Back-Propagation and Algorithms for Training Artificial Neural Networks with TensorFlow

Gero Kauerauf

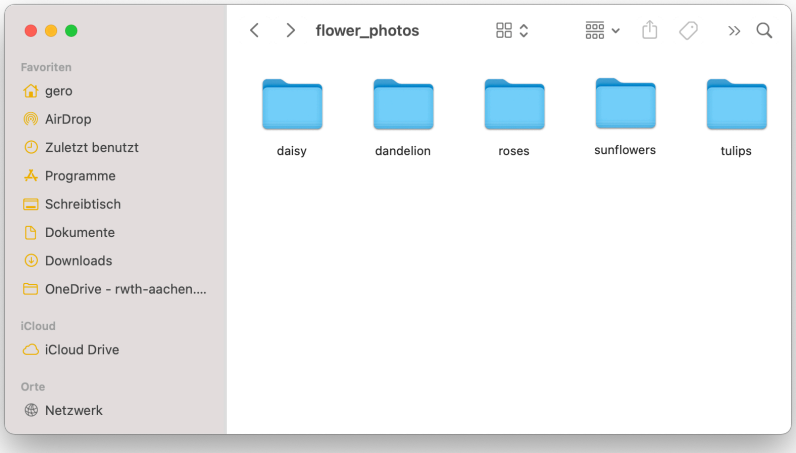
11. Dezember 2020

Überblick

- Bildklassifizierung mit TensorFlow.Keras
  - Was soll das Modell können?
  - Der Datensatz
  - Konstruktion des Python-Programms

Was soll unser Modell können?

- Das Modell soll unterschiedliche Blumen erkennen können
- Der Datensatz besteht aus Blumenbildern, welche von Menschen kategorisiert



wurden

- Es gibt Bilder von Gänseblümchen, Löwenzahn, Rosen, Sonnenblumen und Tuplen
- Mit diesem Datensatz kann ein Modell trainiert werden, um Blumen unterscheiden zu können

Datensatz

- Beispielbilder.

Datensatz

- Zuerst laden wir den Datensatz
- Dazu definieren wir zuerst die Bildgrößen und die Batch-size

```
batch_size = 32
img_height = 180
img_width = 180
```

- Danach wird der Datensatz in einen Trainings- und einen Validierungssatz aufgeteilt
- Mit dem Trainingssatz wird dann das Modell trainiert
- Mit dem Validierungssatz wird das Modell nach dem Training beurteilt

Datensatz

- Dazu nutzen wir die Methode `image_dataset_from_directory` aus `tf.keras.preprocessing`

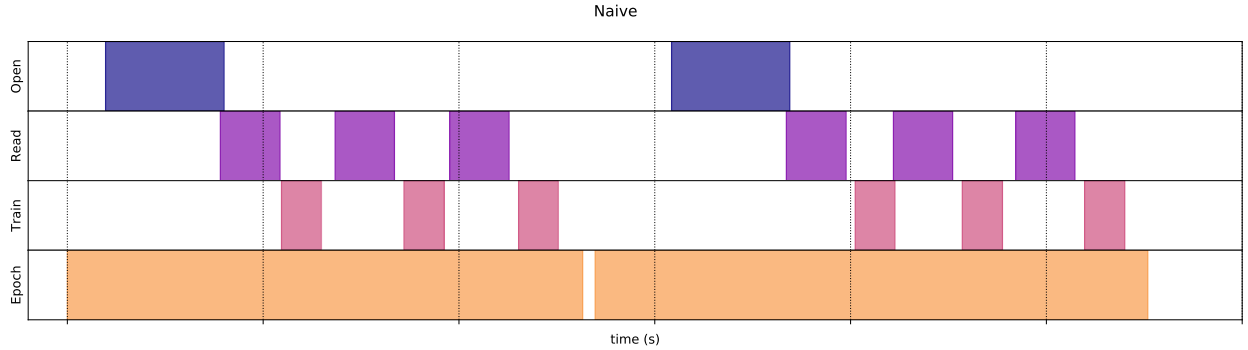
```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size,
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size,
)

class_names = train_ds.class_names
```

Leistungsoptimierung

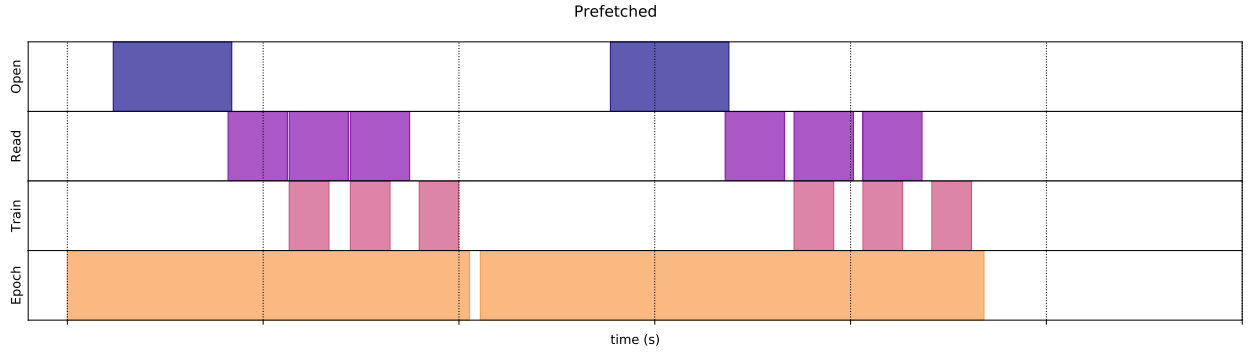
- Wir verwenden zwei Methoden um die Performance beim Laden der Daten zu verbessern
- `Dataset.cache()`
  - Geladene Bilder werden im RAM gelassen und nicht jedes mal erneut eingeladen
- `Dataset.prefetch()`
  - Datenvorverarbeitung und Modelltraining werden überlappt
  - Im i-ten Trainingsschritt werden die Daten für den den $i + 1$-ten Schritt gelesen.

Leistungsoptimierung

- Ohne prefetching



- Mit prefetching



https://www.tensorflow.org/guide/data_performance

Leistungsoptimierung

- Wir nutzen die experimentelle Methode `AUTOTUNE`
- Diese passt die `buffer_size` von `prefetch` dynamisch zur Laufzeit an

```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

- Des weiteren ist es in der Praxis von Vorteil die Daten zu standardisieren
- Da der RGB-Farbraum das Interval $[0, 255]$ umfasst, können wir die Daten skalieren, indem wir mit einem Faktor von $1./255$ multipilizieren
- Danach liegen die Werte jedes Pixels in $[0, 1]$

- Es gibt nun zwei Möglichkeiten zu standardisieren
  - Wir standardisieren den Datensatz
  - Das erste Layer in unserem Modell skaliert die Eingabe $\leftarrow$ ✓

Sequentielles Modell

- Als nächstes erstellen wir ein sequentielles Modell
- Dazu nutzen wir `tf.keras.Sequential`

```
num_classes = 5
model = Sequential(
    [
        layers.experimental.preprocessing.Rescaling(
            1.0 / 255, input_shape=(img_height, img_width, 3)
        ),
        layers.Conv2D(16, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(num_classes),
    ]
)
```

Sequentielles Modell

- Wir konfigurieren das Modell mit `tf.keras.Sequential.compile`
- Hierbei wählen wir den *Optimierer*, die *Fehlerfunktion* und die *Metrik*

```
model.compile(
    optimizer="adam",
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"],
)
```

- `adam` ist ein stochastisches Gradientenabstiegsverfahren, welches auf adaptiven Schätzungen der Momente ersten und zweiten Grades beruht
- `SparseCategoricalCrossentropy` ist eine Fehlerfunktion für mehrere Labels
- `accuracy` berechnet wie häufig die Klassifizierung durch das Modell mit der tatsächlichen Klasse übereinstimmt

Training des Modells

- Nachdem wir den Datensatz vorbereitet und das Modell konfiguriert haben, beginnt nun das Training

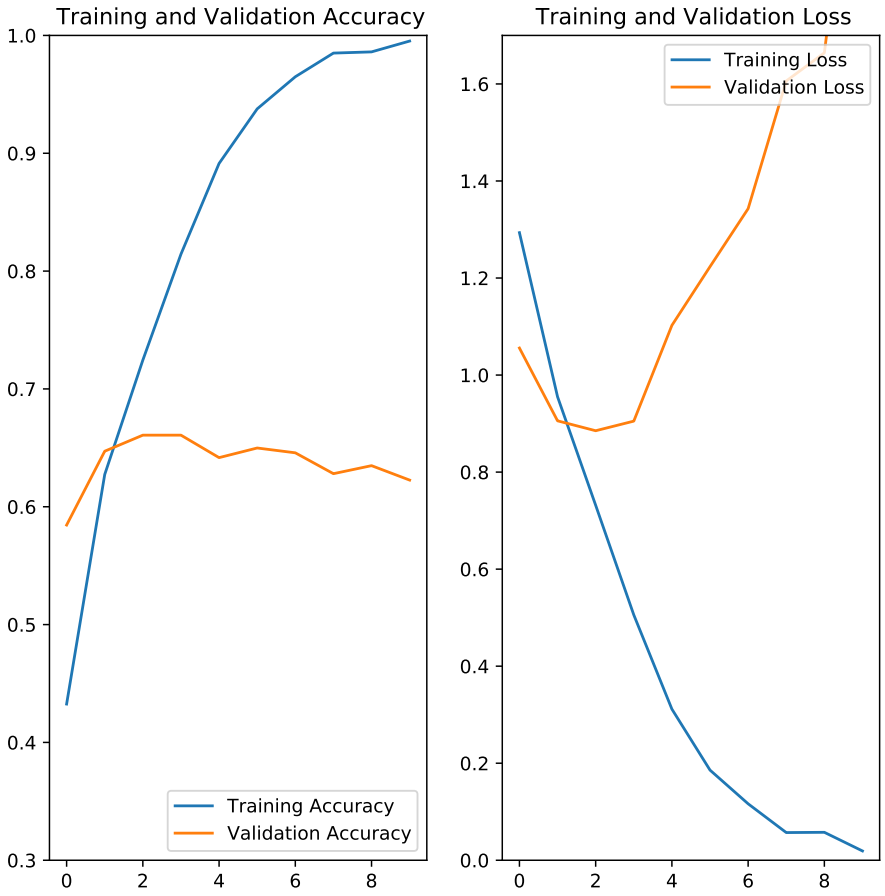
- Wir trainieren das Modell 10 Epochen lang
- Das bedeutet wir iterieren im Training 10-mal über den Trainingsdatensatz

```
epochs = 10
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=epochs)
```

- Nun möchten wir natürlich sehen, wie gut das Training des Modells funktioniert hat
- Dazu zeichnen wir die Trainingsgeschichte mithilfe des Moduls `matplotlib`

Trainingsergebnisse

- Die gezeichnete Grafik enthält Informationen über den Verlauf des Trainings



- Die Genauigkeit der Vorhersagen …
  - … für den Trainingsdatensatz steigt
  - … für den Validierungsdatensatz stagniert

Overfitting

- Der auftretende Effekt wird Overfitting (dt. Überanpassung) genannt
- Aufgrund des kleinen Umfangs unseres Datensatzes lernt das Modell den Trainingsdatensatz im Prinzip „auswendig“
- Es tritt also eine Überanpassung an den Trainingsdatensatz auf, das Modell lernt „ungewünschte“ Details der Bilder

- Wir können das Modell mit zwei Methoden verbessern
  1. Data augmentation (dt. Datenerweiterung)
  2. Dropout (dt. Rauswerfen)

Data augmentation

- Um unseren Datensatz künstlich zu erweitern, können wir bereits im Datensatz enthaltene Bilder „leicht“ verändern
- Dies ist zum Beispiel durch Spiegeln, Rotation oder Zoomen möglich

- Dafür nutzen wir drei experimentelle Schichten aus TensorFlow

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip(
            "horizontal", input_shape=(img_height, img_width, 3)
        ),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomZoom(0.1),
    ]
)
```

Verbessertes Modell

- Zu unserem verbesserten Modell fügen wir nun noch eine `tf.keras.layers.Dropout`-Schicht hinzu

```
model = Sequential(
    [
        data_augmentation,
        layers.experimental.preprocessing.Rescaling(1.0 / 255),
        layers.Conv2D(16, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Dropout(0.2),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(num_classes),
    ]
)
```

Training des verbesserten Modells

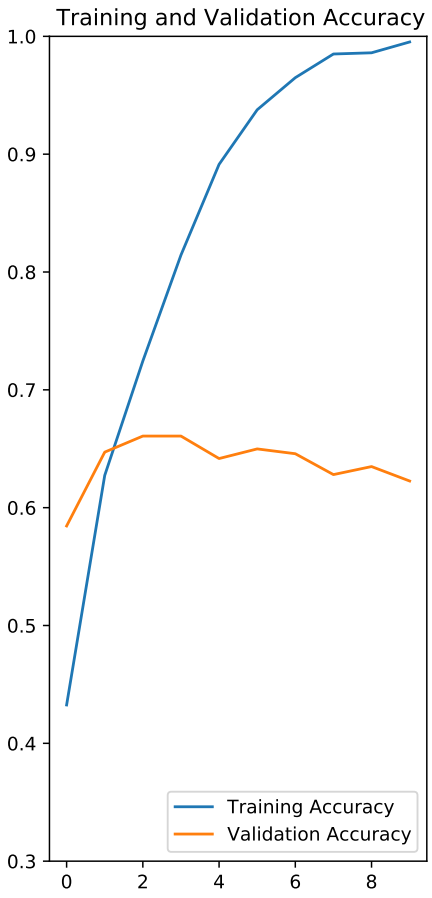
- Nun kompilieren und trainieren wir das verbesserte Modell noch

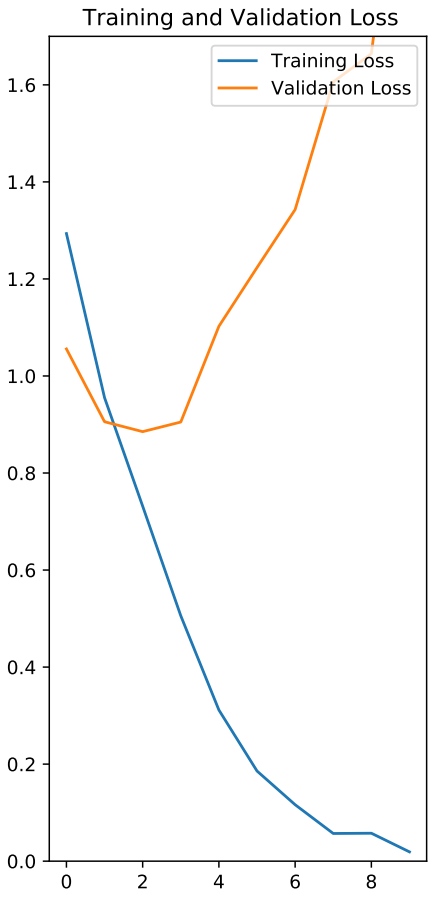
```
model.compile(
    optimizer="adam",
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"],
)
```

```
epochs = 15
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=epochs)
```

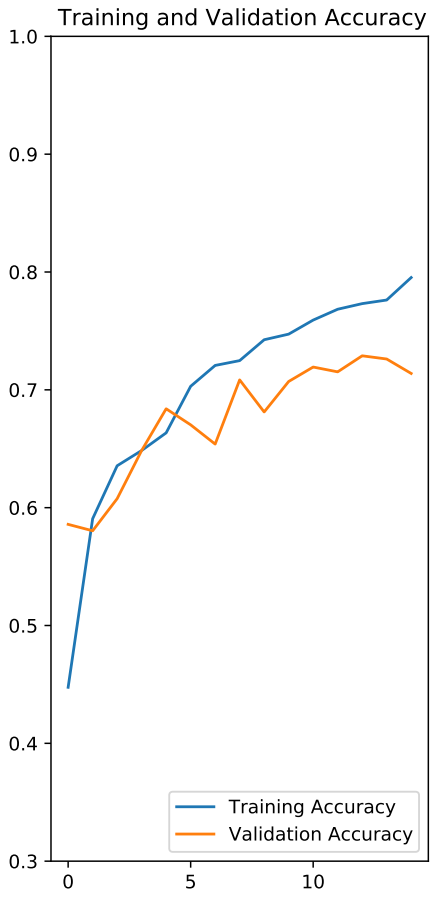
Ergebnis der Verbesserung

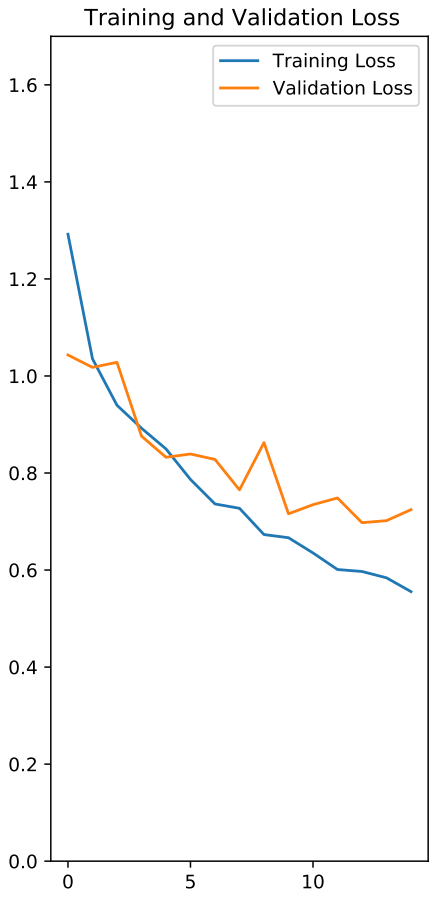
- Vorher





- Nachher





Vorhersagen von neuen Daten

- Jetzt wollen wir natürlich auch Vorhersagen für beliebige (neue) Bilder machen

```
class_names = ["daisy", "dandelion", "roses", "sunflowers", "tulips"]
prediction = model(img_array, training=False)
score = tf.nn.softmax(prediction)

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence.".format(
        class_names[np.argmax(score)], 100 * np.max(score)
    )
)
```

- Live Demonstration.

Fazit

- TensorFlow ist ein *mächtiges* Modul für **Machine Learning**

- Quellen
  - https://www.tensorflow.org/tutorials/images/classification
  - https://www.tensorflow.org/api_docs/python/tf
  - https://www.tensorflow.org/guide/data_performance
  - https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
  - https://arxiv.org/pdf/1511.07122.pdf