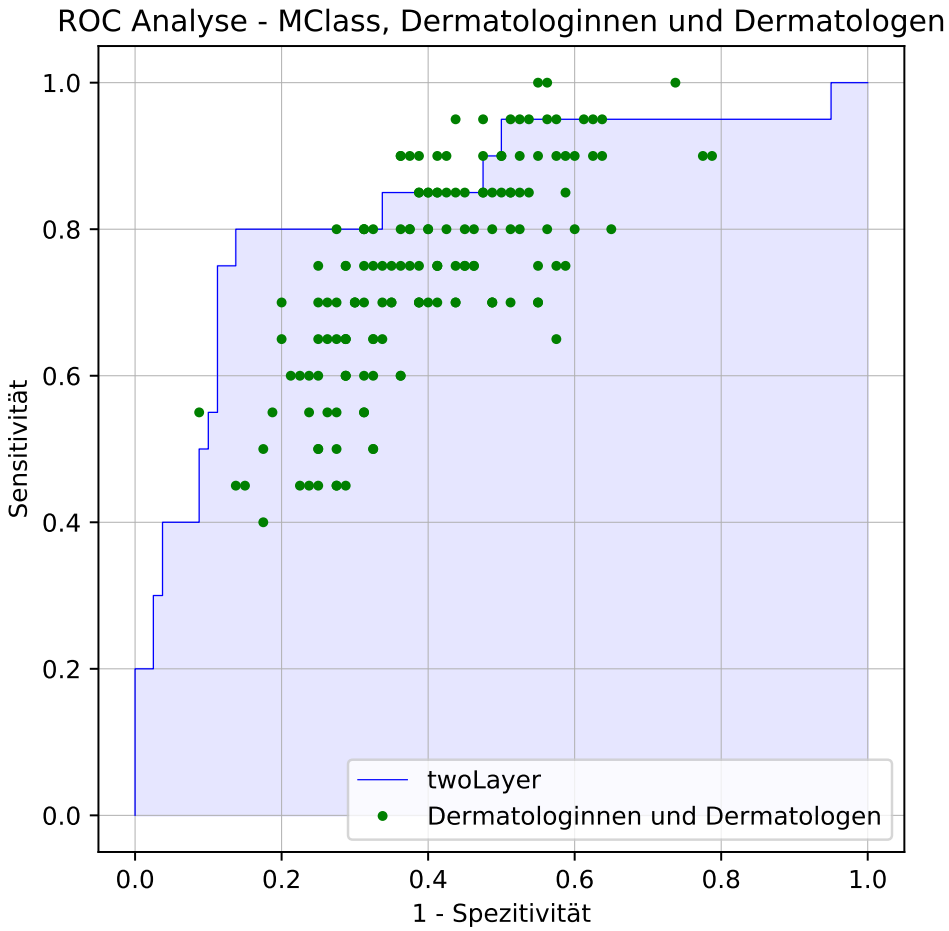

Abbildung 3.8.: ROC-Kurve des *twoLayer*-Modells im Vergleich mit Dermatologen und Dermatologinnen anhand des MClass Datensatzes. Die grünen Punkte stellen die Ergebnisse der einzelnen Dermatologen und Dermatologinnen dar.

4. Diskussion

Die Ziele dieser Arbeit sind mit der Implementierung, dem Training, der Auswertung mit öffentlichen Bildern, dem Vergleich mit Dermatologinnen und Dermatologen sowie der Veröffentlichung erreicht.

Die Modelle *twoLayer* und *simple* werden mit den Bildern der Datensätze ISIC2016 und MClass ausgewertet und mit dem Zufallsklassifikator *random* verglichen. *twoLayer* erreicht im Vergleich zu den Modellen, die an der ISIC Challenge 2016 Part 3 (Gutman u. a., 2016) teilgenommen haben, den elften Platz (International Society for Digital Imaging of the Skin, 2019b). Dies bezieht sich auf die Metrik aveP, da nach dieser Metrik die teilnehmenden Modelle gereiht sind. Der Gewinner dieses Part 3 Lequan Yu (Yu u. a., 2016) erreicht eine aveP von 0.637 und eine AUC von 0.804. twoLayer erreicht mit einer AUC von 0.785 ähnliche Werte, in der Metrik aveP ist der Abstand in Prozentpunkten jedoch größer. Mögliche Erklärungen sind eine unterschiedliche Modell-Architektur, ein anderes Trainings-Verfahren sowie andere Trainings-Bilder. Eine genauere Erklärung ist hier leider nicht möglich.

In Brinker, Hekler, A. H. Enk u. a., 2019 ist ein CNN-Modell am MClass-Datensatz evaluiert worden. Es liegen jedoch keine aveP- und AUC-Metriken vor. Stattdessen werden die Ergebnisse der Dermatologinnen und Dermatologen aus Brinker, Hekler, Hauschild u. a., 2019 mit jenen des Modells mit den Metriken *mean specificity* und *mean sensitivity* verglichen. Es ist nicht bekannt, ob diese mit der aveP und AUC verglichen werden können.

In Haenssle u. a., 2018 ist ein CNN mit 58 Dermatologen und Dermatologinnen verglichen und auch am ISIC2016 Datensatz getestet worden. Die AUC dieses Modells am ISIC2016 beträgt 0.7868. In diesem Paper wurden die 58 Dermatologen und Dermatologinnen und das CNN an zwei eigenen Datensätzen der Universität Heidelberg verglichen. Diese Datensätze bestehen zu 20% aus malignen und zu 80% aus benignen Bildern. An diesen Datensätzen erreicht das CNN eine AUC von 0.864 und 0.953. Es konnte leider nicht erschlossen werden, warum das CNN von Haenssle u. a., 2018 an diesen Datensätzen eine höhere AUC aufweist.

In den Abbildungen 3.1 und 3.3 mag die Linie des Zufallklassifikators *random* merkwürdig erscheinen, da sie vom Punkt $(0, 1)$ auf $(0, 0)$ fällt. Der Punkt $(0, 1)$ wird von scikit-learn (Pedregosa u. a., 2011) festgelegt. Der Punkt $(0, 0)$ lässt sich dadurch erklären, dass bei *random* ein benignes Bild den höchsten Score aufweist. Die PRC-Kurve kann auch wie folgt interpretiert werden: Das Modell, oder in diesem Fall *random*, ordnet jedem Bild einen Score zu. Die Bilder werden nun nach aufsteigenden Scores in einer Reihe sortiert und anschließend wird eine Schwelle zwischen zwei Bilder gesetzt und die Precision und Sensitivität an dieser Schwelle berechnet. Diese Prozedur wiederholt sich für alle Schwellen zwischen den Bildern. Für den Fall, dass die Schwelle links vom ersten Bild gesetzt wird, ist die Sensitivität 1 und die Precision entspricht dem Anteil maligner Bilder. Wenn die Schwelle rechts vom Bild mit dem höchsten Score gesetzt wird ist die Sensitivität 0 und die Precision ist undefiniert, da 0 nicht durch 0 geteilt werden kann. In diesem Fall setzt scikit-learn den Wert der Precision auf 1. Diese Situation wird deutlicher wenn die Bilder von ISIC2016 nach aufsteigenden *random*-Scores geordnet werden. Die Bilder werden durch 1 für maligne und 0 für benigne Bilder symbolisiert. Die Darstellung der letzten 10 Bilder dieser Liste ist `[0, 0, 0, 0, 0, 1, 1, 1, 1, 0]`. So wird ersichtlich, dass *random* ein benignes Bild mit dem höchsten Score versieht. Wird die Schwelle zwischen das letzte und vorletzte Bild (`[000001111|0]`) gesetzt, so wird deutlich, dass die Precision und Sensitivität bei dieser Schwelle 0 sind.

Eine Limitation dieser Arbeit manifestiert sich darin, dass die Modelle nur an Bildern des ISIC-Archives (International Society for Digital Imaging of the Skin, 2019a) getestet wurden. Es können keine Aussagen darüber getroffen werden, ob dies auch auf andere

Bilder anwendbar ist.

Eine weitere Limitation besteht darin, dass die Läsionen nicht über einen Zeitraum hinweg betrachtet werden. Somit können sie nicht auf eine etwaige Größenzunahme kontrolliert werden. Dermatologen und Dermatologinnen können jedoch die Patienten und Patientinnen bei einem Kontrolltermin erneut untersuchen und Läsionen auf eine zeitliche Veränderung hin untersuchen.

Die Bilder werden auf 224 mal 224 Pixel skaliert. Dadurch geht Bildinformation verloren; es ist nicht klar, ob die Klassifikation mit voller Auflösung eine höhere aveP- oder AUC-Metrik haben würde.

Die Größeninformation der Naevi und Melanome steht dem Modell nicht zur Verfügung – es ist nicht klar, ob dies einen Einfluss auf die Metriken haben würde.

Das Modell ist nur mit Bildern trainiert, die mithilfe eines Dermoskops aufgenommen worden sind. Somit ist nicht klar, welche Ergebnisse das Modell bei Fotos von melanozytären Läsionen erreicht.

Eine weitere, bedeutende Limitation neuronaler Netzwerke darf nicht vergessen werden. Neuronale Netzwerke benötigen tausende Trainingsbilder um Melanome von Naevi unterscheiden zu können. Aus eigener Erfahrung mit fachfremden Menschen kann behauptet werden, dass Menschen nach verbaler Information über Asymmetrie, Farbe und Begrenzung in einfachen Fällen Melanome von Naevi unterscheiden können. Das funktioniert sogar, wenn diese Menschen noch nie zuvor Melanome oder Naevi bewusst gesehen haben. Einem neuronalen Netzwerk kann bislang noch nicht verbal erklärt werden, wie sich die Läsionen unterscheiden; es kann nur am Beispiel lernen.

Die Abbildung 3.6 wirft die Frage auf, wo die theoretische Grenze der visuellen Bildklassifizierung liegt. Ist es denn überhaupt möglich alle Läsionen durch rein visuelle Analyse zu klassifizieren? Wie gut können neuronale Netzwerke Melanome und Naevi unterscheiden, wenn sie ein Vielfaches der Trainingsbilder und Rechenleistung zur Ver-

fügung haben? Leider können diese Fragen im Rahmen dieser Arbeit nicht beantwortet werden.

Es soll betont werden, dass neuronale Netzwerke kein Allheilmittel sind. Sie sind lediglich ein Mittel zum Zweck und sollen keinesfalls Forschung an anderen Methoden einschränken. Gäbe es zum Beispiel *billigste Genanalysen aus Stanzbiopsien*, mit denen Melanome zuverlässig von Naevi unterschieden werden können, so wären neuronale Netzwerke auf diesem Gebiet überflüssig.

Der Neuheitswert dieser Arbeit ist, dass die Modelle und der Quellcode veröffentlicht sind. Zum einen kann mit diesen Modellen, welche an zwei öffentlich zugänglichen Datensätzen getestet worden sind, Melanom-Naevus-Klassifizierung durchgeführt werden. Zum anderen können mit diesem Quellcode weitere Modelle trainiert oder ausgewertet werden, ohne dass der Code zum Herunterladen der Bilder oder zum Auswerten selbst geschrieben werden muss. Ich hoffe, dass diese Arbeit zukünftigen Arbeiten einen deutlichen Zeitvorteil einräumen kann.

Literatur

Abadi, Martin, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vinegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu und Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `https://www.tensorflow.org/`.

Bellman, Richard u. a. (1954). „The theory of dynamic programming“. In: *Bulletin of the American Mathematical Society* 60.6, S. 503–515.

Binder, Michael, Margot Schwarz, Alexander Winkler, Andreas Steiner, Alexandra Kaider, Klaus Wolff und Hubert Pehamberger (1995). „Epiluminescence microscopy: a useful tool for the diagnosis of pigmented skin lesions for formally trained dermatologists“. In: *Archives of dermatology* 131.3, S. 286–291.

Brinker, Titus J, Achim Hekler, Alexander H Enk, Joachim Klode, Axel Hauschild, Carola Berking, Bastian Schilling, Sebastian Haferkamp, Dirk Schadendorf, Tim Holland-Letz u. a. (2019). „Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task“. In: *European Journal of Cancer* 113, S. 47–54.

Brinker, Titus J, Achim Hekler, Axel Hauschild, Carola Berking, Bastian Schilling, Alexander H Enk, Sebastian Haferkamp, Ante Karoglan, Christof von Kalle, Michael Weichenthal u. a. (2019). „Comparing artificial intelligence algorithms to 157 German dermatologists: the melanoma classification benchmark“. In: *European Journal of Cancer* 111, S. 30–37.

Centers for Disease Control and Prevention (2019). *Melanoma Incidence and Mortality, United States–2012–2016. USCS Data Brief, no. 9.* URL: `https://www.cdc.gov/cancer/uscs/about/data-briefs/no9-melanoma-incidence-mortality-UnitedStates-2012-2016.htm` (besucht am 17. 12. 2019).

Chollet, François u. a. (2015). *Keras*. `https://keras.io`.

Cireşan, Dan, Ueli Meier und Jürgen Schmidhuber (2012). „Multi-column deep neural networks for image classification“. In: *arXiv preprint arXiv:1202.2745*.

Codella, Noel, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti u. a. (2019). „Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic)“. In: *arXiv preprint arXiv:1902.03368*.

Domingues, Beatriz, José Manuel Lopes, Paula Soares und Helena Pópulo (2018). „Melanoma treatment in review“. In: *ImmunoTargets and therapy* 7, S. 35.

Duchi, John, Elad Hazan und Yoram Singer (2011). „Adaptive subgradient methods for online learning and stochastic optimization“. In: *Journal of Machine Learning Research* 12.Jul, S. 2121–2159.

Esteva, Andre, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau und Sebastian Thrun (2017). „Dermatologist-level classification of skin cancer with deep neural networks“. In: *Nature* 542.7639, S. 115.

GitHub Inc. (2019). *GitHub*. URL: `https://github.com/` (besucht am 16. 12. 2019).

Google LLC (2019). *Google Cloud Platform*. URL: `https://cloud.google.com/` (besucht am 30. 09. 2019).

Gutman, David, Noel CF Codella, Emre Celebi, Brian Helba, Michael Marchetti, Nabin Mishra und Allan Halpern (2016). „Skin lesion analysis toward melanoma

detection: A challenge at the international symposium on biomedical imaging (ISBI) 2016, hosted by the international skin imaging collaboration (ISIC)“. In: *arXiv preprint arXiv:1605.01397*.

Haenssle, Holger A, Christine Fink, R Schneiderbauer, Ferdinand Toberer, Timo Buhl, A Blum, A Kalloo, A Ben Hadj Hassen, L Thomas, A Enk u. a. (2018). „Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists“. In: *Annals of Oncology* 29.8, S. 1836–1842.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren und Jian Sun (2016). „Deep residual learning for image recognition“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 770–778.

Hunter, John D (2007). „Matplotlib: A 2D graphics environment“. In: *Computing in science & engineering* 9.3, S. 90.

International Society for Digital Imaging of the Skin (2019a). *International Skin Imaging Collaboration: Melanoma Project.* URL: `https://www.isic-archive.com` (besucht am 30. 09. 2019).

– (2019b). *ISIC 2016: Skin Lesion Analysis Towards Melanoma Detection Part 3: Lesion Classification.* URL: `https://challenge.kitware.com/#phase/5667455bcad3a56fac786791` (besucht am 16. 12. 2019).

Kiefer, Jack, Jacob Wolfowitz u. a. (1952). „Stochastic estimation of the maximum of a regression function“. In: *The Annals of Mathematical Statistics* 23.3, S. 462–466.

Kittler, Harold, H Pehamberger, K Wolff und M Binder (2002). „Diagnostic accuracy of dermoscopy“. In: *The lancet oncology* 3.3, S. 159–165.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay u. a. (2016). „Jupyter Notebooks-a publishing format for reproducible computational workflows.“ In: *ELPUB*, S. 87–90.

Krizhevsky, Alex, Ilya Sutskever und Geoffrey E Hinton (2012).
„Imagenet classification with deep convolutional neural networks“.
In: *Advances in neural information processing systems*, S. 1097–1105.

Kumar, Vinay, Abul K Abbas, Nelson Fausto und Jon C Aster (2014).
Robbins and Cotran pathologic basis of disease, ninth edition. Elsevier.

LeCun, Yann, Léon Bottou, Yoshua Bengio, Patrick Haffner u. a. (1998).
„Gradient-based learning applied to document recognition“.
In: *Proceedings of the IEEE* 86.11, S. 2278–2324.

McCulloch, Warren S und Walter Pitts (1943).
„A logical calculus of the ideas immanent in nervous activity“.
In: *The bulletin of mathematical biophysics* 5.4, S. 115–133.

McKinney, Wes u. a. (2010). „Data structures for statistical computing in python“.
In: *Proceedings of the 9th Python in Science Conference*. Bd. 445. Austin, TX, S. 51–56.

Menegola, Afonso, Julia Tavares, Michel Fornaciali, Lin Tzy Li, Sandra Avila und Eduardo Valle (2017). „RECOD titans at ISIC challenge 2017“.
In: *arXiv preprint arXiv:1703.04819*.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg u. a. (2011). „Scikit-learn: Machine learning in Python“.
In: *Journal of machine learning research* 12.Oct, S. 2825–2830.

Pratt, Lorien Y (1993). „Discriminability-based transfer between neural networks“.
In: *Advances in neural information processing systems*, S. 204–211.

Rosenblatt, Frank (1961).
Principles of neurodynamics. perceptrons and the theory of brain mechanisms.
Techn. Ber. Cornell Aeronautical Lab Inc Buffalo NY.

Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams u. a. (1988).
„Learning representations by back-propagating errors“.
In: *Cognitive modeling* 5.3, S. 1.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot u. a. (2016).

„Mastering the game of Go with deep neural networks and tree search“. In: *nature* 529.7587, S. 484.

Stewart, Bernhard W., Christopher P. Wild u. a. (2014). *World cancer report 2014*. Lyon: International Agency for Research on Cancer.

Van Rossum, Guido und Fred L Drake Jr (1995). *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam.

Werbos, Paul (1974). „Beyond Regression:"New Tools for Prediction and Analysis in the Behavioral Sciences“. In: *Ph. D. dissertation, Harvard University*.

Yu, Lequan, Hao Chen, Qi Dou, Jing Qin und Pheng-Ann Heng (2016). „Automated melanoma recognition in dermoscopy images via very deep residual networks“. In: *IEEE transactions on medical imaging* 36.4, S. 994–1004.

A. Quellcode

In diesem Anhang wird der Quellcode zur Erstellung der Modelle und deren Auswertung aufgelistet. In der Quellcode-Auflistung A.1 wird die allgemeine Struktur der einzelnen Dateien angegeben. In der Quellcodeauflistung A.2 ist das als Python Script exportierte Jupyter-Notebook zu lesen. Dieses ist aufgrund seiner wesentlich einfacheren Formatierung exportiert worden. In den weiteren Quellcode-Auflistungen folgt der weitere Quellcode.

Quellcode-Auflistung A.1: `Ordnerstruktur`

```
melanomaclassification/
|-- .env.example
|-- .gitignore
|-- example.config.yaml
|-- LICENSE.md
|-- melanoma_classification/
|   |-- __init__.py
|   |-- ISBI_2016/
|   |   --- loadISBI2016metadata.py
|   |-- isicdownload/
|   |   |-- download_images.py
|   |   |-- download_metadatafiles.py
|   |   |-- loadisicmetadata.py
|   |   --- test_download_images.py
|   |-- lib/
|   |   |-- categories.py
|   |   |-- models/
|   |   |   |-- callbacks.py
|   |   |   --- models.py
|   |   --- test_categories.py
|   |-- mclass/
|   |   --- loadmclassmetadata.py
|   |-- melanomaclassification.py
|   --- utils/
|       |-- decorators/
```

```
|       |   |-- decorators.py
|       |   --- test_decorators.py
|       |-- load_config.py
|       |-- logging_configuration.py
|       |-- test_load_config.py
|       --- test_logging_configuration.py
|-- metadata/
|   |-- ISIC_images_metadata.csv
|   --- README.md
|-- model_evaluation.ipynb
|-- Pipfile
|-- Pipfile.lock
|-- README.md
--- test_gpu.ipynb
```

Quellcode-Auflistung A.2: `model_evaluation.py`

```
#!/usr/bin/env python
# coding: utf-8

# # Model Evaluation

# ### Setup

# In[ ]:


import os

from sklearn.metrics import average_precision_score as avp
from sklearn.metrics import precision_recall_curve as prc
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score, recall_score
from sklearn.metrics import precision_score, confusion_matrix
from sklearn.utils import shuffle
import numpy as np
from numpy.random import default_rng
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon

import melanoma_classification.melanomaclassification as mc


# In[ ]:

```

```
33
34 # make sure, the directory for storing results exists.
35 os.makedirs('../results', exist_ok=True)
36
37
38 # In[ ]:
39
40
41 # assign more concise names to the two test-dataframes
42 isic2016 = mc.isic2016test_std_df
43 mclass = mc.mclass_std_df
44
45
46 # In[ ]:
47
48
49 # assert that there are only two labels
50 assert(len(isic2016['category'].unique()) == 2)
51 assert(len(mclass['category'].unique() == 2))
52
53 # data generators
54 isic2016_gen = mc.resnet50_testDataGen_flow_df(
55     isic2016,
56     mc.images_base_path,
57     batch_size=1,
58     class_mode='binary')
59
60 mclass_gen = mc.resnet50_testDataGen_flow_df(
61     mclass,
62     mc.images_base_path,
63     batch_size=1,
64     class_mode='binary')
65
66
67 # ### Load the models
68
69 # In[ ]:
70
71
72 twoLayer = tf.keras.models.load_model(
73     '../models/twoLayer.h5', compile=False)
74 simple = tf.keras.models.load_model(
75     '../models/simple.h5', compile=False)
76
77
78 # In[ ]:
79
80
81 twoLayer.summary()
```

```
82
83
84 # ### Predict using the models and save results
85
86 # The cells in this section don't have to be run if the results already exist.
87
88 # In[ ]:
89
90
91 twoLayer_isic2016_predictions = twoLayer.predict_generator(isic2016_gen)
92 simple_isic2016_predictions = simple.predict_generator(isic2016_gen)
93 # in this case classes returns an array of 0 and 1. 1 for each malignant
94 # image and 0 for each benign image.
95 isic2016['y_true'] = isic2016_gen.classes
96 isic2016['y_score_two_Layer'] = twoLayer_isic2016_predictions
97 isic2016['y_score_simple'] = simple_isic2016_predictions
98 isic2016.to_csv('../results/isic2016_predictions.csv')
99 #isic2016
100
101
102 # In[ ]:
103
104
105 twoLayer_mclass_predictions = twoLayer.predict_generator(mclass_gen)
106 simple_mclass_predictions = simple.predict_generator(mclass_gen)
107 mclass['y_true'] = mclass_gen.classes
108 mclass['y_score_two_Layer'] = twoLayer_mclass_predictions
109 mclass['y_score_simple'] = simple_mclass_predictions
110 mclass.to_csv('../results/mclass_predictions.csv')
111 #mclass
112
113
114 # ## Evaluation
115
116 # ### Load results
117
118 # After executing the first cell, this can be an alternative entry point.
119 # If the results already exist, there is no need to predict again.
120
121 # In[ ]:
122
123
124 isic2016 = pd.read_csv('../results/isic2016_predictions.csv')
125 mclass = pd.read_csv('../results/mclass_predictions.csv')
126
127
128 # In[ ]:
129
130
```

```
# random-classifier-model:
# chooses y_score randomly from a uniform distribution on [0,1].
rg = default_rng(123456789)
isic_random_y_scores = rg.random(isic2016.y_true.values.shape)
mclass_random_y_scores = rg.random(mclass.y_true.values.shape)


# ### Evaluation Function

# In[ ]:


def evaluate_model(y_true, y_score, model_name='NoName'):
    '''
    y_true and y_score shall be 1-D numpy ndarrays
    '''
    results_dict = {}
    results_dict['model_name'] = model_name
    results_dict['average_precision'] = avp(y_true, y_score)
    precision, recall, prc_thresholds = prc(y_true, y_score)
    results_dict['precision'] = np.flip(precision)
    results_dict['recall'] = np.flip(recall)
    results_dict['prc_thresholds'] = np.flip(prc_thresholds)

    results_dict['roc_auc'] = roc_auc_score(y_true, y_score)
    fpr, tpr, roc_thresholds = roc_curve(y_true, y_score)
    results_dict['fpr'] = fpr
    results_dict['tpr'] = tpr
    results_dict['roc_thresholds'] = roc_thresholds

    return results_dict


# In[ ]:


# isic
isic_tL_eval = evaluate_model(isic2016.y_true.values,
                                            isic2016.y_score_two_Layer,
                                            model_name='twoLayer')
isic_tL_eval['color'] = 'b'

isic_simple_eval = evaluate_model(isic2016.y_true.values,
                                            isic2016.y_score_simple,
                                            model_name='simple')

isic_simple_eval['color'] = 'r'
isic_random_eval = evaluate_model(isic2016.y_true.values,
                                  isic_random_y_scores,
```

```
                                    model_name='random')

isic_random_eval['color'] = 'g'


# mclass
mclass_tL_eval = evaluate_model(mclass.y_true.values,
                                                mclass.y_score_two_Layer,
                                                model_name='twoLayer')
mclass_tL_eval['color'] = 'b'

mclass_simple_eval = evaluate_model(mclass.y_true.values,
                                                mclass.y_score_simple,
                                                model_name='simple')

mclass_simple_eval['color'] = 'r'
mclass_random_eval = evaluate_model(mclass.y_true.values,
                                    mclass_random_y_scores,
                                    model_name='random')

mclass_random_eval['color'] = 'g'


# In[ ]:


print('ISIC2016\n-------')
for model in (isic_tL_eval, isic_simple_eval, isic_random_eval):
    print('''Model: {},
  Average Precision = {:.4f},
  ROC-AUC = {:.4f}'''.format(
        model['model_name'], model['average_precision'], model['roc_auc']))
    model['dataset'] = 'isic2016'

print('\n\n')

print('MClass\n------')
for model in (mclass_tL_eval, mclass_simple_eval, mclass_random_eval):
    print('''Model: {},
  Average Precision = {:.4f},
  ROC-AUC = {:.4f}'''.format(
        model['model_name'], model['average_precision'], model['roc_auc']))
    model['dataset'] = 'MClass'


# In[ ]:


eval_df = pd.DataFrame(
```

```
    [isic_tL_eval, isic_simple_eval, isic_random_eval,
     mclass_tL_eval, mclass_simple_eval, mclass_random_eval],
    columns=['model_name', 'dataset', 'average_precision', 'roc_auc'])
eval_df.set_index(['dataset', 'model_name'], inplace=True,
                  verify_integrity=True)
# eval_df


# In[ ]:


d_eval = eval_df.copy()
d_eval.rename(columns={'average_precision': 'AveP', 'roc_auc': 'AUC'},
              inplace=True)
d_eval.index.names = ['Dataset', 'Model']
# d_eval


# In[ ]:


# print LaTeX format
print(d_eval.to_latex(float_format="{:0.4f}".format))


# ### Plots

# In[ ]:


def plot_diagram(fig, ax, model, xkey, ykey,
                 xlabel='', ylabel='', title='', style='.-'):
    ax.plot(model[xkey], model[ykey], style,
            label=model['model_name'], color=model['color'],
           linewidth = 0.5)
    ax.fill_between(model[xkey], model[ykey]
                    , color=model['color'], alpha=0.1)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.set_aspect('equal')

    return fig, ax


# In[ ]:


def plot_precision_recall_curve(models, title='',fig=None,
```

```
                                ax=None, style='.-'):
    if None in (fig, ax):
        fig, ax = plt.subplots()
    for model in models:
        plot_diagram(fig, ax, model, 'recall', 'precision',
                     xlabel='Recall', ylabel='Precision',
                     title=title, style=style)

    ax.grid(linestyle='-', linewidth=0.1)
    ax.legend()
    return fig,ax

def plot_roc_curve(models, title='',fig=None, ax=None, style='.-'):
    if None in (fig, ax):
        fig, ax = plt.subplots()
    for model in models:
        plot_diagram(fig, ax, model, 'fpr', 'tpr',
                     xlabel='FPR', ylabel='TPR',
                     title=title, style=style)

    ax.grid(linestyle='-', linewidth=0.1)
    ax.legend()
    return fig,ax


# In[ ]:


# To uncomment, if you want subplots.
# fig, ax_arr = plt.subplots(nrows=1, ncols=2, figsize=(15,7.5))
# fig, ax_arr[0] = plot_precision_recall_curve(
#     (isic_tL_eval,
#      isic_simple_eval,
#      isic_random_eval),
#     'Precision-Recall-Diagramm - ISIC 2016',
#     fig=fig, ax=ax_arr[0],
# )
# ax_arr[0].set_aspect('equal')
# #ax_arr[0].set_xlim(0,1)
# #ax_arr[0].set_ylim(0,1)

# fig, ax_arr[1] = plot_roc_curve(
#     (isic_tL_eval,
#      isic_simple_eval,
#      isic_random_eval),
#     'ROC Analyse - ISIC 2016',
#     fig=fig, ax=ax_arr[1],
# )
# ax_arr[1].set_aspect('equal')
```

```
327
328
329 # fig.savefig(
330 #     '../results/ISIC2016_subplots.pdf',
331 #     bbox_inches='tight',
332 #     pad_inches=0,
333 # )
334
335
336 # In[ ]:
337
338
339 fig1, ax1 = plot_precision_recall_curve(
340     (isic_tL_eval,
341      isic_simple_eval,
342      isic_random_eval),
343     'Precision-Recall - ISIC 2016',
344 )
345 fig1.savefig(
346     '../results/prc-ISIC2016.pdf',
347     bbox_inches='tight',
348     pad_inches=0,
349 )
350
351
352 # In[ ]:
353
354
355 fig2, ax2 = plot_roc_curve(
356     (isic_tL_eval,
357      isic_simple_eval,
358      isic_random_eval),
359     'ROC Analysis - ISIC 2016')
360 fig2.savefig(
361     '../results/roc-ISIC2016.pdf',
362     bbox_inches='tight',
363     pad_inches=0,
364 )
365
366
367
368 # In[ ]:
369
370
371 fig3, ax3 = plot_precision_recall_curve(
372     (mclass_tL_eval,
373      mclass_simple_eval,
374      mclass_random_eval),
375     'Precision-Recall - MClass')
```

```
fig3.savefig(
    '../results/prc-mclass.pdf',
    bbox_inches='tight',
    pad_inches=0,
)


# In[ ]:


fig4, ax4 = plot_roc_curve(
    (mclass_tL_eval,
     mclass_simple_eval,
     mclass_random_eval),
    'ROC Analysis  - MClass')
fig4.savefig(
    '../results/roc-mclass.pdf',
    bbox_inches='tight',
    pad_inches=0,
)


# ## Comparison to Dermatologists

# In[ ]:


mclass_dermatologist_results = pd.read_csv(
    '../metadata/MClass_ResultsDermoscopic.csv', index_col='Dermatologist')
mclass_dermatologist_results.replace(
    to_replace='biopsy / further treatment', value=1, inplace=True)
mclass_dermatologist_results.replace(
    to_replace='no further treatment', value=0, inplace=True)
mclass_dermatologist_results.rename(
    inplace=True,
    columns={s:i for s,i in zip(
        mclass_dermatologist_results.columns,
        range(0, len(mclass_dermatologist_results.columns)))})
mclass_dermatologist_results = mclass_dermatologist_results.transpose()
mclass_dermatologist_results
mclass_dermatologist_results.index.name = 'id'
mclass_dermatologist_results.rename(
    inplace=True,
    index=dict(mclass['id'])
)
# mclass_dermatologist_results


# In[ ]:
```

```
425
426
427 def mclass_recall(y_pred):
428     return recall_score(mclass['y_true'], y_pred)
429
430 dermatologist_recall = mclass_dermatologist_results.apply(mclass_recall,
431                                                           axis=0)
432
433 def mclass_precision(y_pred):
434     return precision_score(mclass['y_true'], y_pred)
435
436 def mclass_fpr(y_pred):
437     conf_matrix = confusion_matrix(mclass['y_true'], y_pred)
438     tn, fp, fn, tp = confusion_matrix(mclass['y_true'], y_pred).ravel()
439
440     fpr = fp / (fp+tn)
441     return fpr
442
443 dermatologist_recall = mclass_dermatologist_results.apply(
444     mclass_recall, axis=0)
445 dermatologist_precision = mclass_dermatologist_results.apply(
446     mclass_precision, axis=0)
447
448 dermatologist_fpr = mclass_dermatologist_results.apply(
449     mclass_fpr, axis = 0)
450
451 dermatologist_evaluation = pd.DataFrame(data={
452     'recall': dermatologist_recall,
453     'precision': dermatologist_precision,
454     'fpr': dermatologist_fpr})
455 #dermatologist_evaluation
456
457
458 # In[ ]:
459
460
461 fig5, ax5 = plt.subplots(nrows=1, ncols=1, figsize=(8,6))
462 fig5, ax5 = plot_precision_recall_curve(
463     (mclass_tL_eval,),
464     'Precision-Recall - MClass, Dermatologists',style='-',
465     fig=fig5, ax=ax5,
466 )
467 ax5.plot(
468     dermatologist_recall,
469     dermatologist_precision, '.g', label='Dermatologists')
470 ax5.legend()
471 fig5.savefig(
472     '../results/prc-mclass-dermatologists.pdf',
473     bbox_inches='tight',
```

```
    pad_inches=0,
)


# In[ ]:


fig6, ax6 = plt.subplots(nrows=1, ncols=1, figsize=(8,6))

fig6, ax6 = plot_roc_curve(
    (mclass_tL_eval,),
    'ROC Analyse - MClass, Dermatologists',style='-',
    fig=fig6, ax=ax6,
)
ax6.plot(
    dermatologist_fpr,
    dermatologist_recall, '.g', label='Dermatologists')
ax6.legend(loc='lower right')
fig6.savefig(
    '../results/roc-mclass-dermatologists.pdf',
    bbox_inches='tight',
    pad_inches=0,
)


# ### Box Plot

# In[ ]:


tL_probas_mal = mclass['y_score_two_Layer'][mclass['y_true']==1]
tL_probas_ben = mclass['y_score_two_Layer'][mclass['y_true']==0]

fig7, ax7 = plt.subplots()
ax7.boxplot([tL_probas_mal, tL_probas_ben], labels =['malignant', 'benign'],
                #showfliers=False,
                #whis=1.2
)
ax7.set_ylabel('Model-Score')
ax7.set_title('MClass: twoLayer Model scores for each category')
fig7.savefig('../results/boxplot.pdf',
             bbox_inches='tight',
              pad_inches=0,
             )


# ### Image examples

# Show images of the program excelling and failing.
```

```
523
524 # In[ ]:
525
526
527 # mclass
528
529
530 # In[ ]:
531
532
533 mclass[mclass['y_true']==0].sort_values(
534     by=['y_score_two_Layer']).head()#['id.1']
535
536
537 # In[ ]:
538
539
540
541 fig8, ax8 = plt.subplots(nrows = 4, ncols=5,
542                          gridspec_kw={'hspace': 0, 'wspace': 0},
543                          figsize=(5, 4))
544
545 for ax, fname in zip(
546         ax8[0],
547         mclass[mclass['y_true']==1].sort_values(
548             by=['y_score_two_Layer']).tail(5)['id.1']):
549     ax.axis('Off')
550     img = mpimg.imread('../Images/' + fname)
551     imgplot = ax.imshow(img,)
552
553
554
555 for ax, fname in zip(
556         ax8[1],
557         mclass[mclass['y_true']==1].sort_values(
558             by=['y_score_two_Layer']).head(5)['id.1']):
559     ax.axis('Off')
560     img = mpimg.imread('../Images/' + fname)
561     imgplot = ax.imshow(img,)
562
563 for ax, fname in zip(
564         ax8[2],
565         mclass[mclass['y_true']==0].sort_values(
566             by=['y_score_two_Layer']).head(5)['id.1']):
567     ax.axis('Off')
568     img = mpimg.imread('../Images/' + fname)
569     imgplot = ax.imshow(img,)
570
571 for ax, fname in zip(
```

```
        ax8[3],
        mclass[mclass['y_true']==0].sort_values(
            by=['y_score_two_Layer']).tail(5)['id.1']):
    ax.axis('Off')
    img = mpimg.imread('../Images/' + fname)
    imgplot = ax.imshow(img,)


fig8.subplots_adjust(bottom=0, left=0, right=1, top=1)


fig8.savefig('../results/images_rows.pdf', dpi=300,
             bbox_inches='tight',
             pad_inches=0,
            )


# ### Count points inside polygon

# #### precision - recall

# In[ ]:


mclass_tL_precision = mclass_tL_eval['precision']
mclass_tL_recall = mclass_tL_eval['recall']


# In[ ]:


mclass_tL_precision_polygon = np.append(mclass_tL_precision, [0, 0])
mclass_tL_recall_polygon = np.append(mclass_tL_recall, [1, 0])
# mclass_tL_precision_polygon
# mclass_tL_recall_polygon


# In[ ]:


prc_polyg = Polygon([(x,y) for (x,y) in zip(
    mclass_tL_recall_polygon, mclass_tL_precision_polygon)])
# prc_polyg


# In[ ]:


prc_points_list = [Point(x,y) for x, y in zip(
```

```
    dermatologist_recall, dermatologist_precision)]
# prc_points_list[5]


# In[ ]:


prc_inside = pd.Series([prc_polyg.contains(p) for p in prc_points_list])
prc_touch = pd.Series([prc_polyg.touches(p) for p in prc_points_list])
prc_disjoint = pd.Series([prc_polyg.disjoint(p) for p in prc_points_list])


# In[ ]:


# prc_inside.value_counts()


# In[ ]:


# prc_touch.value_counts()


# In[ ]:


# prc_disjoint.value_counts()


# In[ ]:


mclass_tL_ = mclass_tL_eval['precision']
mclass_tL_recall = mclass_tL_eval['recall']


# In[ ]:


mclass_tL_fpr = mclass_tL_eval['fpr']
mclass_tL_tpr = mclass_tL_eval['tpr']
# mclass_tL_fpr
# mclass_tL_tpr


# In[ ]:


```

```
mclass_tL_fpr_polygon = np.append(mclass_tL_fpr, [1])
mclass_tL_tpr_polygon = np.append(mclass_tL_tpr, [0])
# mclass_tL_fpr_polygon
# mclass_tL_tpr_polygon


# In[ ]:


roc_polyg = Polygon([(x,y) for (x,y) in zip(
    mclass_tL_fpr_polygon, mclass_tL_tpr_polygon)])
# roc_polyg


# In[ ]:


roc_points_list = [Point(x,y) for x, y in zip(
    dermatologist_fpr, dermatologist_recall)]
# roc_points_list[5]


# In[ ]:


roc_inside = pd.Series([roc_polyg.contains(p) for p in roc_points_list])
roc_touch = pd.Series([roc_polyg.touches(p) for p in roc_points_list])
roc_disjoint = pd.Series([roc_polyg.disjoint(p) for p in roc_points_list])


# In[ ]:


# roc_inside.value_counts()


# In[ ]:


# roc_touch.value_counts()


# In[ ]:


# roc_disjoint.value_counts()


# In[ ]:
```

```
719
720
721 points_eval_df = pd.DataFrame(
722     {
723         'd_fpr': list(dermatologist_fpr),
724         'd_recall': list(dermatologist_recall),
725         'd_precision': list(dermatologist_precision),
726         'prc_inside': prc_inside,
727         'prc_touch': prc_touch,
728         'prc_disjoint': prc_disjoint,
729         'roc_inside': roc_inside,
730         'roc_touch': roc_touch,
731         'roc_disjoint': roc_disjoint,
732
733     })
734 # points_eval_df
735
736
737 # In[ ]:
738
739
740 roc_points_eval = pd.Series(
741     {key:val.value_counts()[True] for (key,val) in zip(
742         ('below', 'on', 'above'), (roc_inside, roc_touch, roc_disjoint))})
743 prc_points_eval = pd.Series(
744     {key:val.value_counts()[True] for (key,val) in zip(
745         ('below', 'on', 'above'), (prc_inside, prc_touch, prc_disjoint))})
746 points_df = pd.DataFrame({'PRC': prc_points_eval, 'ROC':roc_points_eval})
747
748 # roc_points_eval
749 # prc_points_eval
750 # points_df
751 print(points_df.to_latex())
752
753
754 # ### pictures from training set
755
756 # In[ ]:
757
758
759 ts = mc.training_set
760 # ts
761
762
763 # In[ ]:
764
765
766 ts_malignant = ts[ts['category']=='malignant']
767 malignant_ts = shuffle(ts_malignant, random_state=1).head(5)
```

```
# malignant_ts


# In[ ]:


ts_benign = ts[ts['category']=='benign']
benign_ts = shuffle(ts_benign, random_state=10).head(5)
# benign_ts


# In[ ]:


fig9, ax9 = plt.subplots(nrows = 2, ncols=5,
                         gridspec_kw={'hspace': 0, 'wspace': 0},
                         figsize=(5, 2))

for ax, fname in zip(
        ax9[0],
        malignant_ts['id']):
    ax.axis('Off')
    img = mpimg.imread('../Images/' + fname)
    imgplot = ax.imshow(img,)



for ax, fname in zip(
        ax9[1],
        benign_ts['id']):
    ax.axis('Off')
    img = mpimg.imread('../Images/' + fname)
    imgplot = ax.imshow(img,)



fig9.subplots_adjust(bottom=0, left=0, right=1, top=1)


fig9.savefig('../results/images_training_set.pdf', dpi=300,
             bbox_inches='tight',
             pad_inches=0,
            )


# ### random: Explanation of the Precision-Recall curves
#

# In[ ]:
```

```
817
818
819 isic2016['y_score_random'] = isic_random_y_scores
820 random_sorted = list(isic2016.sort_values(
821     by='y_score_random')['category'].replace(
822     to_replace={'benign':0, 'malignant':1}))
823
824
825 # In[ ]:
826
827
828 random_sorted[-10:]
```

Quellcode-Auflistung A.3: melanoma_classification/__init__.py

```
1 # This is just an __init__ file.
```

Quellcode-Auflistung A.4: melanoma_classification/melanomaclassification.py

```
1  import argparse
2  import logging
3  import os
4
5  import pandas as pd
6  from tensorflow.keras.optimizers import SGD
7  from tensorflow.keras.callbacks import LearningRateScheduler
8
9  from melanoma_classification.utils.logging_configuration import logging_config
10 from melanoma_classification.utils.load_config import (load_config,
11                                                       set_random_seeds)
12
13 from melanoma_classification.isicdownload.loadisicmetadata import load_isic_df
14 from melanoma_classification.isicdownload.download_metadatafiles import (
15     download_all_files)
16 from melanoma_classification.mclass.loadmclassmetadata import load_mclass_df
17 from melanoma_classification.ISBI_2016.loadISBI2016metadata import (
18     load_isbi2016_test_df)
19
20 from melanoma_classification.isicdownload.download_images import download
21 from melanoma_classification.lib.categories import (standardize_columns,
22                                                     random_undersample,
23                                                     create_sets,
24                                                     filter_metadata)
25 from melanoma_classification.lib.models.models import (
26     resnet50_trainDataGen_flow_df, resnet50_testDataGen_flow_df,
27     resnet50_pretrained_simple, resnet50_pretrained_twoLayer)
28 from melanoma_classification.lib.models.callbacks import (
29     modelCheckpoint_callback, CSVLogger_callback, Logger_Callback,
30     Val_Acc_Callback, learning_rate)
```

```
31
32 # configuration
33 if __name__ == '__main__':
34
35     prs = argparse.ArgumentParser(description='Train a neural net')
36     prs.add_argument('--config_file_path',
37                      help='path to config file',
38                      default='example.config.yaml',
39                      required=True)
40     args = prs.parse_args()
41     config_file_path = args.config_file_path
42 else:
43     this_script_path = __file__
44     this_script_dir = os.path.abspath(os.path.split(this_script_path)[0])
45     config_file_path = os.path.realpath(
46         os.path.join(this_script_dir, '../example.config.yaml'))
47
48 # logging
49 logger = logging.getLogger()
50 logger = logging_config(logger)
51
52 # configuration
53 config_dict = load_config(config_file_path)
54
55 # join the metadata paths.
56 config_dict['download_images']['MClass_metadata'] = os.path.join(
57     os.path.abspath(config_dict['metadata_path']),
58     config_dict['download_images']['MClass_metadata_filename'])
59
60 config_dict['download_images']['isbi2016_test_metadata'] = os.path.join(
61     os.path.abspath(config_dict['metadata_path']),
62     config_dict['download_images']['isbi2016_test_metadata_filename'])
63
64 config_dict['download_images']['isic_images_metadata_path'] = os.path.join(
65     os.path.abspath(config_dict['metadata_path']),
66     config_dict['download_images']['isic_images_metadata_filename'])
67
68 set_random_seeds(config_dict, logger)
69
70 logger.info('loading modules, parsing args => Successful')
71
72 # ensure the needed directories exist.
73 # directories needed.
74 needed = {
75     config_dict['download_images']['images_base_path'],
76     config_dict['log_path'],
77     config_dict['models_path'],
78 }  # a set
79 # create directories.
```

```
[os.makedirs(os.path.abspath(d), exist_ok=True) for d in needed]
logger.info('creating directories >> Successful.')

# download metadata files
download_all_files(os.path.abspath(config_dict['metadata_path']))

# download the images
images_base_path = config_dict['download_images']['images_base_path']

isic_metadata_path = (
    config_dict['download_images']['isic_images_metadata_path'])
mclass_metadata_path = (config_dict['download_images']['MClass_metadata'])
isbi2016test_metadata_path = (
    config_dict['download_images']['isbi2016_test_metadata'])

isic_metadata_df = load_isic_df(isic_metadata_path)
mclass_metadata_df = load_mclass_df(mclass_metadata_path)
isbi2016test_metadata_df = load_isbi2016_test_df(isbi2016test_metadata_path)
filtered_isic_metadata_df = filter_metadata(isic_metadata_df)

# union
# of: filtered, mclass and isbi2016
# to download:
# the pipe symbol "|" carries out the union.
download_df = isic_metadata_df.loc[(filtered_isic_metadata_df.index
                                    | mclass_metadata_df.index
                                    | isbi2016test_metadata_df.index)]

df = download(download_df, images_base_path)

# standardize image DataFrames, i.e. select only the relevant columns and
# rename them to 'id' and 'category'.
std_df = standardize_columns(df, config_dict['id_column'],
                             config_dict['category_column'])
# in the ISBI 2016 test set there are two images without clear
# category:
# - 'ISIC_0009959' with category 'indeterminate'
# - 'ISIC_0010454' with category 'indeterminate/malignant'
# in the official ground truth both are counted as malignant
std_df[std_df.isin({'category': ['indeterminate',
                                 'indeterminate/malignant']})] = 'malignant'
# 'ISIC_0011319' has no category. in the official ground truth it is treated as
# benign
std_df.loc[std_df['id'] == 'ISIC_0011319.jpg', 'category'] = 'benign'

# select images which are in filtered but not in mclass or isic2016test
# a set difference operation.
filtered_std_df = std_df.loc[filtered_isic_metadata_df.index.difference(
    mclass_metadata_df.index.union(isbi2016test_metadata_df.index))]
```

```
129
130 # isic 2016 test std
131 isic2016test_std_df = std_df.loc[isbi2016test_metadata_df.index]
132 # mclass std
133 mclass_std_df = std_df.loc[mclass_metadata_df.index]
134
135 # random undersample, such that there the same amount of images in each
136 # category
137 rus_filtered_std_df = random_undersample(filtered_std_df)
138
139 # create training, validation and test set
140 training_set, validation_set, testing_set = create_sets(
141     rus_filtered_std_df, config_dict['validation_size'],
142     config_dict['testing_size'])
143
144 # create DataGenerators
145 training_gen = resnet50_trainDataGen_flow_df(
146     training_set,
147     images_base_path,
148     batch_size=config_dict['batch_size'],
149     class_mode='binary')
150 validation_gen = resnet50_testDataGen_flow_df(
151     validation_set,
152     images_base_path,
153     batch_size=config_dict['batch_size'],
154     class_mode='binary')
155
156 # adapt steps to workflow_testing flag
157 if config_dict['workflow_testing']:
158     steps_per_epoch = 3
159     validation_steps = 3
160 else:
161     steps_per_epoch = len(training_gen)
162     validation_steps = len(validation_gen)
163
164 # models
165 simple_model = resnet50_pretrained_simple()
166 twoLayer_model = resnet50_pretrained_twoLayer()
167
168
169 def train(model):
170     # compile the model
171     model.compile(loss='binary_crossentropy',
172                   optimizer=SGD(lr=1e-03, momentum=0.9),
173                   metrics=['acc'])
174
175     num_non_trainable_epochs = 3
176
177     # train the classification layer
```

```
    history = model.fit_generator(
        training_gen,
        steps_per_epoch=steps_per_epoch,
        epochs=num_non_trainable_epochs,
        validation_data=validation_gen,
        validation_steps=validation_steps,
        verbose=1,
        # initial_epoch=epoch,
        callbacks=[
            modelCheckpoint_callback(config_dict['models_path'],
                                     config_dict['model_name']),
            CSVLogger_callback(config_dict['log_path'],
                               config_dict['log_name']),
            LearningRateScheduler(lambda epoch, lr: 10**(-(epoch + 3))),
            Logger_Callback(os.path.join(config_dict['log_path'], 'mylog.csv'),
                            validation_gen=validation_gen),
        ])

    # unfreeze all layers
    for layer in model.layers:
        layer.trainable = True
    #
    model.compile(loss='binary_crossentropy',
                  optimizer=SGD(lr=1e-5, momentum=0.9),
                  metrics=['acc'])

    # train the whole network
    history = model.fit_generator(
        training_gen,
        steps_per_epoch=steps_per_epoch,
        epochs=100 + num_non_trainable_epochs,
        validation_data=validation_gen,
        validation_steps=validation_steps,
        verbose=1,
        initial_epoch=num_non_trainable_epochs,
        callbacks=[
            modelCheckpoint_callback(config_dict['models_path'],
                                     config_dict['model_name']),
            CSVLogger_callback(config_dict['log_path'],
                               config_dict['log_name']),
            LearningRateScheduler(
                learning_rate(start_epoch=num_non_trainable_epochs,
                              start_learning_rate=1e-5)),
            Logger_Callback(os.path.join(config_dict['log_path'], 'mylog.csv'),
                            validation_gen=validation_gen,
                            some_layers_frozen=False),
        ])

    return model
```

Quellcode-Auflistung A.5: melanoma_classification/mclass/loadmclassmetadata.py

```
1
2  import os
3
4  import pandas as pd
5
6
7  def load_mclass_df(metadata_path):
8      '''
9      I:
10         metadata_path ... str, relative or absolute path to metadata csv file.
11
12     return pandas DataFrame
13       - zeropad the 'File_name' column with 7 zeros and
14         convert it to str and prefix it with ISIC_.
15       - rename the 'File_name' column to 'id'.
16       - set the 'id' column as index.
17     '''
18     metadata_path = os.path.abspath(metadata_path)
19     df = pd.read_csv(metadata_path,
20                      converters={
21                          'File_name': lambda x: 'ISIC_' + str(x).zfill(7)})
22     df.rename(columns={'File_name': 'id'}, inplace=True)
23     df.set_index('id', drop=False, inplace=True)
24     return df
```

Quellcode-Auflistung A.6: melanoma_classification/utils/load_config.py

```
1  import copy
2  import os
3  import random
4
5  import numpy as np
6  import tensorflow
7
8  import yaml
9
10
11 def load_config(path_str):
12     '''
13     I:
14         path_str    str     absolute or relative path for config yaml file
15     O:
16                     dict    dict of dicts or lists containing the configuration
17     '''
18     with open(os.path.abspath(path_str)) as yamlfile:
19         config = yaml.safe_load(yamlfile)
20         return config
21
```

```

def set_random_seeds(config_dict, logger):
    logger.debug('setting random seeds')
    random.seed(a=config_dict['random_seeds']['built_in'])
    np.random.seed(seed=config_dict['random_seeds']['numpy'])
    tensorflow.random.set_seed(config_dict['random_seeds']['tensorflow'])
    logger.debug('set random seeds - DONE')
```

Quellcode-Auflistung A.7: melanoma_classification/utils/test_load_config.py

```

from melanoma_classification.utils.load_config import load_config


def test_load_config():
    conf_dict = load_config('example.config.yaml')
    assert isinstance(conf_dict, dict)
```

Quellcode-Auflistung A.8: melanoma_classification/utils/test_logging_configuration.py

```
import logging

from melanoma_classification.utils import (
    logging_configuration
)


def test_logging(caplog):
    logger = logging.getLogger()
    logger = logging_configuration.logging_config(logger)
    test_message = 'hello, this is a test'
    logger.info(test_message)
    print(caplog)
    messages = [log.message for log in caplog.records]
    assert(test_message in messages)
    # assert('hallo' in messages) # This shall give an error.
```

Quellcode-Auflistung A.9: melanoma_classification/utils/logging_configuration.py

```
import logging
import time


def logging_config(logger: logging.Logger, level: int = 20) -> logging.Logger:
    '''
    configure logging
    '''
    # the line below does not work. the logging objects are singletons.
    # local_logger = copy.deepcopy(logger)
```

```
11
12      logger.setLevel(level)
13
14      # console handler
15      console_handler = logging.StreamHandler()
16
17      # this handler shall respond to all logging levels.
18      # set the logging level in
19      console_handler.setLevel(logging.DEBUG)
20
21      # formatter
22      formatter = logging.Formatter(
23          '%(asctime)s - %(name)s - %(levelname)s - %(message)s')
24      # set time to UTC
25      formatter.converter = time.gmtime
26      # add formatter to handlers
27      console_handler.setFormatter(formatter)
28      # add handlers to logger
29      logger.addHandler(console_handler)
30
31      logger.debug('configured logger, time is logged in UTC')
32      return logger
```

Quellcode-Auflistung A.10: melanoma_classification/utils/decorators/decorators.py

```
1
2  import copy
3
4
5  def decorator_deepcopy_arguments_and_return_value(f):
6      def f_wrapper(*args, **kwargs):
7          # deepcopy the arguments and keyword arguments
8          (copied_args, copied_kwargs) = tuple(
9              map(copy.deepcopy, (args, kwargs)))
10         # call the function
11         return_value = f(*copied_args, **copied_kwargs)
12
13         # deepcopy the return values
14         copied_return_value = copy.deepcopy(return_value)
15         return copied_return_value
16
17     return f_wrapper
```

Quellcode-Auflistung A.11: melanoma_classification/utils/decorators/test_decorators.py

```
1  import copy
2
3  from decorators import decorator_deepcopy_arguments_and_return_value
4
```

```
5
6  def dummy_function(l, kw_l=[]):
7      ret_val = l + kw_l
8      tuple(map(print, (l, kw_l, ret_val)))
9      tuple(map(print, map(id, (l, kw_l, ret_val))))
10     return (l, kw_l, ret_val)
11
12
13 decorated_dummy_function = (
14     decorator_deepcopy_arguments_and_return_value(dummy_function))
15
16
17 def test_deepcopy_decorator():
18     arg_list = [3, 4]
19     kw_list = [5, 6, 7]
20     val1 = dummy_function(arg_list, kw_l=kw_list)
21     print(val1)
22
23     val2 = decorated_dummy_function(arg_list, kw_l=kw_list)
24     print(val2)
25
26     print('\n', val1, val2, sep='\n')
27     assert all(id(x) != id(y) for x, y in zip(val1, val2))
```

Quellcode-Auflistung A.12: melanoma_classification/ISBI_2016/loadISBI2016metadata.py

```
1  import os
2
3  import pandas as pd
4
5
6  def load_isbi2016_test_df(metadata_path):
7      '''
8      I:
9          metadata_path ... str, relative or absolute path to metadata csv file.
10
11     return pandas DataFrame
12     - set the 'name' column as index with name 'id'
13     '''
14     metadata_path = os.path.abspath(metadata_path)
15     df = pd.read_csv(metadata_path,
16                      low_memory=False,
17                      header=None,
18                      names=['id', 'category'])
19     df.set_index('id', drop=False, inplace=True)
20     return df
```

Quellcode-Auflistung A.13: melanoma_classification/lib/categories.py

```python
# -*- coding: utf-8 -*-

'''

This module has the goals:

- BASE_SET -> TRAIN SET, VAL SET, TEST SET


In this module the concepts of categories and stages is used.

An image can belong to a category, e.g. 'benign' or 'malignant'.

An image can be used in a stage like 'training' or 'validation' or 'testing'.

'''
import logging
import copy
import random
import math
import itertools

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler

# configure logging for this module
module_logger = logging.getLogger(__name__)


# set random state
random.seed(a=1)
initial_random_state = random.getstate()


def combine_dataframes(metadata_df, download_df):
    '''
    Specific for my dataset.
    I:
      2 dataframes

    Combines them based on '_id' and 'name' column

    Returns a combined pandas DataFrame
    '''
    return pd.merge(metadata_df, download_df, how='inner', on=['_id', 'name'])


```

```
50 def filter_metadata(metadata_df):
51     '''
52     My specific filter function.
53     Filters the metadata_df.
54     Very specific to this project.
55 
56     I:
57         metadata_df     pandas DataFrame    from ISIC_images_metadata.csv
58 
59     O:
60         filtered_metadata_df    pandas DataFrame    filtered.
61     '''
62     filter_df = (
63         # only images with known dignity, i.e. benign or malignant
64         (metadata_df['meta_clinical_benign_malignant'].isin(
65             ['benign', 'malignant']))
66         # only melanocytic images
67         & (metadata_df['meta_clinical_melanocytic'])
68         # only images confirmed by histology
69         & (metadata_df['meta_clinical_diagnosis_confirm_type'] == (
70             'histopathology'))
71         # only dermoscopic images
72         & (metadata_df['meta_acquisition_image_type'] == 'dermoscopic'))
73 
74     filtered_metadata_df = metadata_df[filter_df]
75     return filtered_metadata_df
76 
77 
78 def standardize_columns(df, id_column, category_column):
79     '''
80     - only selects the id_column and category_column.
81     - renames
82         id_column to 'id'
83         category_column to 'category'
84     I:
85         df ... pandas DataFrame object
86         id_column ... str
87         category_column ... str
88 
89     O:
90         pandas DataFrame object
91     '''
92     select_columns_df = df.loc[:, [id_column, category_column]]
93     output_df = select_columns_df.rename(
94         columns={
95             id_column: 'id',
96             category_column: 'category'})
97     # remove the '.jpg' part
98     output_df['index_col'] = output_df['id'].apply(lambda x: x[:-4])
```

```
    output_df.set_index('index_col', inplace=True)
    return output_df


def _duplicates(*dfs):
    '''
    check if there are elements in more than 1 dataframe
    I:
        dfs... standardized (column 'id') pandas DataFrames
    O:
        bool..True ( There are  duplicates) or False
    '''
    sets = [set(df['id'])
            for df in dfs]
    intersect = set.intersection(*sets)
    return bool(intersect)


def create_sets(standardized_df, validation_size, test_size):
    '''
    I:
        standardized_df pandas DataFrame, with columns: 'id', 'category'.
        validation_size, test_size:
            float - percentage, or int - absolute count
    O:
        Returns a tuple of 3 pandas DataFrames: training, validation, testing

    Partition the data into 3 sets.
    First split off testing_percentage, then split off validation_percentage
    from the rest.
    '''
    process_further, testing = train_test_split(
        standardized_df,
        test_size=test_size,
        random_state=0,
        stratify=standardized_df['category']
    )
    training, validation = train_test_split(
        process_further,
        test_size=validation_size,
        random_state=0,
        stratify=process_further['category']
    )
    [df.reset_index(inplace=True, drop=True)
     for df in [training, validation, testing]]

    if _duplicates(training, validation, testing):
        raise ValueError('There are duplicates')

```

```
148     return (training, validation, testing)
149
150
151 def random_undersample(standardized_df):
152     '''
153     I:
154         standardized_df - pandas DataFrame -> columns: ['id', 'category']
155     O:
156         standardized_df with random_undersampling
157     '''
158     random_undersampler = RandomUnderSampler(random_state=1)
159     x, y = random_undersampler.fit_resample(
160         standardized_df.loc[:, ['id']], standardized_df['category'])
161     return pd.DataFrame({'id': np.ravel(x), 'category': y})
```

Quellcode-Auflistung A.14: melanoma_classification/lib/test_categories.py

```
1  import math
2
3  import pandas as pd
4  from sklearn.model_selection import train_test_split
5
6  import melanoma_classification.lib.categories as categories
7
8
9  def create_test_df(length=30, c1_percentage=0.5):
10     '''
11     creates a test_df
12     I:
13         kwarg:
14             length.... integer.
15             c1_percentage .. float between 0.0 an 1.0
16     O:
17         pandas DataFrame
18     '''
19     num_1 = math.floor(length * c1_percentage)
20     num_2 = length - num_1
21     df = pd.DataFrame(
22         {
23             'id_column_name': list(range(length)),
24             'category_column_name': num_1 * [1] + num_2 * [2],
25             'dummy_column': length
26         })
27     return df
28
29
30 def test_create_test_df():
31     df = create_test_df()
32     assert(isinstance(df, pd.DataFrame))
```

```
    print(df)


def test_standardize_columns():
    df = create_test_df(length=2)
    standardized_df = categories.standardize_columns(
        df,
        'id_column_name',
        'category_column_name'
    )
    assert(list(standardized_df.columns) == ['id', 'category'])
    print(standardized_df)


def test_StratifiedShuffleSplit():
    df = create_test_df()
    standardized_df = categories.standardize_columns(
        df,
        'id_column_name',
        'category_column_name'
    )
    splits = train_test_split(
        standardized_df,
        train_size=0.8,
        random_state=0,
        stratify=standardized_df['category'])

    assert(list(splits[1].groupby('category').size()) == [3, 3])
    assert(list(splits[0].groupby('category').size()) == [12, 12])

    [print(split, split.groupby('category').size()) for split in splits]


def test_create_sets():
    df = create_test_df(length=100)
    standardized_df = categories.standardize_columns(
        df,
        'id_column_name',
        'category_column_name'
    )
    training, validation, testing = categories.create_sets(
        standardized_df, 10, 10)
    print(training)
    print(validation)
    print(testing)
    assert(len(validation) == 10 and len(testing) == 10)


def test_random_undersample():
```

```
    df = create_test_df(length=30, c1_percentage=0.8)
    standardized_df = categories.standardize_columns(
        df,
        'id_column_name',
        'category_column_name'
    )
    rus_df = categories.random_undersample(standardized_df)
    print(rus_df)
    assert(
        list(
            rus_df['id'] == [
                13,
                18,
                3,
                14,
                20,
                17,
                24,
                25,
                26,
                27,
                28,
                29]))


def test_duplicates():
    df1 = pd.DataFrame({'id': [1, 2, 3, 4]})
    df2 = pd.DataFrame({'id': [2, 3]})
    df3 = pd.DataFrame({'id': [5, 6]})

    assert(categories._duplicates(df1, df2))
    assert(not categories._duplicates(df1, df3))
```

Quellcode-Auflistung A.15: melanoma_classification/lib/models/callbacks.py

```
import os
import datetime

import pandas as pd
import numpy as np
from tensorflow.keras.callbacks import (Callback, ModelCheckpoint, CSVLogger)
from sklearn.metrics import accuracy_score


def modelCheckpoint_callback(dir_path, name):
    '''
    I:
      dir_path, where to safe the model .. str
      name, ... model name ... str, e.g. 'model1'
```

```
15
16      Saves the model
17
18      '''
19      filepath = os.path.join(
20          os.path.abspath(dir_path),
21          name) + '_epoch_{epoch:02d}_valloss_{val_loss:.2f}' + '.hdf5'
22
23      return ModelCheckpoint(filepath, )
24
25
26  def CSVLogger_callback(dir_path, name, append=True):
27      filepath = os.path.join(os.path.abspath(dir_path), name) + '.csv'
28      return CSVLogger(filepath, append=append)
29
30
31  def custom_validation_accuracy(validation_gen, model):
32      ground_truth = np.array(validation_gen.classes)
33      # print('\n', ground_truth, '\n')
34      predictions = model.predict_generator(validation_gen)
35      # print('\n', predictions, '\n')
36      predictions = np.ravel(predictions)  # 1-D
37      accuracy = accuracy_score(ground_truth, np.around(predictions))
38      # print('validation accuracy = {}'.format(accuracy))
39      return accuracy
40
41
42  class Logger_Callback(Callback):
43      def __init__(self, filepath, some_layers_frozen=True, validation_gen=None):
44          super(Logger_Callback, self).__init__()
45          self.filepath = filepath
46          self.some_layers_frozen = some_layers_frozen
47          self.validation_gen = validation_gen
48
49      def on_epoch_end(self, epoch, logs=None):
50          # read the pandas DataFrame if it exists:
51          if os.path.exists(self.filepath):
52              df = pd.read_csv(self.filepath)
53          else:
54              df = pd.DataFrame()
55
56          this_dict = {k: [v] for (k, v) in logs.items()}
57          this_dict['epoch'] = [epoch]
58          this_dict['frozen_layers'] = [self.some_layers_frozen]
59          this_dict['custom_val_acc'] = [
60              custom_validation_accuracy(self.validation_gen, self.model)
61          ]
62          this_dict['current_time'] = datetime.datetime.isoformat(
63              datetime.datetime.utcnow())
```

```
64
65          this_epoch_df = pd.DataFrame(this_dict)
66          # print('\n')
67          # print(df)
68          # print(this_epoch_df)
69
70          concated = pd.concat([df, this_epoch_df])
71          # print(concated)
72          concated.to_csv(self.filepath, index=False)
73
74
75  class Val_Acc_Callback(Callback):
76      def __init__(self, validation_gen):
77          super(Val_Acc_Callback, self).__init__()
78          self.validation_gen = validation_gen
79
80      def on_epoch_end(self, epoch, logs=None):
81          # just 1 D numpy array
82          ground_truth = np.array(self.validation_gen.classes)
83          # print('\n', ground_truth, '\n')
84          predictions = self.model.predict_generator(self.validation_gen)
85          # print('\n', predictions, '\n')
86          predictions = np.ravel(predictions)  # 1-D
87          accuracy = accuracy_score(ground_truth, np.around(predictions))
88          # print('validation accuracy = {}'.format(accuracy))
89
90
91  def learning_rate(start_epoch=0,
92                    one_tenth_after_no_epochs=3,
93                    start_learning_rate=1e-5):
94      '''
95      Returns a function which takes an integer - epoch - as input.
96      Output: a float, the new learning rate.
97      '''
98      def lr(epoch):
99          return start_learning_rate * 10**(-(epoch - start_epoch) /
100                                            one_tenth_after_no_epochs)
101
102     return lr
```

Quellcode-Auflistung A.16: melanoma_classification/lib/models/models.py

```
1  from tensorflow.keras.preprocessing.image import ImageDataGenerator
2  from tensorflow.keras.applications.resnet50 import (
3      preprocess_input as resnet50_preprocess, )
4  from tensorflow.keras.applications.resnet50 import ResNet50
5  from tensorflow.keras import (Model, layers)
6
7
```

```
def trainDataGen(preprocessing_fn):
    return ImageDataGenerator(preprocessing_function=preprocessing_fn,
                              rotation_range=359,
                              horizontal_flip=True)


def testDataGen(preprocessing_fn):
    return ImageDataGenerator(preprocessing_function=preprocessing_fn)


def resnet50_trainDataGen_flow_df(df,
                                  directory_path,
                                  batch_size=20,
                                  class_mode='binary'):
    return trainDataGen(resnet50_preprocess).flow_from_dataframe(
        df,
        directory=directory_path,
        x_col='id',
        y_col='category',
        target_size=(224, 224),
        batch_size=batch_size,
        seed=0,
        class_mode='binary')


def resnet50_testDataGen_flow_df(df,
                                 directory_path,
                                 batch_size=20,
                                 class_mode='binary'):
    return testDataGen(resnet50_preprocess).flow_from_dataframe(
        df,
        directory=directory_path,
        x_col='id',
        y_col='category',
        target_size=(224, 224),
        batch_size=batch_size,
        seed=0,
        class_mode='binary',
        shuffle=False)


def resnet50_pretrained_simple():
    '''
    returns a pretrained resnet50 model
    - input shape: (224, 224, 3)
    - global-avg_pooling
    - then 1 fully connected layer with  1 neuron
     = classification layer
    - only the classification layer is trainable
```

```
    '''

    resnet50_model_pretrained = ResNet50(include_top=False,
                                         input_shape=(224, 224, 3),
                                         pooling='avg')
    # set the pretrained layers as not trainable
    for layer in resnet50_model_pretrained.layers:
        layer.trainable = False
    pretrained_output = resnet50_model_pretrained.layers[-1].output
    model_with_classifier = layers.Dense(
        1, activation='sigmoid')(pretrained_output)

    # create keras model
    this_model = Model(resnet50_model_pretrained.input, model_with_classifier)
    return this_model


def resnet50_pretrained_twoLayer():
    '''
    returns a pretrained resnet50 model
    - input shape: (224, 224, 3)
    - global-avg_pooling
    - then 1 fully connected layer with  1024 neurons
    - then 1 fully connected layer with 1 neuron
     = classification layer
    - only the last 2 layers are trainable
    '''

    resnet50_model_pretrained = ResNet50(include_top=False,
                                         input_shape=(224, 224, 3),
                                         pooling='avg')
    # set the pretrained layers as not trainable
    for layer in resnet50_model_pretrained.layers:
        layer.trainable = False
    pretrained_output = resnet50_model_pretrained.layers[-1].output
    fully_connected = layers.Dense(1024, activation='relu')(pretrained_output)
    model_with_classifier = layers.Dense(1,
                                         activation='sigmoid')(fully_connected)

    # create keras model
    this_model = Model(resnet50_model_pretrained.input, model_with_classifier)
    return this_model
```

Quellcode-Auflistung A.17: `melanoma_classification/isicdownload/download_images.py`

```
# -*- coding: utf-8 -*-

import asyncio
import nest_asyncio
```

```
nest_asyncio.apply()
import functools
import logging
import os
import random
import sys
import time
from operator import xor

import aiofiles
import aiohttp
import numpy as np
import pandas as pd
import requests
from aiohttp import ClientSession

import melanoma_classification.lib.categories as categories

# configure logging for this module
module_logger = logging.getLogger(__name__)


def batch_list(input_list, num_elements_per_chunk):
    '''
    '''
    for position in range(0, len(input_list), num_elements_per_chunk):
        # yield the batch
        yield input_list[position:position + num_elements_per_chunk]


async def fetch_image(url, session='None', logger=module_logger):
    try:
        resp = await session.request(method='GET', url=url)
        resp.raise_for_status()
        logger.debug("Got response {} for URL {}".format(resp.status, url))
        img = await resp.read()
        return img
    except Exception as e:
        logger.warning(
            f'downloading: \n\n     {url}\n\nfailed.\n\nError message:\n',
            exc_info=True)
        raise


async def write_one_img_to_file(row,
                                session='None',
                                dst=None,
                                logger=module_logger):
    '''
```

```
    write image to file
    '''

    logger.debug('called write_one_img_to_file')
    if dst is None:
        # just raise any error for now.
        raise ValueError

    dictionary = row.to_dict()
    dictionary['already_exists'] = None
    dictionary['successful_download'] = None
    dictionary['successful_writeFile'] = None
    dictionary['file_size'] = None
    dictionary['download_name'] = dictionary['name'] + '.jpg'

    image_path = os.path.join(dst, dictionary['download_name'])
    url = 'https://isic-archive.com/api/v1/image/{}/download'.format(
        dictionary['_id'])

    if os.path.isfile(image_path):
        if os.path.getsize(image_path) < 10:
            os.remove(image_path)
        else:
            dictionary['already_exists'] = True
    else:
        dictionary['already_exists'] = False
        try:
            img = await fetch_image(url, session)
            logger.debug('fetched an image')
            dictionary['successful_download'] = True
        except Exception as e:
            dictionary['successful_download'] = False

    if dictionary['already_exists']:
        pass
    elif dictionary['successful_download']:
        try:
            async with aiofiles.open(image_path, 'wb') as f:
                await f.write(img)
            dictionary['successful_writeFile'] = True
        except Exception as e:
            dictionary['successful_writeFile'] = False
            if os.path.isfile(image_path):
                os.remove(image_path)

    # get the file size
    if dictionary['already_exists'] or dictionary['successful_writeFile']:
        dictionary['file_size'] = os.path.getsize(image_path)

```

```
103     return dictionary
104
105
106 async def bulk_download_and_write(dst,
107                                   df,
108                                   logger=module_logger,
109                                   max_num_connections=5):
110     '''
111     Download and write concurrently to file multiple images.
112     '''
113     logger.debug('called bulk_download_and_write')
114
115     limited_connector = aiohttp.TCPConnector(limit=max_num_connections)
116     async with ClientSession(connector=limited_connector) as session:
117         tasks = []
118         for index, row in df.iterrows():
119             tasks.append(write_one_img_to_file(row, session=session, dst=dst))
120
121         task_batch_list = list(batch_list(tasks, max_num_connections))
122
123         counter = 0
124         results = []
125         for batch in task_batch_list:
126             batch_result = await asyncio.gather(*batch)
127             results.extend(batch_result)
128             counter += max_num_connections
129             logger.debug(f'processed {counter} images')
130
131     logger.debug(f'processed all {counter} images.')
132     return results
133
134
135 def _download(metadata_df,
136               IMAGES_BASE_PATH,
137               logger=module_logger,
138               all_images=True,
139               num_images=None):
140     '''
141     Downloads the images and returns a pandas DataFrame.
142
143     I:
144         metadata_df     pandas DataFrame    must have '_id' and 'name' column
145         IMAGES_BASE_PATH    str             an absolute path.
146
147
148     returns:
149         return_df... a pandas DataFrame object with the following columns
150             _id
151             name
```

```
            already_exists
            successful_download
            successful_writeFile
            file_size
            success
    '''

    logger.info('downloading images...')
    start = time.time()

    # input checking
    if not xor(bool(all_images), bool(num_images)):
        raise ValueError(
            'specifiy the number of images or download all images')

    if num_images:
        df_download = metadata_df.iloc[0:num_images]
    else:
        df_download = metadata_df

    logger.info(f'There are {len(df_download.index)} images to be downloaded')

    # asyncio.run returns the return value of the given function.
    # in our case, results is a dict.
    results = asyncio.run(
        bulk_download_and_write(IMAGES_BASE_PATH, df_download[['_id',
                                                               'name']]))

    # logger.info(f'{list(iter(results[0]))}')

    # create a DataFrame
    result_df = pd.DataFrame(
        results,
        columns=list(iter(results[0]))
        # columns=['_id', 'name', 'already_exists',
        #          'successful_download', 'successful_writeFile', 'file_size',
        #          'download_name']
    )

    # set a new column in the DataFrame, which inidicates whether the image is
    # downloaded correctly
    result_df['success'] = result_df.apply(
        lambda x: x['file_size'] > 100 and
        (x['successful_writeFile'] or x['already_exists']),
        axis=1)

    # check if processing all the images was successful.
    processing_all_images_successful = result_df['success'].all()

```

```
    if processing_all_images_successful:
        logger.info('processed all images successfully')
    else:
        logger.warn('did not process all images successfully')
        raise Exception('did not progress all images successfully')

    logger.debug('\n' + '\n' + f'{result_df}')
    logger.debug(f'elapsed time: {time.time() - start} seconds')

    return result_df


def _download_postprocessing(filtered_metadata_df,
                             download_df,
                             logger=module_logger):
    # combine the dataframes
    combined_df = categories.combine_dataframes(filtered_metadata_df,
                                                download_df)
    logger.debug('combined dataframe: \n' + '\n' + f'{combined_df}')

    # check for duplicates
    bool_duplicates = combined_df['name'].duplicated().all()
    logger.info('there are duplicates: {}'.format(bool_duplicates))
    if bool_duplicates:
        raise ValueError('There are duplicates in the dataframe')

    return combined_df


def download(metadata_df,
             save_directory_path,
             filter_fn=categories.filter_metadata):
    '''
    Highly specific function for ISIC data and this project.
    I:
        metadata_df    pandas dataframe
        save_directory_path  str     path to directory - where to store the
                                    images; relative or absolute

    O:
        pandas Dataframe with download status and metadata

    '''
    #  abspath
    save_directory_path = os.path.abspath(save_directory_path)

    download_df = _download(metadata_df, save_directory_path)
    combined_df = _download_postprocessing(metadata_df, download_df)
    return combined_df
```

Quellcode-Auflistung A.18: melanoma_classification/isicdownload/test_download_images.py

```
1
2 import pandas as pd
3
4 import melanoma_classification.isicdownload.download_images as download_images
5
6
7 def test_download():
8     metadata_path = 'ISIC_images_metadata.csv'
9     save_dir_path = '../Images'
10    df = download_images.download(metadata_path, save_dir_path)
11    assert(isinstance(df, pd.DataFrame))
```

Quellcode-Auflistung A.19: melanoma_classification/isicdownload/loadisicmetadata.py

```
1
2
3 import os
4
5 import pandas as pd
6
7
8 def load_isic_df(metadata_path):
9     '''
10    I:
11        metadata_path ... str, relative or absolute path to metadata csv file.
12
13    return pandas DataFrame
14    - set the 'name' column as index with name 'id'
15    '''
16    metadata_path = os.path.abspath(metadata_path)
17    df = pd.read_csv(metadata_path, low_memory=False)
18    df.set_index('name', drop=False, inplace=True)
19    df.index.rename('id', inplace=True)
20    return df
```

Quellcode-Auflistung A.20: melanoma_classification/isicdownload/download_metadatafiles.py

```
1 import os
2 from io import StringIO, BytesIO
3 import logging
4 import csv
5
6 import requests
7 import pandas as pd
8
9 log = logging.getLogger(__name__)
10
11 isic_link_test = (
```

```
    'https://challenge.kitware.com/api/v1/item/56faf116cad3a5465d878cfb/download'
)
isic_link_training = (
    'https://challenge.kitware.com/api/v1/file/568e86dacad3a5219e3f45a2/download'
)

mclass_resultsdermoscopic_link = (
    'https://skinclass.de/MClass/ResultsDermoscopic.xlsx')
mclass_dermoscopicNamesource_link = (
    'https://skinclass.de/MClass/DermoscopicNameSource.xlsx')
resultsDermoscopic_name = 'MClass_ResultsDermoscopic.csv'
NameSource_name = 'MClass_DermoscopicNameSource.csv'


def download_isic_ground_truth(link, path):
    try:
        r = requests.get(link)
        if r.status_code == 200:
            with open(path, 'bw') as f:
                f.write(r.content)
        else:
            log.critical('link {} could not be downloaded'.format(link))
    except:
        log.critical('link {} could not be downloaded'.format(link))
    return None


def download_isic_ground_truth_files(path):
    for fn, l in zip(('ISBI2016_ISIC_Part3_Test_GroundTruth.csv',
                      'ISBI2016_ISIC_Part3_Training_GroundTruth.csv'),
                     (isic_link_test, isic_link_training)):

        file_path = os.path.abspath(os.path.join(path, fn))
        # print(file_path)
        # print(l)
        download_isic_ground_truth(l, file_path)


def download_resultsDermoscopic(path):
    try:
        r = requests.get(mclass_resultsdermoscopic_link)
        if r.status_code == 200:
            bIO = BytesIO(r.content)
            df = pd.read_excel(bIO, header=1)
            filepath = os.path.join(os.path.abspath(path),
                                    resultsDermoscopic_name)
            df.to_csv(filepath, index=False, quoting=csv.QUOTE_ALL)
        else:
            raise Exception('wrong statuscode')
```

```
    except:
        log.critical('link {} could not be downloaded'.format(
            mclass_dermoscopicNamesource_link))
    return None


def download_DermoscopicNameSource(path):
    try:
        r = requests.get(mclass_dermoscopicNamesource_link)
        if r.status_code == 200:
            bIO = BytesIO(r.content)
            df = pd.read_excel(bIO, header=0, usecols=[0, 1])
            df.rename(columns={'file name': 'File_name'}, inplace=True)

            filepath = os.path.join(os.path.abspath(path), NameSource_name)
            df.to_csv(filepath, index=False, quoting=csv.QUOTE_ALL)
        else:
            raise Exception('wrong statuscode')
    except:
        log.critical('link {} could not be downloaded'.format(
            mclass_dermoscopicNamesource_link))
    return None


def check_if_metadata_ready(path):
    p = os.path.abspath(path)
    files_set = set(os.listdir(p))
    needed_set = {
        'ISBI2016_ISIC_Part3_Test_GroundTruth.csv',
        'ISBI2016_ISIC_Part3_Training_GroundTruth.csv',
        resultsDermoscopic_name, NameSource_name
    }
    return needed_set <= files_set


def download_all_files(path):
    path = os.path.abspath(path)
    if not check_if_metadata_ready(path):
        download_isic_ground_truth_files(path)
        download_resultsDermoscopic(path)
        download_DermoscopicNameSource(path)
    return True
```