HAW HAMBURG

HAUSARBEIT

Nhut Hoa Huynh

Vergleich von Self-Trained Model und Pre-Trained Model bei der Bildregressionsaufgabe im Anwendungsfall von Pawpularity Score

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Nhut Hoa Huynh

Vergleich von Self- Trained Model und Pre-Trained Model bei der Bildregressionsaufgabe im Anwendungsfall von Pawpularity Score

Hausarbeit eingereicht im Rahmen der Prüfung des Moduls „Machine Learning“
im Masterstudiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Prüfer: Prof. Dr. Peer Stelldinger

Eingereicht am: 12.03.2023

Inhaltsverzeichnis

1 Einleitung

1.1 Ziele dieser Arbeit

Ziele dieser Arbeit sind ein besseres Verständnis der verfügbaren Pre-Trained-Modelle und die Untersuchung der Vor- und Nachteile von Pre- Trained -Modellen im Vergleich zu Self-Trained -Modellen. Durch die Diskussion von Forschungsergebnissen soll geklärt werden, in welchen spezifischen Fällen welche Methode besser geeignet sein soll. Außerdem wird auf mögliche zukünftige Forschungen zu diesem Thema eingegangen.

1.2 Gliederung

Die Arbeit ist in die folgenden Kapitel unterteilt.

In Kapitel 2 werden die grundlegenden Kenntnisse über Self- Trained -Modelle sowie Pre-Trained -Modelle vorgestellt.

Kapitel 3 beschreibt das durchgeführte Experiment.

Kapitel 4 enthält die Ergebnisse des Experiments sowie die Evaluierung der Ergebnisse.

In Kapitel 5 wird das Fazit mit den Vor- und Nachteilen von vortrainierten und selbst erstellten Modellen dargestellt und es werden Empfehlungen für die Auswahl gegeben. Mögliche zukünftige Forschungsarbeiten werden ebenfalls erwähnt.

2 Grundlagen

2.1 Die Konzepte der Self- Trained Modelle

Self-trained Modelle werden in der Regel für konkrete Aufgaben erstellt. Die Architektur solcher Modelle ist sehr individuell. Ein self-trained Modell wird von Grund auf neu trainiert, indem es mit einer großen Menge an Daten trainiert wird, ohne dass das Modell vorher mit anderen Daten vortrainiert wird. Dies kann Stunden, Tage oder sogar Wochen dauern, bis ein Modell in der Lage ist, genaue Vorhersagen zu treffen.

Self-trained Modelle können in solchen Fällen nützlich sein, in denen es keine entsprechenden vortrainierten Modelle gibt oder wenn der Datensatz sehr spezifisch ist und keine grundlegenden Merkmale erfassen kann.

Ein Beispiel für ein self-trained Modell ist ein neuronales Netz, das auf einem bestimmten Datensatz von medizinischen Bildern trainiert wird, um Diagnosen von Krankheiten zu erstellen.

2.2 Die Konzepte der Pre- Trained Modelle

Pre-trained Modelle sind Modelle, die bereits auf großen Datensätzen trainiert wurden und in der Lage sind, grundlegende Merkmale zu erkennen. Diese Modelle können dann für andere Aufgaben verwendet werden, indem sie an die spezifischen Anforderungen der neuen Aufgabe angepasst werden.

Um das Modell an eine bestimmte Aufgabe anzupassen, wird in der Regel eine Technik verwendet, die als Feinabstimmung bezeichnet wird. Dabei werden die Gewichte des Modells an einen kleineren Datensatz angepasst, der speziell für die neue Aufgabe relevant ist. Dies geschieht in der Regel durch das Einfrieren einiger Schichten des Modells und das Hinzufügen anderer Schichten.

Durch die Verwendung von vortrainierten Modellen können Entwickler zwar die Vorlaufzeit und Ressourcen für die Entwicklung von Machine-Learning-Modellen reduzieren, aber die Feinabstimmung eines solchen Modells kann zeitaufwändig sein.

Beispiele für vortrainierte Modelle sind:

- VGGNet; InceptionNet: CNNs, die mit dem ImageNet-Datensatz trainiert wurden und für die Klassifizierung und Segmentierung von Bildern verwendet werden können.
- BERT; OpenAI's GPT-3: Sprachmodelle, die auf einem riesigen Textkorpus vortrainiert wurden und für die Texterstellung genutzt werden können.
- U-Net: Ein CNN, das auf dem ISBI-2012-Datensatz für biomedizinische Bildsegmentierung vortrainiert wurde.

3 Experiment

3.1 Übersicht

Es handelt sich um eine Bildregressionsaufgabe, die die Attraktivität der Fotos von geretteten Haustieren bestimmt. Haustiere mit attraktiven Fotos wecken mehr Interesse und können schneller adoptiert werden.

Das Problem ist ein Wettbewerb auf kaggle.com, der im Jahr 2021 von PetFinder.my organisiert wurde. PetFinder.my ist eine der führenden Tierschutzplattformen Malaysias mit über 180.000 Tieren, von denen 54.000 glücklich adoptiert wurden.

In diesem Wettbewerb werden die Kandidaten die Rohbilder und die Metadaten analysieren, um die „Pawpularity“ von Tierfotos vorherzusagen. Das Modell wird auf den Tausenden von Haustierprofilen von PetFinder.my trainiert und getestet.

3.2 Beschreibung des Datensatzes

Der Datensatz enthält 9912 Bilder von Haustieren. Der Name eines jeden Bildes ist die ID des Haustiers. Es gibt außerdem eine csv-Datei, in der die Haustier-ID und der Pawpularity Score gespeichert sind. Der Pawpularity Score jedes Haustieres wird aus der Statistik der Seitenaufrufe abgeleitet.

In der csv-Datei gibt es ebenfalls handbeschriftete Metadaten für jedes Foto, zum Beispiel: Fokus, Nähe, Augen, Gesicht usw. Diese Informationen werden in dieser Arbeit nicht verwendet, da sich diese Arbeit nur auf die Aufgabe der Bildregression konzentrieren wird.

3.3 Datenvorbereitung und Aufteilung in Trainings- & Testdatensatz

Diese Arbeit konzentriert sich nur auf den Vergleich der Leistung verschiedener Machine-Learning-Modelle, daher wird die Explorative Analyse-Phase hier nicht vorgestellt.

Die Phase der Datenvorbereitung wurde wie folgt durchgeführt. Zuerst wurde die csv-Datei geladen. Die Metadaten wurden entfernt, so dass nur die Haustier-ID und die Pawpularity verbleiben. Dann wurden die Pfade zu den Bildern der Tiere mit Hilfe der IDs in die Tabelle eingefügt.

Dann wurde eine Funktion erstellt, die alle Bilder vorverarbeitet und dann in ein großes Array einfügt. Dieses wird dann in das ML-Modell eingespeist. Diese Vorverarbeitungsfunktion nimmt eine Bild-URL als Eingabe an und gibt einen Eager Tensor in TensorFlow aus. Das Bild wird anschließend auf eine bestimmte Größe von 128 x 128 Pixel angepasst. Es werden die rgb Bilder mit 3 Kanälen verwendet.

Die Ergebnisse der Datenvorverarbeitung wurden für alle an dem Experiment beteiligten Modelle verwendet, um die Modelle so vergleichbar wie möglich zu machen.

Die vorbereiteten Daten wurden anschließend in einen Trainings- und einen Testdatensatz im Verhältnis 80-20 aufgeteilt, d.h. 80% der Daten werden für das Training und die restlichen 20% für das Testen verwendet. Hierfür wird die Funktion train_test_split aus der Sklearn-Bibliothek verwendet.

3.4 Erstellung von einem Self- Trained Modell

Es handelt sich um ein Modell eines neuronalen Faltungsnetzwerks (CNN) mit 7 Faltungsschichten, 2 Dense-Schichten und mehreren Batch-Normalisierungs- und Dropout-Schichten. Das Self-Trained Model nimmt ein 128x128 RGB-Bild mit 3 Kanälen als Eingabe und gibt einen einzelnen Wert als Pawpularity Score aus, der zwischen 1 und 100 liegt.

Die ersten beiden Faltungsschichten haben 16 bzw. 32 Filter mit einer Kernelgröße von 3x3. Die nächste Schicht ist eine Batch-Normalisierungsschicht, die zur Normalisierung der Ausgabe der Faltungsschicht beiträgt. Die vierte Faltungsschicht hat 32 Filter und eine Kernelgröße von 3x3, gefolgt von einer Batch-Normalisierungsschicht und einer Dropout-Schicht, die eine Überanpassung verhindert.

Die nächsten Faltungsschichten bestehen aus 64 bzw. 128 Filtern mit einer Kernelgröße von 3x3. Auf die 6. Faltungsschicht folgt eine Max-Pooling-Schicht, die die räumlichen Dimensionen der Ausgabe um die Hälfte reduziert. Die letzte Faltungsschicht hat 128 Filter und eine Kernelgröße von 3x3, gefolgt von einer weiteren Dropout-Schicht.

Die Ausgabe wird dann geglättet und durch zwei Dense-Schichten mit 512 bzw. 1 Neuron geleitet, wobei die letzte Schicht einen einzigen Wert ausgibt. Das Modell hat insgesamt 4.490.241 Parameter und ist trainierbar.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 61, 61, 16)        2368
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 32)        4640
_________________________________________________________________
batch_normalization (BatchNo (None, 61, 61, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 31, 31, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 31, 31, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 31, 31, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 31, 31, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
```

```
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,490,241
Trainable params: 4,489,345
Non-trainable params: 896
_________________________________________________________________
```

Tabelle 1: Architektur des Self-Trained Modells

3.5 Erstellung von Pre-Trained Modellen

Die Pre- Trained Modelle in dieser Arbeit werden ResNet als Basismodell verwenden. ResNet ist ein tiefes neuronales Netz, das für die Bilderkennung entwickelt wurde. Der Name ResNet steht für „Residual Network“ und bezieht sich auf die Verwendung von Residualeinheiten, die es dem Netz ermöglichen, sehr tief zu sein, ohne an Leistung zu verlieren.

ResNet verwendet eine so genannte „Skip Connection“, um „Sprünge“ zwischen verschiedenen Schichten im Netz zu ermöglichen, so dass das Netz leichter trainiert werden kann und in der Lage ist, komplexe Beziehungen in den Eingabedaten besser zu erfassen. Die Architektur von ResNet umfasst eine große Anzahl von Schichten, die in Blöcken gruppiert sind, die wiederum aus mehreren Residualeinheiten bestehen.

Das erste Pre- Trained Modell verwendet die ResNet50-Architektur als Basismodell und fügt eine neue „Head“ bzw. Top-Schicht hinzu, die eine Bottleneck-Schicht ist. Eine Bottleneck-Schicht ist eine Schicht in einem neuronalen Netz, die die Anzahl der Parameter im Modell

reduziert und gleichzeitig die wichtigen Informationen in der Eingabe beibehält, so dass eine Überanpassung verhindert wird. In diesem konkreten Modell kann die Bottleneck-Schicht einen von drei Typen haben: „flatten“, „avg“, oder „max“.

- Wenn der Typ „flatten“ ist, reduziert die Bottleneck-Schicht die Ausgabe des Basismodells in einen eindimensionalen Vektor, bevor sie an die endgültige Vorhersageschicht weitergegeben wird.
- Wenn der Typ „avg“ ist, wendet die Bottleneck-Schicht ein globales Average-Pooling auf die Ausgabe des Basismodells an, d. h. sie bildet den Durchschnittswert der einzelnen Merkmale in der Ausgabe.
- Wenn der Typ „max“ ist, wendet die Bottleneck-Schicht globales Max-Pooling auf die Ausgabe des Basismodells an, d. h. sie nimmt den maximalen Wert jedes Merkmals in der Ausgabe.

Das ResNet50-Basismodell verfügt über vortrainierte Gewichte aus dem ImageNet-Datensatz, einem großen Bilddatensatz, der üblicherweise zum Trainieren von Deep-Learning-Modellen für Computer-Vision-Aufgaben verwendet wird.

Die Ausgabeschicht ist, wie das Self-Trained Modell, eine einzelne Dense-Schicht mit einer ReLU-Aktivierungsfunktion. Vor der Ausgabeschicht befindet sich eine Dropout-Schicht mit einer Dropout-Rate von 0,5.

Das gesamte Modell verwendet außerdem zwei Callbacks, EarlyStopping und ReduceLROnPlateau, zur Überwachung des root mean squared error der Validierung (val_rmse) während des Trainings. EarlyStopping stoppt das Training, wenn sich der RMSE der Validierung in 10 aufeinanderfolgenden Epochen nicht verbessert. ReduceLROnPlateau reduziert die Lernrate um den Faktor 0,2, wenn sich der Validierungs-RMSE in 3 aufeinanderfolgenden Epochen nicht verbessert. Die minimale Lernrate wird auf 1e-7 festgelegt.

Das Pre- Trained Modell nimmt ebnefalls eine Eingabeform von (128, 128, 3) für RGB-Bilder an.

```
def create_model(input_shape, top='flatten'):
    if top not in ('flatten', 'avg', 'max'):
        raise ValueError('unexpected top layer type: %s' % top)

    # connects base model with new "head"
    BottleneckLayer = {
        'flatten': keras.layers.Flatten(),
        'avg': keras.layers.GlobalAveragePooling2D(),
        'max': keras.layers.GlobalMaxPool2D()
    }[top]

    base  =  tf.keras.applications.resnet50.ResNet50(input_shape=input_shape,  include_top=False,
weights='imagenet')

    x = BottleneckLayer(base.output)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(1, activation='relu')(x)
    model = tf.keras.Model(inputs=base.inputs, outputs=x)
    return model

early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_rmse',
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True
    )
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor="val_rmse",
    factor=0.2,
    patience=3,
    min_lr=1e-7,
    verbose=1,
    mode='min'
    )
callbacks = [early_stop,reduce_lr]

input_shape=(128,128,3)
model=create_model(input_shape)
```

Tabelle 2: Erstellung des Modells mit ResNet50 als Basismodell

Neben das ResNet50, das 50 Schichten hat, gibt es in der in Keras verfügbaren ResNet-Familie auch ResNet 101 und 152. ResNet 101 besteht aus 101 Schichten, während ResNet 152 aus 152 Schichten besteht. Diese beiden Größen werden in dieser Arbeit auch mit der gleichen Architektur („Head“ und „Callbacks“) wie ResNet50 aufgebaut, so dass sie alle vergleichbar sind.

Somit gibt es in dieser Arbeit insgesamt 4 Machine-Learning-Modelle, die untersucht werden sollen. Im Folgenden wird das Training dieser Modelle vorgestellt.

3.6 Trainingsphase

Jedes der vier Modelle wird mit zwei verschiedenen Verlustfunktionen und den folgenden Einstellungen kompiliert.

Die erste verwendete Verlustfunktion ist der Mean Squared Error (MSE). Diese Funktion berechnet den durchschnittlichen quadratischen Fehler zwischen den Vorhersagen des Modells und den tatsächlichen Werten. Die zweite verwendete Verlustfunktion ist der Mean Absolute Error (MAE). Sie misst die durchschnittliche absolute Differenz zwischen den vorhergesagten und den tatsächlichen Werten. Diese 2 Verlustfunktionen werden häufig bei Regressionen verwendet.

Der Optimierer ist auf „Adam“ („adaptive moment estimation“) eingestellt, eine weit verbreitete Variante des stochastischen Gradientenabstiegs. „Adam“ ist ein sehr effektiver Optimierungsalgorithmus für tiefe neuronale Netze.

Die beim Training zu überwachenden Metriken sind Root-Mean-Squared-Error (RMSE) und Mean Absolute Error (MAE).

Bevor die Modelle trainiert werden, wird noch der Prozess von Daten Augmentation durchgeführt. Ziel ist es, die Trainingsdaten künstlich zu vergrößern und so die Generalisierungsfähigkeit des Modells zu verbessern und besser mit Variationen und Verzerrungen in den Eingabedaten umzugehen.

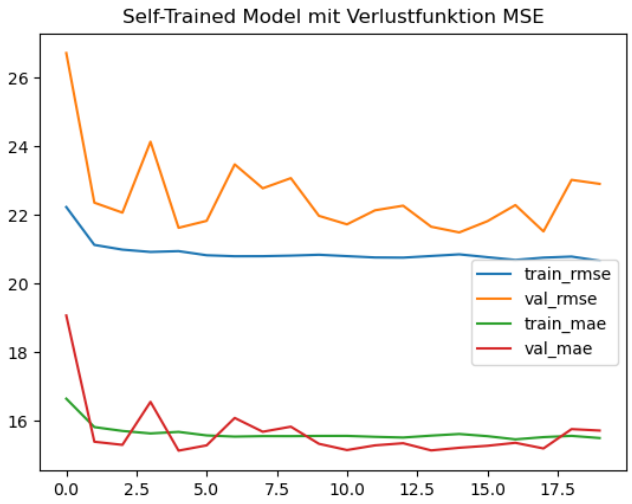
Dieser Prozess wird mit Hilfe von ImageDataGenerator, einem Bilddatengenerator in Keras, durchgeführt. Es werden mehrere folgende Parameter verwendet, die die Arten der an den Bildern vorzunehmenden Erweiterungen definieren.

- rotation_range: ein Bereich in Grad, um das Bild nach dem Zufallsprinzip zu drehen. Hier 15 Grad.
- zoom_range: ein Bereich zum zufälligen Zoomen des Bildes. Hier 0.15, was bedeutet, dass die Eingabebilder bis zu 15% vergrößert oder verkleinert werden können.
- width_shift_range und height_shift_range: ein Bereich zum zufälligen Verschieben des Bildes in horizontaler und vertikaler Richtung. Hier sind beide Werte 0,2, was bedeutet, dass die Eingabebilder zufällig um bis zu 20% der Gesamtbreite und -höhe in jede Richtung verschoben werden können.
- shear_range: ein Bereich, um zufällige Schertransformationen auf das Bild anzuwenden. Hier 0,1, der maximale Winkel für die zufällige Anwendung von Schertransformationen auf die Eingabebilder.
- horizontal_flip: ein boolescher Wert, um das Bild zufällig horizontal zu spiegeln. Hier werden True gesetzt, d. h. einige Bilder werden gespiegelt, wodurch neue Variationen desselben Bildes entstehen.
- fill_mode: die Methode, mit der neu erstellte Pixel gefüllt werden. Hier „nearest“, was bedeutet, dass der Wert des nächstgelegenen Pixels zum Füllen des neuen Pixels verwendet wird.

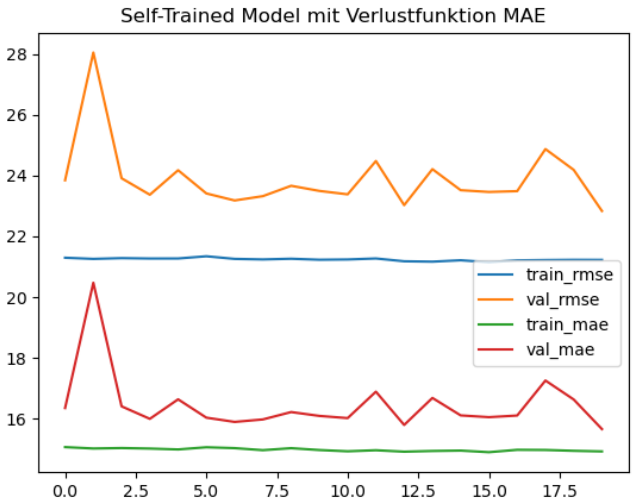
Nachdem die Trainingsdaten augmentiert sind, wird jedes der 4 Modelle mit 20 Epochen trainiert. Die Anzahl der Epochen soll im Fall einer verfügbaren höheren Rechnerleistung erhöht werden. Da es 2 Verlustfunktionen gibt, gibt es insgesamt 8 Möglichkeiten. Diese Ergebnisse werden im Folgenden vorgestellt und evaluiert.

4 Evaluierungsprozess

4.1 Evaluierung des Self- Trained Modells



Das Self- Trained Modell hat im besten Fall beim Training einen RSME von etwa 20,7 und einen MAE von etwa 15,5 erreicht, während es für den Validierungsdatensatz im besten Fall einen RSME von etwa 21,6 und einen MAE von etwa 15,1 erreichte. Obwohl es Schwankungen zwischen den Epochen gibt, ist zu erkennen, dass der RMSE und MAE von Epoche zu Epoche abnehmen, was darauf hindeutet, dass das Modell gelernt hat, besser zu generalisieren.

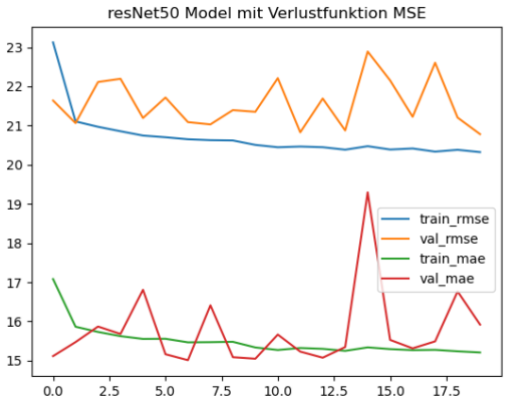


In dem Fall von Verlustfunktion MEA scheint das Training zu einer Überanpassung geführt zu haben, da die RMSE- und MAE-Werte für den Trainingsdatensatz viel niedriger sind als für den Validierungsdatensatz. Dies deutet darauf hin, dass sich das Modell die Trainingsdaten gemerkt hat, anstatt gut auf ungesehene Daten zu generalisieren.

Der beste RMSE-Wert für den Validierungsdatensatz lag bei etwa 23, während der beste MAE-Wert bei etwa 15 lag. Diese Werte sind jedoch nicht aussagekräftig, da das Modell überangepasst ist.

4.2 Evaluierung des Pre- Trained Modells

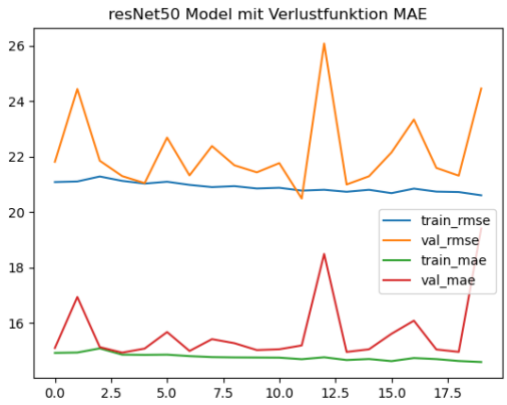
4.2.1 ResNet50



Während des Trainingsprozesses nehmen RMSE und MAE ab. Diese Werte für den Validierungssatz variieren auch hier von etwa 21 bis etwa 23. Allerdings scheinen sich die Validierungsmetriken, insbesondere der RMSE, allmählich zu verbessern. Die beste Leistung wird bei Epoche 20 erreicht, wo der Validierungs-RMSE am niedrigsten ist (20,7813).

Da die Trainingsmetriken und die Bewertungsmetriken nicht sehr unterschiedlich waren, handelt es sich nicht um eine Überanpassung.

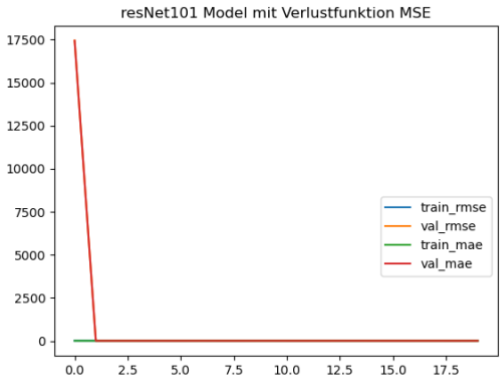
Es ist auch bemerkenswert, dass das Training dieses Modells doppelt so lange dauert wie das Training des selbst trainierten Modells, von etwa 17s pro Epoche auf etwa 45s pro Epoche.



Die Bewertungsmetriken bleiben beim Training nahezu unverändert. Die Leistung des Modells hat sich somit über die 20 Trainingsepochen hinweg nicht wesentlich verbessert.

Die Bewertungsmetriken bei der Validierung scheinen zu schwanken und sich stark von denen beim Training zu unterscheiden. In einigen Epochen, z. B. in der zweiten und dreizehnten, sind die RMSE- und MAE-Werte recht hoch, was auf eine schlechte Leistung hinweist. Die letzte Epoche hat einen Validierungs-RMSE von 24,4596 und einen Validierungs-MAE von 19,4114, was ebenfalls eine der schlechtesten Epochen ist. Hier liegt also eine Überanpassung vor.

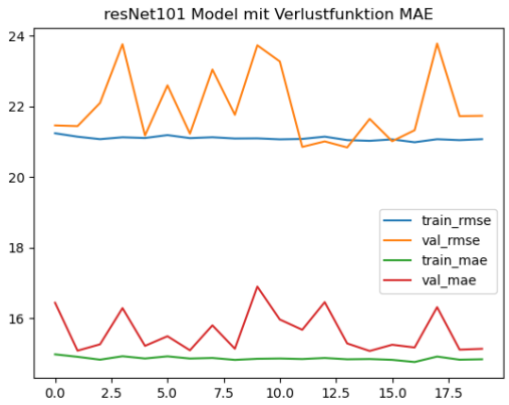
4.2.2 ResNet101



Das Verlustdiagramm kann die Leistung dieses Modells nicht gut darstellen, da die erste Epoche sehr schlecht abschneidet. Das Trainingsprotokoll wird daher wie folgt angezeigt.

```
Epoch 1/20
loss: 513.9606 - rmse: 22.6707 - mae: 16.8870 - val_loss: 304282560.0000 - val_rmse: 17443.6973 - val_mae: 17428.2
422
Epoch 2/20
loss: 454.2473 - rmse: 21.3131 - mae: 15.9821 - val_loss: 474.3205 - val_rmse: 21.7789 - val_mae: 15.1889
Epoch 3/20
loss: 433.9117 - rmse: 20.8305 - mae: 15.5946 - val_loss: 443.4412 - val_rmse: 21.0580 - val_mae: 15.2175
Epoch 4/20
loss: 429.9992 - rmse: 20.7364 - mae: 15.5410 - val_loss: 441.4767 - val_rmse: 21.0113 - val_mae: 15.1016
Epoch 5/20
loss: 425.7884 - rmse: 20.6346 - mae: 15.4465 - val_loss: 469.8192 - val_rmse: 21.6753 - val_mae: 15.1242
Epoch 6/20
loss: 426.0378 - rmse: 20.6407 - mae: 15.4387 - val_loss: 447.8859 - val_rmse: 21.1633 - val_mae: 15.0765
Epoch 7/20
loss: 423.1521 - rmse: 20.5707 - mae: 15.4011 - val_loss: 440.2635 - val_rmse: 20.9825 - val_mae: 15.7173
Epoch 8/20
loss: 421.3500 - rmse: 20.5268 - mae: 15.3627 - val_loss: 462.5995 - val_rmse: 21.5081 - val_mae: 14.9927
Epoch 9/20
loss: 420.5376 - rmse: 20.5070 - mae: 15.3461 - val_loss: 436.8997 - val_rmse: 20.9021 - val_mae: 15.2918
Epoch 10/20
loss: 422.0476 - rmse: 20.5438 - mae: 15.3841 - val_loss: 436.8852 - val_rmse: 20.9018 - val_mae: 16.1120
Epoch 11/20
loss: 420.1197 - rmse: 20.4968 - mae: 15.3749 - val_loss: 441.8183 - val_rmse: 21.0195 - val_mae: 15.2293
Epoch 12/20
loss: 420.0441 - rmse: 20.4950 - mae: 15.3559 - val_loss: 433.1866 - val_rmse: 20.8131 - val_mae: 15.3049
Epoch 13/20
loss: 421.7302 - rmse: 20.5361 - mae: 15.4039 - val_loss: 438.4232 - val_rmse: 20.9386 - val_mae: 15.5946
Epoch 14/20
loss: 423.6747 - rmse: 20.5834 - mae: 15.4074 - val_loss: 438.8059 - val_rmse: 20.9477 - val_mae: 16.4078
Epoch 15/20
loss: 421.9814 - rmse: 20.5422 - mae: 15.3788 - val_loss: 436.7556 - val_rmse: 20.8987 - val_mae: 15.3135
Epoch 16/20
loss: 417.9926 - rmse: 20.4449 - mae: 15.2983 - val_loss: 461.4244 - val_rmse: 21.4808 - val_mae: 15.0961
Epoch 17/20
loss: 419.9928 - rmse: 20.4937 - mae: 15.3023 - val_loss: 439.8286 - val_rmse: 20.9721 - val_mae: 15.2141
Epoch 18/20
loss: 423.3058 - rmse: 20.5744 - mae: 15.4309 - val_loss: 437.7581 - val_rmse: 20.9227 - val_mae: 15.3406
Epoch 19/20
loss: 418.6436 - rmse: 20.4608 - mae: 15.3118 - val_loss: 441.4720 - val_rmse: 21.0112 - val_mae: 15.0752
Epoch 20/20
loss: 417.6513 - rmse: 20.4365 - mae: 15.2985 - val_loss: 446.0232 - val_rmse: 21.1193 - val_mae: 16.2348
```

Anhand des oben dargestellten Trainingsprotokolls lässt sich erkennen, dass die RMSE- und MAE-Werte für den Trainingssatz abnehmen. Die RMSE- und MAE-Werte für die Validierungsdaten bleiben nahezu konstant und sind auch im Vergleich zu den Trainingsdaten relativ gleich. Dies bedeutet, dass das Modell keine Überanpassung aufweist.

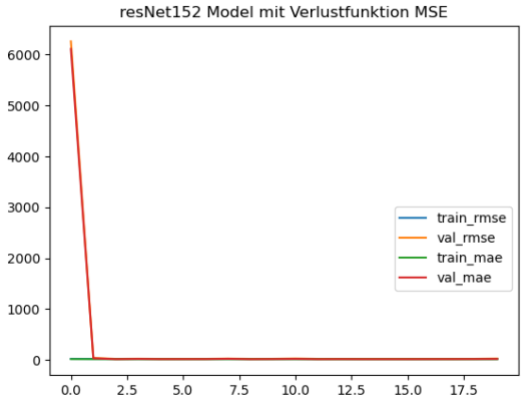


Ähnlich wie bei dem Modell mit ResNet101 und der Verlustfunktion MSE variierte bei demselben Modell mit der Verlustfunktion MAE die RMSE-Metrik bei der Validierung von etwa 21 bis etwa 24, während die RMSE des Trainings bei ca. 21 liegt. Die MAE-Metrik der Validierung wurde von 15-16 variiert, während die Trainings-MAE bei etwa 15 bleibt.

Insgesamt scheint sich das Training nicht signifikant verbessert zu haben und es hat auch einen Overfit.

Es ist auch zu bemerken, dass ResNet101 die doppelte Laufzeit wie Resnet50 benötigte und damit 4-mal länger als das Self- Trained Modell.

4.2.3 ResNet152



In der ersten Epoche sind die Metriken wieder recht hoch, was darauf hinweist, dass die Vorhersagen des Modells recht weit von den wahren Werten entfernt sind. Das Trainingsprotokoll sieht wie folgt aus.

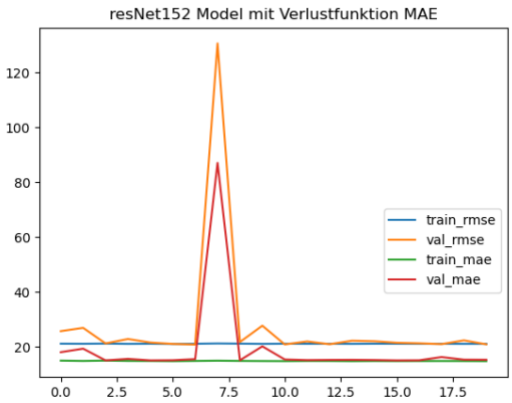
```
Epoch 1/20

loss: 494.6186 - rmse: 22.2400 - mae: 16.7066 - val_loss: 39199100.0000 - val_rmse: 6260.9185 - val_mae: 6111.9097


Epoch 2/20
loss: 444.5023 - rmse: 21.0832 - mae: 15.7592 - val_loss: 1497.4055 - val_rmse: 38.6963 - val_mae: 32.5449
Epoch 3/20
loss: 432.1321 - rmse: 20.7878 - mae: 15.6466 - val_loss: 436.2681 - val_rmse: 20.8870 - val_mae: 15.5271
Epoch 4/20
loss: 430.3770 - rmse: 20.7455 - mae: 15.5544 - val_loss: 564.1239 - val_rmse: 23.7513 - val_mae: 20.3435
Epoch 5/20
loss: 428.7589 - rmse: 20.7065 - mae: 15.4941 - val_loss: 437.4680 - val_rmse: 20.9157 - val_mae: 16.0875
Epoch 6/20
loss: 427.0966 - rmse: 20.6663 - mae: 15.4678 - val_loss: 446.4165 - val_rmse: 21.1286 - val_mae: 16.6479
Epoch 7/20
loss: 425.8339 - rmse: 20.6357 - mae: 15.4613 - val_loss: 454.0388 - val_rmse: 21.3082 - val_mae: 16.8340
Epoch 8/20
loss: 428.8095 - rmse: 20.7077 - mae: 15.5457 - val_loss: 678.2455 - val_rmse: 26.0431 - val_mae: 22.7615
Epoch 9/20
loss: 426.9992 - rmse: 20.6640 - mae: 15.4944 - val_loss: 468.3849 - val_rmse: 21.6422 - val_mae: 15.1110
Epoch 10/20
loss: 429.0671 - rmse: 20.7139 - mae: 15.5301 - val_loss: 490.4586 - val_rmse: 22.1463 - val_mae: 17.0953
Epoch 11/20
loss: 427.0984 - rmse: 20.6664 - mae: 15.4763 - val_loss: 789.2004 - val_rmse: 28.0927 - val_mae: 20.6207
Epoch 12/20
loss: 427.3842 - rmse: 20.6733 - mae: 15.4477 - val_loss: 460.6877 - val_rmse: 21.4636 - val_mae: 15.0917
Epoch 13/20
loss: 425.9406 - rmse: 20.6383 - mae: 15.4568 - val_loss: 444.1500 - val_rmse: 21.0749 - val_mae: 16.4810
Epoch 14/20
loss: 423.7513 - rmse: 20.5852 - mae: 15.4371 - val_loss: 452.3541 - val_rmse: 21.2686 - val_mae: 15.1210
Epoch 15/20
loss: 425.6556 - rmse: 20.6314 - mae: 15.4409 - val_loss: 440.1859 - val_rmse: 20.9806 - val_mae: 15.1649
Epoch 16/20
loss: 423.0790 - rmse: 20.5689 - mae: 15.4246 - val_loss: 456.1893 - val_rmse: 21.3586 - val_mae: 15.1209
Epoch 17/20
loss: 421.9630 - rmse: 20.5417 - mae: 15.4243 - val_loss: 433.6805 - val_rmse: 20.8250 - val_mae: 15.9230
Epoch 18/20
loss: 427.3895 - rmse: 20.6734 - mae: 15.5064 - val_loss: 440.3328 - val_rmse: 20.9841 - val_mae: 16.2952
Epoch 19/20
loss: 426.6494 - rmse: 20.6555 - mae: 15.4947 - val_loss: 468.8145 - val_rmse: 21.6521 - val_mae: 17.5225
Epoch 20/20
loss: 427.2682 - rmse: 20.6705 - mae: 15.4952 - val_loss: 720.0922 - val_rmse: 26.8345 - val_mae: 19.2458
```

Nach dem Trainingsprotokoll bleiben RMSE und MAE beim Training fast konstant, was darauf hindeutet, dass sich das Modell nicht verbessert hat. Außerdem schwanken RMSE und MAE bei der Validierung stark und unterscheiden sich stark von RMSE und MAE beim Training. Das bedeutet, dass sich das Modell nicht verbessert hat und sogar überangepasst hat.

Es ist auch erwähnenswert, dass die Zeit, die für jede Epoche benötigt wird, variiert, wobei die erste Epoche am längsten dauert (154 Sekunden) und die nachfolgenden Epochen etwa 100 Sekunden benötigen.



Es ist zu erkennen, dass die Bewertungsmetriken während des Trainings nicht nur schwanken, sondern sich auch stark zwischen Training und Validierung unterscheiden, was darauf hindeutet, dass das Modell nicht verbessert und sogar überangepasst hat: Der Trainings-RMSE und MAE liegen bei 21 bzw. 15. Während der Validierungs-RMSE zwischen 20 und 27 schwankt und im schlechtesten Fall 130 beträgt, liegt der Validierungs-MAE zwischen 15 und 20 und im schlechtesten Fall 87.

4.3 Performance- Vergleich

Die nachstehende Tabelle fasst die Performance von 8 Varianten zusammen.

In der ersten Spalte geht es darum, ob die Modelle die Bewertungsmetrik RSME & MAE während des Trainings reduzieren können.

In der zweiten Spalte geht es darum, ob es einen Unterschied zwischen den Bewertungsmetriken RSME und MAE während der Validierung und des Trainings gibt, d.h. ob ein Overfitting vorliegt.

In der letzten Spalte geht es um die Zeit, die für die Epoche benötigt wird.

Modell	Verbesserung im Training	Overfitting	Zeit
Self- Trained & MSE	Ja	Nein	17s
Self- Trained & MAE	Nein	Ja	17s
ResNet50 &MSE	Ja	Nein	45s
ResNet50 &MAE	Nein	Ja	45s
ResNet101 &MSE	Ja	Nein	75s
ResNet101 &MAE	Nein	Ja	75s
ResNet152 &MSE	Nein	Ja	100s
ResNet152 &MAE	Nein	Ja	100s

Die grün markierten Modelle sind die besten Modelle im Rahmen des hier durchgeführten Experiments. Es lässt sich sagen, dass das Self Trained Model in kürzerer Laufzeit eine gute Leistung erbringen konnte. Die Modelle mit ResNet50 sowie ResNet101 als Basismodelle konnten ebenfalls eine nicht schlechte Leistung nachweisen, benötigen aber mehrfache Zeiten. Das Modell mit ResNet152 schneidet in diesem konkreten Experiment am schlechtesten ab. Es braucht nicht nur sehr lange zum Trainieren, sondern es kommt auch immer wieder zu Überanpassungen.

Außerdem kann man sagen, dass die Verlustfunktion MAE in diesem Experiment immer schlechter funktioniert als MSE.

Es ist weiterhin zu beobachten, dass es bei allen untersuchten Modellen immer Schwankungen in den Validierungsmetriken gibt. Bei anderen Modellen, die von anderen Teilnehmern des Wettbewerbs in Kaggle erstellt wurden, ist es ebenfalls der Fall, dass die Metriken in der Validierung stark schwanken, auch wenn die Anzahl der Epochen deutlich über 20 lag. Die möglichen Gründe dafür werden in dem Fazit erläutert.

5 Fazit

5.1 Ergebnisse dieser Arbeit

Wie bereits erwähnt, schnitt das Self-Trained Model selbst nach kürzeren Trainingszeiten genauso gut ab wie ResNet 50 und ResNet101, während RestNet152 zu tief war und zu schlecht für diese Aufgabe abschnitt. Der Grund, warum das Self Trained Modell hier die bessere Wahl sein kann, ist der folgende.

Diese Aufgabe ist zu spezifisch, es geht um die Bildregression, was sich von der Aufgabe unterscheidet, für die die vortrainierten Modelle trainiert wurden. Beim Training von ResNet ging es um den Unterschied zwischen z.B. Tieren und Autos und nicht darum, wie attraktiv ein Tier ist.

Zu dieser Art von Aufgaben kann man auch sagen, dass auch ein Mensch die Attraktivität eines Tieres anhand dieser Bilder nicht genau vorhersagen kann, was mich zu der Annahme bringt, dass es für ein maschineller Lernalgorithmus ebenfalls schwer ist, die Attraktivität eines Tieres vorherzusagen. Das bedeutet, dass das Problem so subjektiv ist, dass es sich nicht leicht auf maschinelle Lernmodelle übertragen lässt.

Dies kann auch erklären, warum es immer wieder Schwankungen bei den Validierungsmetriken gibt. In diesem Fall können die Schwankungen bei den Validierungsverlusten auf unregelmäßige Pawpularity-Werte im Datensatz zurückzuführen sein. Wenn es Schwankungen bei den Validierungsverlusten gibt, diese aber langsam und allmählich abnehmen und schließlich gegen Null konvergieren, kann dies bedeuten, dass das Modell tatsächlich gute Vorhersagen auf den Validierungsdaten macht.

Im Folgenden werden die in dieser Arbeit gewonnenen Erkenntnisse über selbst erstellte Modelle und vortrainierte Modelle vorgestellt.

5.2 Vor- und Nachteile der Self- Trained Modelle

Die Vor- und Nachteile selbst erstellter Modelle sind somit die Folgenden.

Die Vorteile von Self-Trained Modellen sind

- Aufgabenspezifität: Self-Trained Modelle werden speziell für die jeweilige Aufgabe trainiert. Dies führt zu einer besseren aufgabenspezifischen Leistung im Vergleich zu vortrainierten Modellen.
- Keine Verzerrung: Self-Trained Modelle werden auf den spezifischen Daten trainiert, wodurch das Risiko einer Verzerrung durch vortrainierte Daten verringert wird.
- Flexibel: Self-Trained Modelle können auf allen verfügbaren Daten trainiert werden, unabhängig von deren Größe oder Annotationen.

Die Nachteile von Self-Trained Modellen sind

- Erhöhter Rechenaufwand: Self-Trained Modelle benötigen im Vergleich zu vortrainierten Modellen mehr Rechenressourcen und Zeit zum Trainieren.
- Risiko der Überanpassung: Self-Trained Modelle haben ein höheres Risiko der Überanpassung, da sie speziell für eine bestimmte Aufgabe trainiert werden und nicht für eine Vielzahl von Aufgaben und Daten wie vortrainierte Modelle.

5.3 Vor- und Nachteile der Pre- Trained Modelle

Die Vor- und Nachteile Pre-Trained Modelle sind somit die Folgenden.

Die Vorteile von Pre-Trained Modelle sind

- Schnelligkeit: Pre-Trained Modelle lassen sich schnell auf eine bestimmte Aufgabe feinabstimmen, was Zeit und Rechenressourcen sparen kann.
- Leistung: Pre-Trained Modelle können eine hohe Genauigkeit liefern, insbesondere wenn die Aufgabe und die vorbereiteten Daten ähnlich sind.
- Transfer-Lernen: Pre-Trained Modelle können für das Transfer-Lernen verwendet werden, bei dem das vorgefertigte Modell auf eine neue Aufgabe feinabgestimmt wird, wobei das bei der ursprünglichen Aufgabe gewonnene Wissen genutzt wird.

Die Nachteile von Pre-Trained Modellen sind

- Begrenzte Anpassungsfähigkeit an die Aufgabe: Pre-Trained Modelle können bei Aufgaben, die sich stark von der ursprünglichen Aufgabe unterscheiden, keine gute Leistung erbringen.
- Verzerrungen: Pre-Trained Modelle können durch die Daten und Aufgaben, auf denen sie ursprünglich trainiert wurden, Verzerrungen verursachen, die die Leistung des fein abgestimmten Modells beeinträchtigen können.
- Datenanforderungen: Pre-Trained Modelle erfordern oft große Mengen an annotierten Daten zum Trainieren, was für einige Aufgaben nicht möglich ist.

5.4 Empfehlungen zur Auswahl geeigneter Methoden für unterschiedliche Situationen.

Die Entscheidung, ob ein vortrainiertes Modell oder ein selbst erstelltes Modell verwendet werden soll, hängt von verschiedenen Faktoren ab. Hier sind einige Dinge, die bei der Entscheidung zu berücksichtigen sind:

Datenmenge: Wenn man über eine große Menge an Trainingsdaten verfügt, kann man in der Regel ein Self-Trained Modell entwickeln, das auf die spezifischen Anforderungen der Anwendung zugeschnitten ist. Stehen jedoch nur begrenzte Trainingsdaten zur Verfügung, kann es sinnvoller sein, ein vortrainiertes Modell zu verwenden, da es bereits mit einem großen Datensatz trainiert wurde.

Komplexität der Aufgabe: Bei einer komplexen Aufgabe, wie z. B. Bilderkennung oder Sprachverarbeitung, kann es schwierig und zeitaufwändig sein, ein Self-Trained Modell zu trainieren, das eine hohe Leistung aufweist. In diesem Fall kann die Verwendung eines Pre-Trained Modells, das bereits für ähnliche Aufgaben trainiert wurde, die bessere Option sein.

Verfügbarkeit von Pre-Trained Modellen: Für einige Anwendungen gibt es möglicherweise keine vortrainierten Modelle, die für die jeweilige Aufgabe relevant sind. In diesem Fall muss man möglicherweise ein Self-Trained Modell erstellen.

Zeit und Ressourcen: Wenn für das Trainieren eines Modells nur wenig Zeit oder weniger Ressourcen (z. B. Rechenleistung und Speicher) zur Verfügung stehen, kann die Verwendung eines Pre-Trained Modells helfen, Zeit und Ressourcen zu sparen und schnellere Ergebnisse zu erzielen.

5.5 Mögliche zukünftige Forschung

Für die künftige Forschung gibt es mehrere Möglichkeiten.

Eine Möglichkeit wäre, mehr Epochen zu gebrauchen, wenn der Rechenaufwand dafür vorhanden ist. Außerdem könnten die Modelle neu aufgebaut werden, um noch bessere Ergebnisse zu erzielen.

In Keras sind auch andere Implementierungen von ResNet-Modellen verfügbar, z. B. ResNet50V2, ResNet101V2 und Res-Net152V2. Die verschiedenen Versionen der ResNet-Modelle unterscheiden sich in ihrer Architektur, wobei die V2-Versionen Änderungen enthalten, um Einschränkungen der V1-Versionen zu beheben.

Es wäre auch interessant, das gleiche Experiment mit anderen Anwendungsfällen wie der Bildklassifizierung durchzuführen. Ein weiterer möglicher Ansatz wäre die Durchführung von Experimenten mit vortrainierten Modellen im Bereich der Textgenerierung.