Digital Applications & Data Management

WS25/26

Modul 10

Dr. Jens Kohl

Roadmap Vorlesung

1. Grundlagen Künstlicher Intelligenz, Machine Learning und Daten
2. Grundlagen Data Science
3. Angewandte Data Science (Teil 1)
4. Angewandte Data Science (Teil 2)
5. Data Science Use Case
6. Grundlagen unüberwachtes Lernen
7. Grundlagen überwachtes Lernen (tabellarische Daten)
8. Machine Learning Use Case
9. Grundlagen überwachtes Lernen (Bilddaten)
10. Transfer Learning Bilddaten und Fallbeispiel
11. Grundlagen Generative AI
12. Prompt Engineering, Agenten
13. Ausblick, Wiederholung, Fragestunde
14. Fragestunde

10. Überwachtes Lernen:

Transfer Learning auf Bilddaten

Was machen wir heute?

Motivation

Wir schauen uns heute an, wie man die Modellgüte verbessern kann bei gleichzeitiger Reduzierung des Aufwands für das Training (Zeit und Geld).

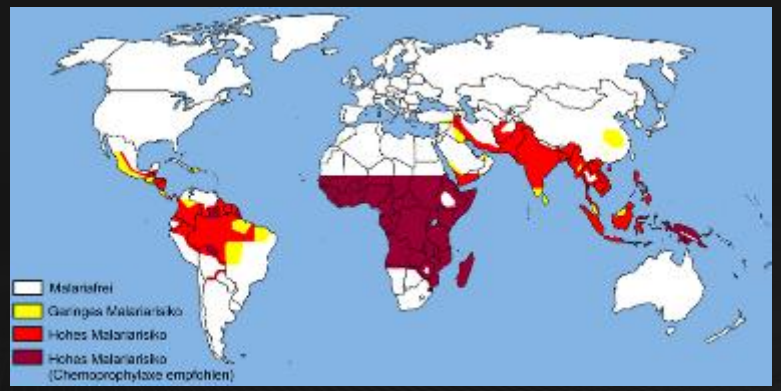
Dies erfolgt, in dem wir Modelle nehmen, die auf Millionen von Bildern vortrainiert werden, und diese an unseren Anwendungsfall anpassen.

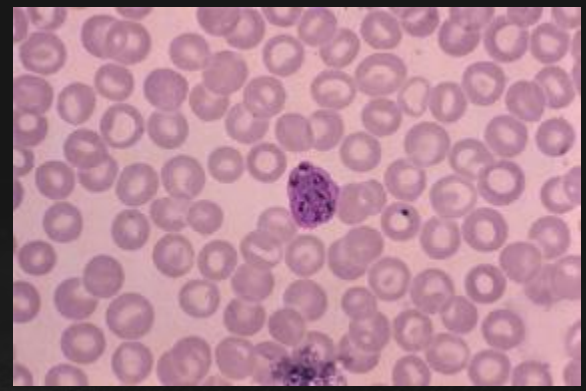
Überwachtes Lernen

Motivation

- Malaria ist eine Infektionskrankheit, die durch den Stich einer Moskito übertragen wird.
- Häufigste Infektionskrankheit mit ca. 200 Mio. erkrankten Menschen pro Jahr.
- Diagnose: kostengünstigste Möglichkeit ist DIE Analyse roter Blutkörperchen auf Plasmodien (einzellige Parasiten) per Mikroskop.

➔ Aber: wie kann man DAS skalieren?





Ziel: Automatisiertes Klassifizieren einer Zelle (Infiziert oder Gesund?) per CNN

Case Study CNN

Schritt 1: Ask an interesting question

Ask an interesting question

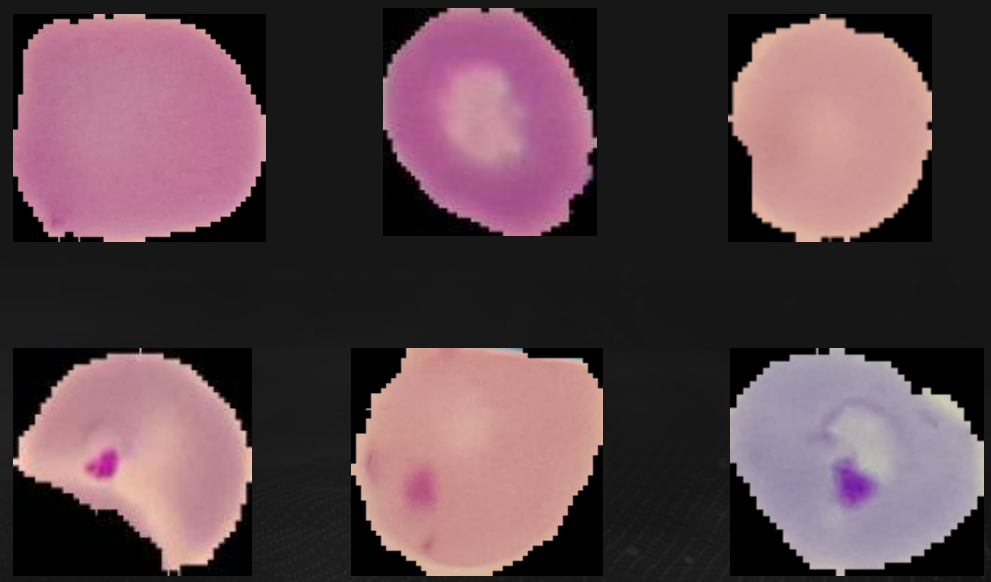
Get the Data

Explore the Data

Model the Data

Communicate/Visualize the Results

„Erkennen Malariaerkennung anhand eines Mikroskopbildes einer Zelle“



Case Study CNN

Schritt 2: Get the data

Ask an interesting question

Get the Data

Daten organisieren

Explore the Data

Model the Data

Communicate/Visualize the Results

Case Study CNN

Schritt 2: Get the data

```
# Standardlibraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Case Study CNN

Schritt 2: Get the data

```
#!wget ftp://lhcftp.nlm.nih.gov/Open-Access-Datasets/Malaria/cell_images.zip
!wget https://data.lhncbc.nlm.nih.gov/public/Malaria/cell_images.zip
!unzip cell_images.zip
```

!WGET: Dateien aus dem Internet laden
!UNZIP: Entpacken der Dateien

Case Study CNN

Schritt 2: Get the data

```
import os
import glob

INFECTED_DIR = '/content/cell_images/Parasitized'
HEALTHY_DIR = '/content/cell_images/Uninfected'

infected_files = glob.glob(INFECTED_DIR + '/*.png')
healthy_files  = glob.glob(HEALTHY_DIR + '/*.png')

print(len(infected_files), "Dateien mit Bildern infizierter Zellen")
print(len(healthy_files), "Dateien mit Bildern gesunder Zellen")
```

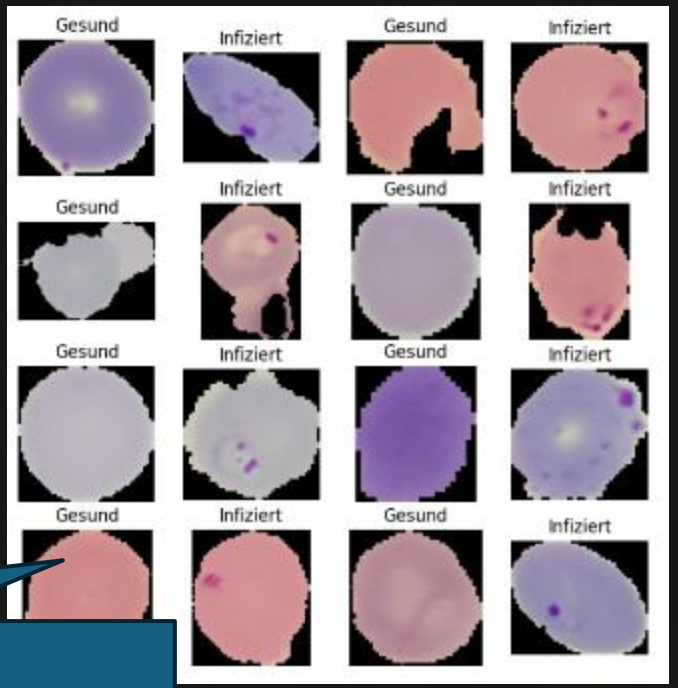
```
13779 Dateien mit Bildern infizierter Zellen
13779 Dateien mit Bildern gesunder Zellen
```

LADEN VON OS UND GLOB (STANDARD LIBRARIES ZUR INTERAKTION MIT DEM BETRIEBSSYSTEM (OS) BZW. ZUM Dateien/ Pfadnamen auFrufen inklusive aller unterverzeichnisse.

Case Study CNN

Schritt 2: Get the data

```
[ ] import numpy as np
    %matplotlib inline

    number_cols = 4
    number_rows = 4
    run_index = 0

    plt.figure(1 , figsize = (8 , 8))

    for i in range(number_cols * number_rows):
      run_index += 1
      cur_plot = plt.subplot(number_cols, number_rows, run_index)

      if run_index % 2 == 0:
        plt.title('Infiziert')
        img = plt.imread(np.random.choice(infected_files)) # wähle ein zufälliges Bild aus der Menge der Bilder
      else:
        plt.title('Gesund')
        img = plt.imread(np.random.choice(healthy_files)) # wähle ein zufälliges Bild aus der Menge der Bilder

      plt.imshow(img)
      cur_plot.axis('Off') # Keine Achsen zeigen

    plt.show()
```



WAS FÄLLT AUF?

- Bilder haben verschiedene Größen.
- Zellen haben verschiedene geometrische Formen.
- Grund für Infektion scheint ein oder mehrere lila Objekte in der Zelle zu sein. Diese können an verschiedenen Stellen in verschiedener Größe vorkommen.

Case Study CNN

Data augmentation

- Data Augmentation: Anwenden verschiedenster Operationen wie Drehen, Farben ändern, verzerren, etc. auf die Trainingsbilder.
- Ermöglicht generieren mehr Trainingsbilder für ein besseres Lernen (so können wir auch Machine Learning auf wenige Bilder starten).
- Macht Modell deutlich robuster, weil es durch die vielen leichten Bildveränderungen das Modell zwingt, diese Veränderungen als mögliche Varianz zu lernen und so mehr zu generalisieren.
- Einziger Nachteil: spezifische Verzeichnisstruktur notwendig (und Training dauert länger).
- Verzeichnisstruktur immer gleich:
  - je einen Ordner für Trainings-, Validierungs- und Testdaten.
  - In jedem dieser Ordner ist Unterordnerstruktur, die Anzahl und Namen der Zielklassen abbildet
  - In dieser Struktur sind dann die passenden Bilder.
- Wir müssen also die gesamte Anzahl Bilder für infizierte und gesunde Zellen gemäß der Klasse aufteilen und dann erneut in das bekannte Verhältnis Train zu Val zu Test.
- Dies erledigt die Bibliotheksfunktion Splitfolders praktischerweise für uns!

Case Study CNN

Data augmentation

- Generiert (zufällig) aus vorhandenen Bildern leicht veränderte Bilder als Input für das Modell.
- Vorteile:
  - Algorithmus wird gezwungen zu generalisieren, das Modell wird somit robuster.
  - Erstellen größeres Datenset als vorhanden
  - Geschieht im Arbeitsspeicher.
- Nachteile:
  - Erhöhter Trainingsaufwand, da mehr Bilder.
  - Modell kann Features lernen, die real nicht existieren (verzerrte Gesichter, Tiere in

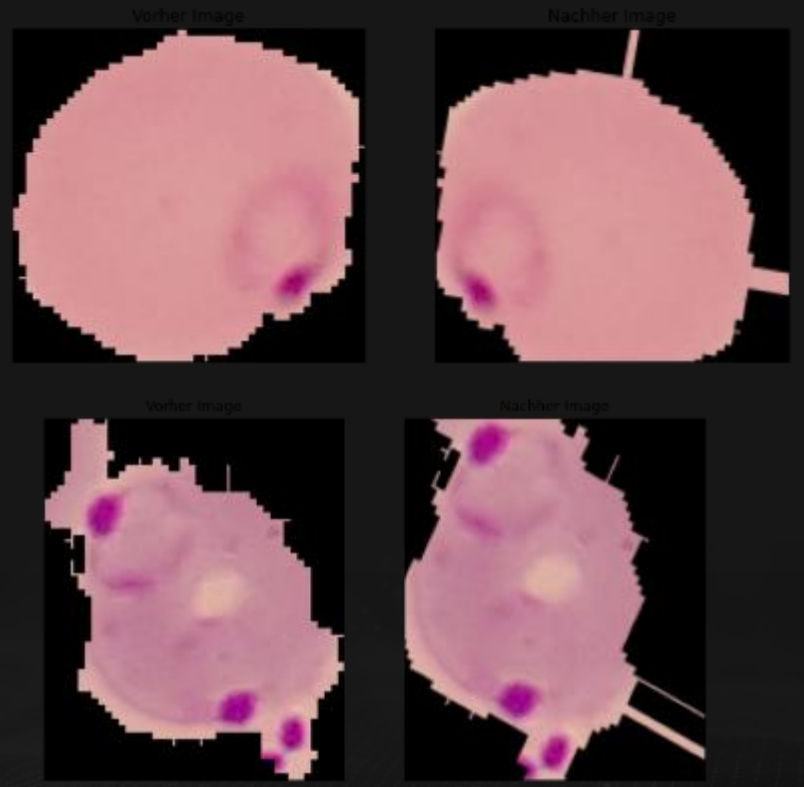


Image Augmentation ermöglicht höhere Modellqualitätauch bei wenigen Daten

Case Study CNN

Data augmentation: Erstellen Verzeichnisstruktur

```
!pip install split-folders tqdm
import splitfolders

input_folder = '/content/cell_images'
splitfolders.ratio(input_folder, output="output", seed=1337, ratio=(.6, .2, .2), group_prefix=None)
```

FÜR NACHHER EINGESETZTE IMAGE AUGMENTATION IST DIE VERZEICHNISSTRUKTUR VORGEGEBEN:

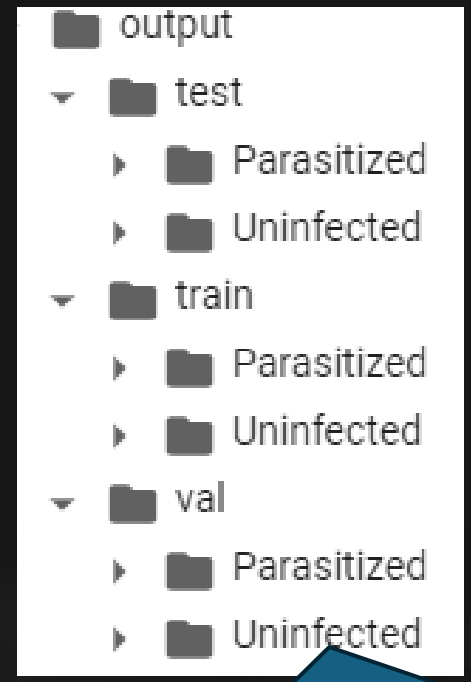
- **train: zum Trainieren**
- **val: zum Überprüfen, wie gut das Modell während des Trainings ist**
- **test: Testen des Modells auf Daten, die das Modell nicht zum trainieren bekommen hat**

In jedem dieser 3 Ordner ist dann jeweils 1 Ordner für die verschiedenen Klassen, die wir vorhersagen wollen (in unserem Fall: infected und non-infected)

Case Study CNN

Data augmentation: Erstellen Verzeichnisstruktur

VORHER:
13779 BILDER MIT INFIZIERTEN ZELLEN
13779 BILDER MIT GESUNDEN ZELLEN



NACHHER:
5512 Testbilder
16534 Trainingsbilder
5512 Validierungsbilder

Case Study CNN

Data augmentation: Image augmentation

Image Augmentation Trainingsdaten

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_image_gen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest')
```

Ziel ist, bei den Trainingsdaten durch die Wahl der möglichen Operatoren für Augmentation verschiedensten Bilder zu erstellen, die real möglich sind!

Image Augmentation Testdaten

```python
test_image_gen = ImageDataGenerator(rescale=1./255)
```

Bei Test- und Validierungsbildern machen wir nur die Normalisierung durch Teilen per 255. Sonst nichts, denn es sollen ja die realen Bilder erkannt werden

Case Study CNN

Data augmentation: Image augmentation generator

Image Augmentation Trainingsdaten

```
train_generator = train_image_gen.flow_from_directory(
    "/content/output/train",
    target_size=(150, 150),  # Alle Bilder haben Einheitsgröße 150 x 150
    color_mode = 'rgb',
    batch_size=256,
    class_mode='binary',
    shuffle=True)
```

Train_generator ist **Pipeline,** um Augmentations auszuführen:

- Erster Parameter: Bilderverzeichnis, auf das wir die Augmentation ausüben wollen.
- Target_size: definiert die Größe der Bilder.
- Batch_size : wie viele Bilder trainiert & getestet werden, bevor Modellupdate erfolgt.
- Class_mode definiert, daß wir nur 2 Klassen haben (krank/ gesund),
- Shuffle: zufälliges mischen.

Image Augmentation Testdaten

```
validation_generator = test_image_gen.flow_from_directory(
    "/content/output/val",
    target_size=(150, 150),
    batch_size=256,
    color_mode = 'rgb',
    class_mode='binary',
    shuffle=True)

test_generator = test_image_gen.flow_from_directory(
     "/content/output/test",
     target_size=(150, 150),
     batch_size=256,
     color_mode = 'rgb',
     class_mode='binary',
     shuffle=True)
```

Case Study CNN

Data augmentation: Image augmentation results.

```
Found 16534 images belonging to 2 classes.
Found 5512 images belonging to 2 classes.
Found 5512 images belonging to 2 classes.
```

```
[ ]  print(train_generator.class_indices)

     {'Parasitized': 0, 'Uninfected': 1}
```

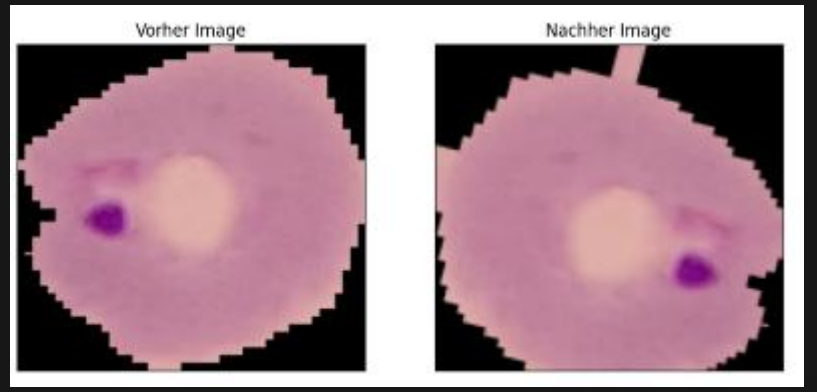
Case Study CNN

Data augmentation: view random augmentations.

Image Augmentation Trainingsdaten

```
example_infected_cell = np.random.choice(infected_files)

plt.figure(1, figsize = (10 , 7))

# Originalbild
plt.subplot(1 , 2 , 1)   # das sagt in Zeile 1 2 Bilder und jetzt wird das erste geplottet
img = plt.imread(example_infected_cell)
plt.imshow(img)
plt.title('Vorher Image')
plt.xticks([]) , plt.yticks([])

# Bild nachher
plt.subplot(1 , 2 , 2) # das sagt in Zeile 1 2 Bilder und jetzt wird das zweite geplottet
modified_img = plt.imread(example_infected_cell)
plt.imshow(train_image_gen.random_transform(modified_img))
plt.title('Nachher Image')
plt.xticks([]) , plt.yticks([])
```

Image Augmentation Testdaten



Case Study CNN

Schritt 4: model the data

Ask an interesting question

Get the Data

Explore the Data

Model the Data

Communicate/Visualize the Results

Definieren, Trainieren, Optimieren und Testen Modell

Case Study CNN

Schritt 4: model the data - Vorgehensweise

- Definieren des Modells und seiner Modellelemente.
  Aufgabe Modell ist das Lernen, was kranke von gesunden Zellen unterscheidet (bspw. der lila punkt in der zelle)
- Definieren Hyperparameter (Trainingsdauer, Batch size, Optimierungsverfahren, ...)
- Modell kompilieren
- Modell trainieren
- Modell optimieren: Schritt 1-4 bis Metriken gut genug sind.

Case Study CNN

Schritt 4: Transfer Learning – was ist die Idee?

- Die Genauigkeit von Deep Learning Verfahren hängt von der Größe und der Datenqualität der Trainingsmenge sowie von der – oft durch Ausprobieren– verwendeten Struktur des Netzes ab.
- An dieser Stelle ermöglicht Transfer Learning deutliche Zeitersparnis und Effizienzgewinne durch das Verwenden eines schon auf Millionen von Daten vortrainierten Modells.
- Idee von Transfer Learning ist, ein vortrainiertes Modell bis auf die letzten 1-2 Schichten direkt zu übernehmen, in denen die Klassifikation erfolgt.
- Die letzten beiden Schichten (meist aber nur die letzte Schicht) werden auf das vorliegende Problem angepaßt – und dann das modifizierte Modell auf die Traningsmenge erneut trainiert.

Case Study CNN

Schritt 4: Transfer Learning

1. Auswahl Transfer Learning Modells und konfigurieren
   - „Merken“ der letzten Schicht des gewählten Modells.
   - Anhängen der gewünschten Layer an die gemerkte, letzte Schicht.
   - Alle Schichten des Modells auf nicht trainierbar setzen und dann hinzugefügte Schichten auf trainierbar setzen.
2. Daten-Pipeline für Image Augmentation aufsetzen
3. Hyperparameter für Training definieren
4. Modell kompilieren.
5. Modell trainieren.
6. Modellgüte testen

Case Study CNN

Schritt 4: Transfer Learning

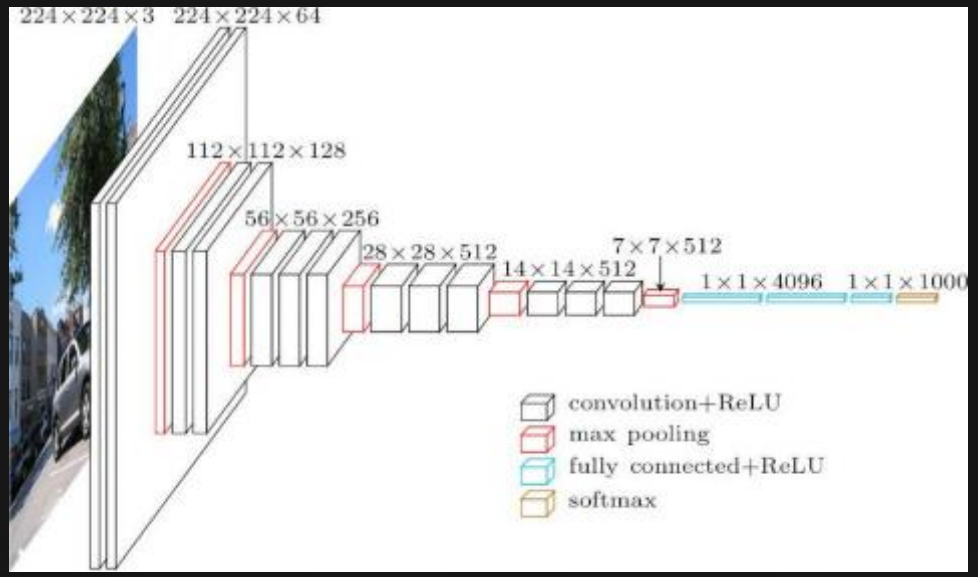
- Rechte Seite zeigt verfügbare, vortrainierte Modelle.
- Top1- und Top5-Genauigkeit bezieht sich auf die Performance Modell bei Benchmark-Datensatz:
  - Top 5: Modell hat 5 „Versuche“; wie oft ist zu erkennendes Objekt darunter
  - Top 1: Modell hat 1 Versuch für Erkennen,
- Parameters: je mehr Parameter, desto mehr Zeit zum Lernen wird benötigt. Je besser kann das Modell aber werden
- Tiefe: je tiefer, desto mehr Zeit wird zum Lernen benötigt, aber Modell kann komplexere Sachen erkennen
- Größe: wie groß ist das fertige Modell? Große Modelle können nicht überall eingesetzt werden aufgrund Speicherplatz (bspw. ältere Handys)
- Time per inference step: wie lang Netz für Inferenz braucht (also Auswertung Input in Netz).

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6

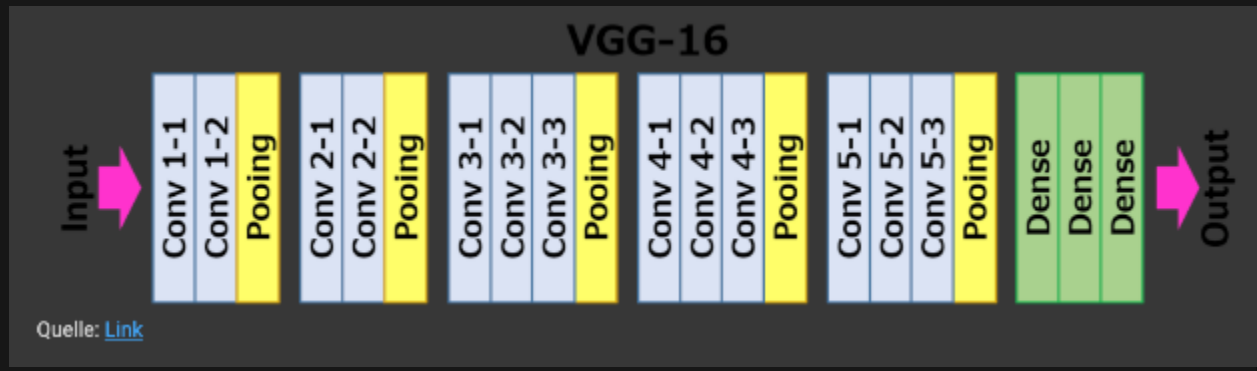
Quelle: https://keras.io/api/applications/

Case Study CNN

Schritt 4: Transfer Learning – Beispiele für Modelle



Quelle: https://keras.io/api/applications/



- Übernahme Bilderkennung in den ersten Schichten, aber Anpassen/ Ersetzen des Klassifikators (grüne Farbe) eigenen Klassifikator.
- Da wir einen neuen Klassifikator haben, müssen wir das Modell auf die vorhandenen Daten trainieren.
- Hierbei sollten wir aber aufpassen, daß nur der geänderte Klassifikator beim Trainieren angepaßt wird. (Sonst überschreiben wir ja das bekannte Wissen, das wir nutzen wollen…)

Case Study CNN

Transfer Learning – Modell auswählen

```python
import tensorflow as tf
import tensorflow.keras   # Keras als Schnittstelle zu den "komplizierteren Umfängen"
from tensorflow.keras import layers
from tensorflow.keras import Model

vgg_new = tf.keras.applications.vgg19.VGG19(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3))
```

VGG16 und MobileNet benötigten als Input-Format für Bilder 224x224x3 → vorher Anpassen mit ImageDataGen (siehe Notebook)

Case Study CNN

Transfer Learning – Modell konfigurieren

```
base_vgg_new = vgg_new
base_out_new = base_vgg_new.output  # speichern des letzten Layers des VGG16 in base_out_new. Hier kommen die neuen Layer dran

# ab hier hängen wir weitere Schichten an das Modell an. Das Ziel ist dabei das Modell auf unseren Anwendungsfall anzupassen
flatten = tf.keras.layers.Flatten()(base_out_new)
out = tf.keras.layers.Dense(1, activation='sigmoid')(flatten)  # wir haben ja nur 2 Klassen (infiziert/ gesund)

# koppeln altes modell mit gerade definierten Schichten
VGG_model_new = tf.keras.Model(inputs=base_vgg_new.input, outputs=out)
```

Kein Pooling vor Flatting in der letzten Schicht - analog Aufbau CNN!

Case Study CNN

Transfer Learning – Modell konfigurieren

```
# ACHTUNG: bei VGG bitte SGD als Optimizer nehmen, Adam braucht viel länger für Optimierung!
VGG_model_new.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.0001),
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
```

Case Study CNN

Transfer Learning – Modell konfigurieren

```
# explizit setzen aller Schichten auf nicht trainierbar.
for layer in VGG_model_new.layers:
  layer.trainable = False

# aber der letzte, neu hinzugefügte Layer, ist trainierbar. Sonst hätte das ganze ja nichts gebracht
for layer in VGG_model_new.layers[-1:]:
  layer.trainable = True
```

```
print("Total Layers:", len(VGG_model_new.layers))
print("Total trainable layers:", sum([1 for l in VGG_model_new.layers if l.trainable]))
```

```
Total Layers: 24
Total trainable layers: 1
```

Zeigt, ob das Definieren und Konfigurieren Transfer Learning funktioniert hat. Hier müssen bei Total Layers mehrere Layers stehen, bei Total trainable aber nur 1!!

Case Study CNN

Transfer Learning – Modell konfigurieren

```
VGG_model_new.summary()
```

block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_2 (Flatten)	(None, 25088)	0
dense_4 (Dense)	(None, 1)	25089

Auszug Struktur transferiertes Modell: Man sieht die neue Struktur. Die Layer sind bis auf die letzten zwei Zeilen der Tabelle vom VGG16, die letzten beiden - Flatten 1 und dense_3 - sind neu hinzugekommen.

Case Study CNN

Transfer Learning – Data pipelines für Image Augmentation generieren

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_image_gen_VGG = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest')
```

```python
test_image_gen_VGG = ImageDataGenerator(rescale=1./255)

train_generator_VGG = train_image_gen_VGG.flow_from_directory(
    "/content/output/train",
    target_size=(224, 224),
    color_mode = 'rgb',
    batch_size=64,
    class_mode='binary',
    shuffle=True)

validation_generator_VGG = test_image_gen_VGG.flow_from_directory(
    "/content/output/val",
    target_size=(224, 224),
    batch_size=64,
    color_mode = 'rgb',
    class_mode='binary',
    shuffle=True)
```

Train_generator ist **Pipeline**, um Augmentations auszuführen:

- Erster Parameter: Bilderverzeichnis, auf das wir die Augmentation ausüben wollen.
- Target_size: definiert die Größe der Bilder. VGG benötigt Größe 224x224
- Batch_size : wie viele Bilder trainiert & getestet werden vor Aktualisierung Modell.
- Class_mode definiert, daß wir nur 2 Klassen haben (krank/ gesund),
- Shuffle: zufälliges Mischen der Bilder.

Case Study CNN

Transfer Learning – Hyperparameter

```
# Modell Callback 1: überwache die Validation accuracy. Falls diese nicht nach 3 Durchläufen besser wird, brich das Training ab.
mcp_early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    mode='max',
    patience=3,
    verbose=1)

# Modell Callback 2: speichere immer das beste Modell in das Verzeichnis /content/best_models/vanilla_cnn
mcp_save_best = tf.keras.callbacks.ModelCheckpoint(
    '/content/best_models/vanilla_cnn/vanilla.weights.h5',
    save_best_only=True,
    monitor='val_accuracy',
    save_weights_only=True,
    mode='max')
```

Callbacks erlauben benutzerspezifische Eingriffe ins Training zu jeder Zeit:
Hier verwenden wir 2 Callbacks:

- early_ stopping: höre das Trainieren auf, wenn nach patience=3 Iterationen die Genauigkeit nicht besser wird → spart Zeit & Kosten.
- save_best: prüfe nach jeder Epoche ob das Modell besser wurde. Falls ja, dann speichere die Gewichte. Hilft, wenn Trainingsprozess abstürzt.

Case Study CNN

Transfer Learning – Trainieren

```
historyVGG = VGG_model_new.fit(
    train_generator_VGG,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=validation_generator_VGG,
    callbacks=[mcp_save_best, mcp_early_stopping]
    )
```

```
Epoch 1/10
259/259 [==============================] - 327s 1s/step - loss: 0.6715 - accuracy: 0.5957 - val_loss: 0.6413 - val_accuracy: 0.6767
Epoch 2/10
259/259 [==============================] - 318s 1s/step - loss: 0.6207 - accuracy: 0.6982 - val_loss: 0.5854 - val_accuracy: 0.7250
Epoch 3/10
259/259 [==============================] - 316s 1s/step - loss: 0.5446 - accuracy: 0.7647 - val_loss: 0.4803 - val_accuracy: 0.8059
Epoch 4/10
259/259 [==============================] - 320s 1s/step - loss: 0.4303 - accuracy: 0.8217 - val_loss: 0.3615 - val_accuracy: 0.8465
Epoch 5/10
259/259 [==============================] - 324s 1s/step - loss: 0.3193 - accuracy: 0.8732 - val_loss: 0.2654 - val_accuracy: 0.8959
Epoch 6/10
259/259 [==============================] - 337s 1s/step - loss: 0.2544 - accuracy: 0.9049 - val_loss: 0.2211 - val_accuracy: 0.9158
Epoch 7/10
259/259 [==============================] - 337s 1s/step - loss: 0.2205 - accuracy: 0.9212 - val_loss: 0.2206 - val_accuracy: 0.9086
Epoch 8/10
259/259 [==============================] - 316s 1s/step - loss: 0.2036 - accuracy: 0.9269 - val_loss: 0.1840 - val_accuracy: 0.9329
Epoch 9/10
259/259 [==============================] - 316s 1s/step - loss: 0.1925 - accuracy: 0.9326 - val_loss: 0.1709 - val_accuracy: 0.9390
Epoch 10/10
259/259 [==============================] - 314s 1s/step - loss: 0.1833 - accuracy: 0.9349 - val_loss: 0.1720 - val_accuracy: 0.9499
```

TRAINING DES TRANSFER LEARNING MODELLS:

- Erster Parameter: hole die Trainingsdaten über die definierte Pipeline inkl. Augmentation
- Zweiter Parameter: trainiere die definierte Anzahl Epochen,
- Dritter Parameter: aktualisiere das Modell nach jeder definierten Batchsize,
- Vierter Parameter: als Daten zum Prüfen der Modellgüte nehme die definierte Pipeline für Trainingsdaten
- Fünfter Parameter: verwende die definierten Callbacks

Case Study CNN

Transfer Learning – Modellgüte ansehen

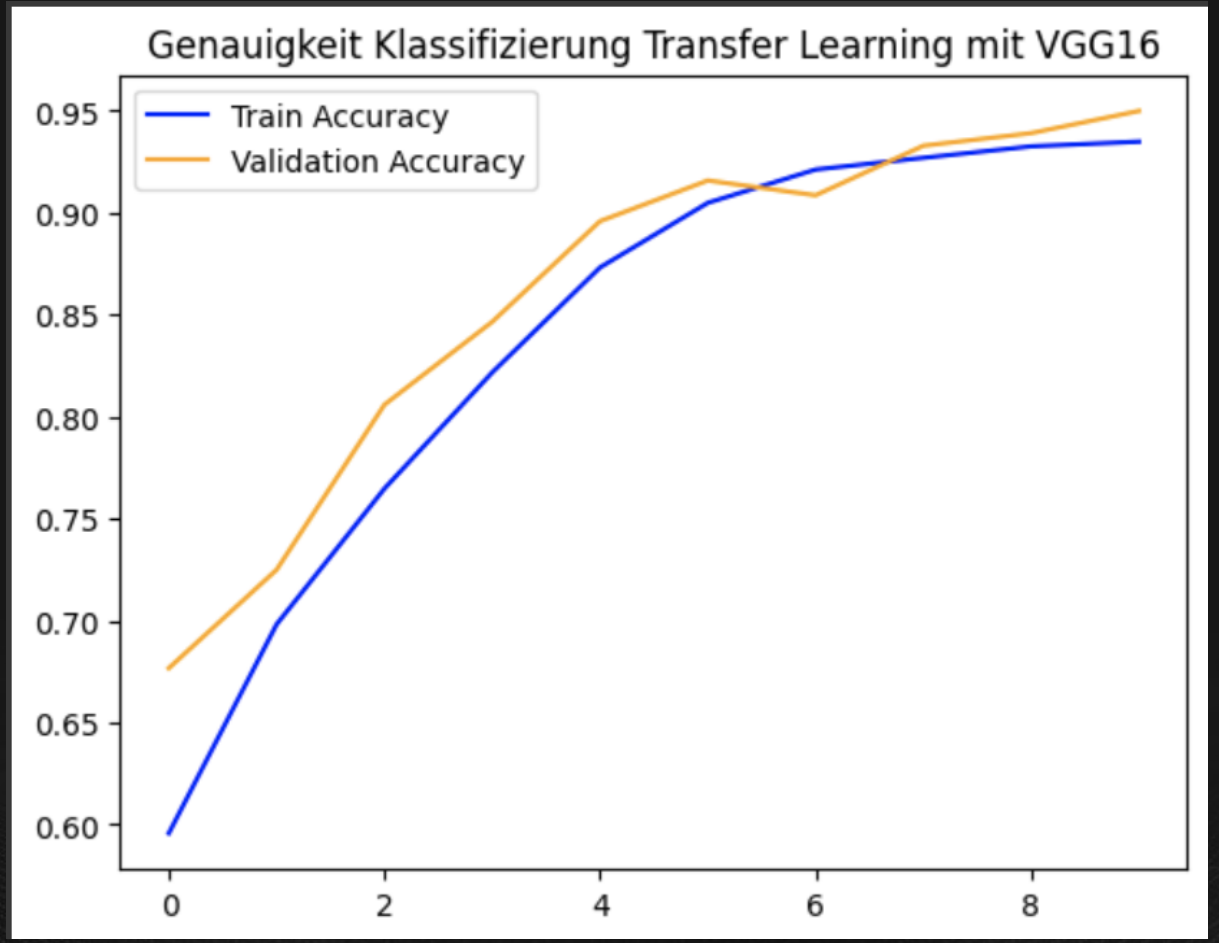
```
# wir laden jetzt das beste Modell während des Trainings.
model.load_weights('/content/best_models/vanilla_cnn/vanilla.weights.h5')
```

Nach dem Trainieren laden wir das Modell mit der höchsten Genauigkeit („bestes Ergebnis wird gezählt“)

```
# Plotte Genauigkeit
plt.title('Genauigkeit Klassifizierung Transfer Learning mit VGG16')
plt.plot(historyVGG.history['accuracy'], color='blue', label='Train Accuracy')
plt.plot(historyVGG.history['val_accuracy'], color='orange', label='Validation Accuracy')
plt.legend()
plt.show()
```

Case Study CNN

Transfer Learning – Modellgüte ansehen



Case Study CNN

Transfer Learning – Modell testen

```
# basic_cnn_preds = basic_cnn.predict(test_imgs_scaled, batch_size=256)
# vgg_aug_preds = vgg_augmented.predict(test_imgs_scaled, batch_size=256)

loss_Basic_CNN, accuracy_Basic_CNN = basic_cnn.evaluate(test_generator)
print('Test accuracy :', accuracy_Basic_CNN)
```

```
22/22 [==============================] - 11s 489ms/step - loss: 0.1411 - accuracy: 0.9554
Test accuracy : 0.9553701281547546
```

```
test_generator_VGG = test_image_gen_VGG.flow_from_directory(
    "/content/output/test",
    target_size=(224, 224),
    batch_size=32,
    color_mode = 'rgb',
    class_mode='binary',
    shuffle=True)

loss_VGG, accuracy_VGG = vgg_augmented.evaluate(test_generator_VGG)
print('Test accuracy :', accuracy_VGG)
```

```
Found 5512 images belonging to 2 classes.
173/173 [==============================] - 33s 161ms/step - loss: 0.1735 - accuracy: 0.9459
Test accuracy : 0.9459361433982849
```

Literatur und Quellen

Künstliche Intelligenz:

- Gröner, Heinecke: Kollege KI
- Burkov: The Hundred-Page Machine Learning Book, online verfügbar unter Link
- Nielsen: Neural Networks and Deep Learning, online verfügbar unter Link
- Russel, Norvig: Artificial Intelligence – a modern approach
- Produktentwicklung mit AI:
  - Ameisen: Building Machine Learning Powered Applications: Going from Idea to Product
  - Ng: Machine Learning Yearning, online verfügbar unter Link

Kostenfreie Online-Kurse (bei Interesse):

- Python-Kurse:
  - Python for Everybody (Link)
  - Udacity Python Course (Link)
  - **Coursera Course Deep Learning** (Link)
  - FAST AI (Link)
- **Web-Links:**
- CNN: https://cs231n.github.io/convolutional-networks/
- CNN: https://medium.com/machine-learning-researcher/convlutional-neural-network-cnn-2fc4faa7bb63
- CNN: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491
- CNN: https://www.wandb.com/articles/fundamentals-of-neural-networks

Backup

DATA AUGMENTATION IM DETAIL: AUSGEWÄHLTE OPERATOREN.

Rotation: ImageDataGenerator(rotation_range=90)

Generiert Bilder mit zufällige Rotation um Wert Parameter

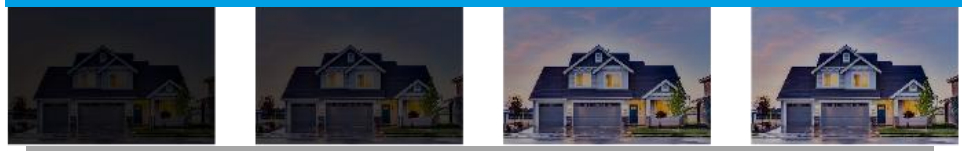
Vert. Schieben:ImageDataGenerator(width_shift_range=0.3)

Gen. Bilder mit zuf. Verschiebung links/ rechts < Parameter

Horiz. Schieben:ImageDataGenerator(heigth_shift_range=0.3)

Gen. Bilder mit zuf. Verschiebung oben/ unten < Parameter

Helligkeit:ImageDataGenerator(brightness_range=(0.1, 0.9))



Gen. Bilder mit zuf. Helligkeitswert Parameter; 0.0 = dunkel

Originalbild

HorizontalFlip: ImageDataGenerator(horizontal_flip=True)

Generiert Bilder inkl. zufälligem horiz. Vertauschen

Vertikaler Flip: ImageDataGenerator(vertical_flip=True)

Generiert Bilder inkl. zufälligem vertik. Vertauschen

Shear Intensity: ImageDataGenerator(shear_range=45.0)

Generiert Bilder durch Stretchen um best. Punkt um X Grad

Zoom: ImageDataGenerator(zoom_range=[0.5, 1.5])

Generiert Bilder per Zoom; < 1 = Vergrößerung

Welche dieser Bilder können in der Realität vorkommen?

Quelle: Link

Vollständige Doku unter: Link
Tutorial unter: Link